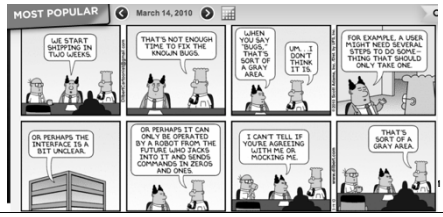
Quality Assurance

Need for a plan
QA Basics
Active reviews

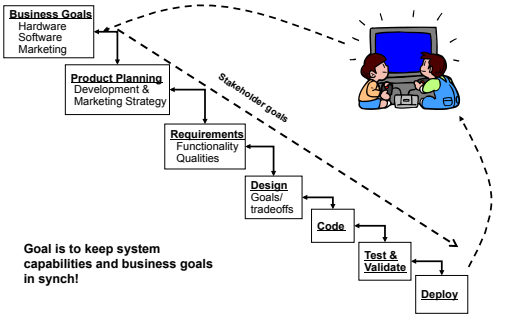


1

View of SE in this Course

- The *purpose of Software Engineering* is to *gain* and *maintain* intellectual and managerial control over the products and processes of software development.
  - **Intellectual control**: able to make rational development decisions based on an understanding of the downstream effects of those choices.
  - **Managerial control** means we likewise control development *resources* (budget, schedule, personnel).

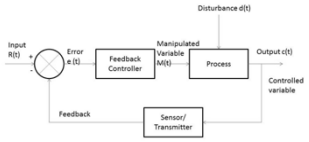
Product Development Cycle



Goal is to keep system capabilities and business goals in synch!

Requires Feedback-Control

- Uncertainty means we cannot get everything under control then run on autopilot
- Rather control requires continuous feedback
  1. Define ideal
  2. Make a step
  3. Measure deviation from idea
  4. Correct direction or redefine ideal and go back to 2



4

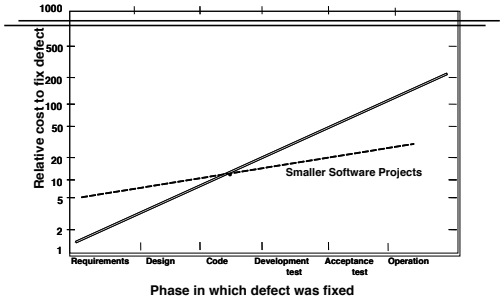
Example: System Requirements

- What happens if we get requirements wrong?
- How do we avoid getting them wrong?
  - What are different ways they can be wrong?
  - How do we check for correctness?
  - How can we maintain correctness over time?
- What is the right time for these activities?
- Who should do the work

QA Questions

- Do the requirements capture what the stakeholders want?
  - Are they correct?
  - Are they complete relative to stakeholder needs?
    - Do they define functional and quality requirements?
- Are they internally complete and consistent?
- What if they change?
- Is the code consistent with the requirements?
- How do we check for these properties?

Increase in Software Cost-to-fix vs. Phase (1976) *



* Barry Boehm - A View of 20th and 21st Century Software Engineering

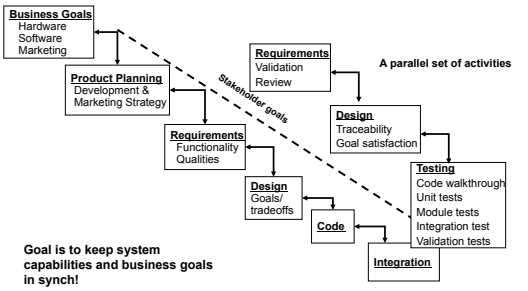
Quality is Cumulative

Requirements Analysis	• Are the requirements valid? • Complete? Consistent? Implementable? • Testable?
Architectural Design	• Does the design satisfy requirements? • Are all functional capabilities included? • Are qualities addressed (performance, maintainability, usability, etc.?
Detailed Design	• Do the modules work together to implement all the functionality? • Are likely changes encapsulated? • Is every module well defined
Coding	• Implement the required functionality? • Race conditions? Memory leaks? Buffer overflow?

We need a plan!

- QA activities are
  - Critical to control (and project success)
  - Part of every phase of the project
  - Time consuming, labor intensive and expensive
    - Potentially unbounded use of resources
    - Consumes significant project resources
  - Cannot do everything, need to choose
- Suggests need to plan QA activities
  - Detect issues as early as possible
  - Target highest priority/risk issues for project
  - Support cost-effective use of resources

Product Development Cycle



Goal is to keep system capabilities and business goals in synch!

QA Plan

- Purpose: synchronize QA activities with project deliverables such that
  - Artifacts satisfy quality goals
  - Delivered code is consistent with stakeholder needs
- The plan make clear how the project will meet overall quality goals
  - The overall QA objectives, strategy, and methodologies
  - The kinds of QA activities that should occur
  - Roles that will carry out the activities
  - When the activities should occur

QA Activities

Verification and Validation

Validation and Verification

- *Validation*: activities to answer the question – "Are we building a system the customer wants?"
  - Familiar activity: customer review of prototype
- *Verification*: activities to answer the question – "Are we building the system consistent with its specifications?"
  - Most familiar verification activity is functional testing
- Both are processes, both have many variations

V&V Methods

- Most applied V&V uses one of two methods
- Review: use of human skills to find defects
  - Pro: applies human understanding, skills. Good for detecting logical errors, problem misunderstanding
  - Con: poor at detecting inconsistent assumptions, details of consistency, completeness. Labor intensive
- Testing: use of machine execution
  - Pro: can be automated, repeated. Good at detecting detail errors, checking assumptions
  - Con: cannot establish correctness or quality
- Tend to reinforce each other

Reviews (1)

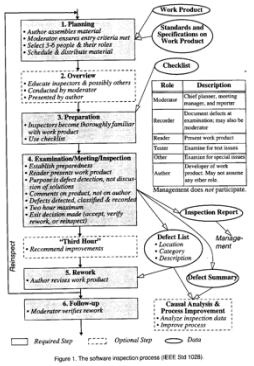
- Informal
  - No explicit process or recording of results
  - "Please read this for me"
    - "This": requirements, architecture or design document, code, test plan, etc.
    - Could be several readers, selected by author
  - Author takes comments and makes revisions as he/she sees fit.

Reviews (2)

- Formal
  - Includes people outside the team
  - Explicit process, results recorded and tracked
  - Examples:
    - Desk check with questions to be answered by a specific date
      - Issues raised by answers recorded
      - Author revises artifact after receiving all answers
      - Revised artifact recirculated among reviewers for consensus
    - Formal Meeting held at a pre-defined time and place (maybe online or by conference call)
      - Reviewers read artifact in advance
      - Comments may or may not be recorded in advance of meeting
      - Facilitator leads discussion of artifact, often on line-by-line basis
      - Issues raised by discussion recorded
      - Author revises artifact after the meeting in response to issues
      - Revised artifact recirculated among reviewers for consensus

Peer Review Process

- Peer Review: a process by which a *software product is examined by peers of the product's authors with the goal of finding defects*
- Why do we do peer reviews?
  - Review is often the only available verification method before code exists
  - Formal peer reviews (inspections) instill some discipline in the review process
- Particularly important for distributed teams
  - Supports communication and visibility
  - Provides feedback on both *quality and understanding*
    - i.e., makes the communication effectiveness and level of understanding visible
  - A good review shows communication is working!



Role	Description
Moderator	Chief planner, meeting manager, and reporter
Recorder	Document defects at examination; may also be moderator
Reader	Present work product
Tester	Examine for test issues
Other	Examine for special issues
Author	Developer of work product. May not assume any other role.

Management does not participate.

Figure 1. The software inspection process (IEEE Std 1028).

18

656

Example: IEEE software inspection process (aka Fagan Inspection)

Active Reviews

Effectiveness of Peer Reviews

- Generally considered *most effective manual technique for detecting defects*
  - Analysis of 12,000 development projects showed defect detection rate of 60-65% for formal inspection 30% for testing
  - Bell-Northern found 1 hour code inspecting saves 2 to 4 hours code testing
  - Effect is magnified in earlier inspections (e.g., 30 times for requirements in one study)
- Means that you should be doing peer reviews, but…
  - Doesn't mean that manual inspections cannot be improved
  - Doesn't mean that manual inspections are the best way to check for every properties (e.g., completeness)
    - Should be one component of the overall V&V process

Peer Review Problems

- Tendency for reviews to be incomplete and shallow
- Reviewers typically swamped with information, much of it irrelevant to the review purpose
- Reviewers lack clear individual responsibility
- Effectiveness depends on reviewers to initiate actions
  - Review process requires reviewers to speak out
  - Keeping quiet gives lowest personal risk
  - Rewards of finding errors are unclear at best

Peer Review Problems (2)

- Large meeting size hampers effectiveness, increases cost
  - Makes detailed discussion difficult
  - Few present reviewers have interest/expertise on any one issue
  - Wastes everyone else's time and energy
- No way to cross-check unstated assumptions

Qualities of Effective Review

- Ensures adequate coverage of artifact in breadth and depth
- Reviewers review only issues on which they have expertise
- Review process is active: i.e., performing the review produces visible output (risk in in doing nothing)
- Individual responsibilities are clear and fulfilling them is evidence of a job well done.

Qualities of Effective Review (2)

- Review process focuses on finding specific kinds of errors.
- Limit meetings to focused groups and purposes requiring common understanding or synergy
  - Permit detailed discussion of issues
  - Expose where assumptions differ

Active Reviews

Goal: Make the reviewer(s) think hard about what they are reviewing

1) Identify several types of review each targeting a different type of error (e.g., UI behavior, consistency between safety assertions and functions).
2) Identify appropriate classes of reviewers for each type of review (specialists, potential users, methodology experts)
3) Assign reviews to achieve coverage: each applicable type of review is applied to each part of the specification

Active Reviews (2)

4) Design review questionnaires (key difference)
   - Define questions that the review must answer by using the specification
   - Target questions to bring out key issues
   - Phrase questions to require "active" answers (not just "yes")
5) Review consists of filling out questionnaires defining
   - Section to be reviewed
   - Properties the review should check
   - Questions the reviewer must answer
6) Review process: overview, review, meet
   - One-on-one or small, similar group
   - Focus on discussion of issues identified in review
   - Purpose of discussion is understanding of the issue (not necessarily agreement)

Examples

- In practice: an active review asks a qualified reviewer to check a specific part of a work product for specific kinds of defects by answering specific questions, e.g.,
  - Ask a designer to check the functional completeness by showing the calls sequences sufficient to implement a set of use cases
  - Ask a systems analyst to check the ability to create required subsets by showing which modules would use which
  - As a developer to check the data validity of a module's specification by showing what the output would be for in-range and out-of-range values
  - Ask a technical writer to check the SRS for grammatical errors
- Can be applied to any kind of artifact from requirements to code

Conventional vs. Active Questions

- **Goal: Make the reviewer(s) think hard about what they are reviewing***
  - **Define questions that the review must answer by using the specification**
  - **Target questions to bring out key issues**
  - **Phrase questions to require "active" answers (not just "yes")**

Conventional Design Review Questions	ActiveBetter Design Review Questions*
Are exceptions defined for every program?	For each access program in the module, what exceptions that can occur?
Are the right exceptions defined for every program?	What is the the range or set of legal values?
Are the data types defined?	For each data type, what are • an expression for a literal value of that data type; • a declaration statement to declare a variable for that type; • the greatest and least values in the range of that data type?
Are the programs sufficient?	Write a short pseudo-code program that uses the design to accomplish (some defined task).

Role of Use Cases

- Use cases or scenarios can be effectively used in active review
- Apply requirements scenarios to verify design against requirements
  - "Show the sequence of program calls that would implement use case C"
  - "Which modules would have to change to add feature F (a likely change)?"
- Conversely, can check properties ask the reviewer to construct scenarios
  - "What sequence of calls would result in an exception E?"

Why Active Reviews Work

- Focuses reviewer's skills and energies where they have skills and where those skills are needed
  - Questionnaire allows reviewers to concentrate on one concern at a time
  - No one wastes time on parts of the document where there is little possibility of return.
- Largest part of review process (filling out questionnaires) is conducted independently and in parallel
- Reviewers must participate actively but need not risk speaking out in large meetings
- Downside: much more work for V&V (but can be productively pursued in parallel with document creation)

Summary

- Need to do reviews to find defects
- Active reviews are more efficient and effective but may take more effort

Testing

- Objectives of software testing
- Principles and testability
- Test case design
- Types of testing
- Testing strategy
- Validation testing and system testing

Testing Objectives

- [Myers79]: Software Testing is the process of executing a program or system with the intent of finding errors.
  - A good test case is one that has a high probability of finding an as-yet undiscovered error
  - A successful test is one that uncovers an as-yet undiscovered error
- [Hetzel88]: It involves any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results.

What is Testing and What is Not?

- Testing is a process for finding semantic or logical errors as a result of executing a program
  - A run-time process, not a compile-time process
- Testing is not aimed at finding syntactic errors
- Testing can reveal the presence of errors, NOT their absence.

"Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence"
- Edsger W. Dijkstra

Software Testing in Practice

- Testing amounts to 40% → 80% of total development costs
  - 40% for information systems
  - 80% for real time embedded systems
- Testing receives the least attention and often not enough time and resources.
- The testers are often forced to abandon testing efforts because changes have been made.
- Testing is (usually) at the end of the development cycle and is often left with little time because other activities have been late

Types of Testing (1)

- White box
  - Based on knowledge of the code within a module
- Operational profile
  - Based on knowledge of how the users (will) use the system
- Black box
  - Based on knowledge of interface (API), available functions, but not of code that implements them
- Soak
  - Based on up-time requirements
- Regression
  - Based on previously run tests

Types of Testing (2)

- Recovery Testing
  - Based on reliability requirements
- Security Testing
  - Based on security requirements
- Stress Testing
  - Based on performance requirements
- Mutation Testing
  - Based on knowledge of expected error types (measure of goodness of test cases)

White Box (1)

- Based on knowledge of the code within a module
- Example:

```
public void displayMessage(int m)
  {
    if (m < 0)
      numField.setText(Integer.toString(-m));
    else
      msgField.setText(Integer.toString(m));
  }
```

- What inputs should I use to ensure that both branches are tested?

White Box (2)

- Goals
  - Code correctly implements interface specification
  - Every non-commentary line of code is executed (statement coverage)
  - Every branch is taken (branch coverage)
  - Every block of code is executed (block coverage)
  - Every path is executed (path coverage)
  - Every exception is thrown
  - Every defined variable is (correctly) used (define-use coverage)
- Tool support
  - Coverage tools, e.g., Cobertura
  - Compile time tools, e.g., Junit
  - Run-time tools, e.g., eclipse debuggers

Black Box (1)

- Based on knowledge of interface (API), available functions, but not of code that implements them
- Example:

push(4).push(-1).push(5).pop.top

- What inputs should I use to ensure that the implementation follows the specification?

Black Box (2)

- Goals
  - For module tests:
    - Returned values conform to syntactic and semantic specifications for the interface
      push(4).push(-1).push(5).pop.top returns -1
    - Inputs beyond parameter bounds, or that violate syntax or semantics, throw exceptions
      pop throws empty stack exception
    - Performance requirements are met (where defined)
  - For integration and system tests
    - Sunny day, rainy day scenarios produce expected results
      –Can be based on use cases
- Tool support: specialized tools for the system

Operational Scenarios (1)

- Based on knowledge of how the users (will) use the system
  - Use cases help
- Example:

Configure the wind speed sensors

- What inputs should I use to mimic typical users?
  - No good specification available for this
  - Users often do the unexpected
  - Best way is to observe and collect data
    - Sit with the users while they're using the system
    - Attend user group meetings
    - Collect data on most-used features (similar to tracking hit counts on pages in a website)
      - Must instrument system to do this
    - Read reviews
    - Conduct surveys

Operational Scenarios (2)

- Goals
  - Usability requirements are met
  - Performance requirements are met
- Tool support
  - Specialized tools for the system: simulators, laboratories
  - Built-in data collection

Soak Tests

- Based on reliability goals
  - Expected up-time
- How can I set up a stream of inputs for the test interval?
  - Example: If a telecommunications switch is expected to run for 6 months with no down time, how long should my soak test run, and how do I generate a typical distribution of phone calls as inputs during the test?
  - Example: If my web server is expected to run for 1 month without crashing, how long should my soak test run, and how do I generate a typical distribution of page hits as input during the test?
- Tool support: Specialized tools for the system: simulators, laboratories
- Goal
  - System will remain in operation without crashing for an extended period of time

RegressionTests

- Based on previously run tests
- How can I be sure that unchanged features still work properly?
  - Example: My FWS was able to produce wind sensor data; can it still do it correctly?
- Goal
  - Features on which the customers depend have not been broken
  - Tool support: Configuration control of test cases

Other Types of Testing (1)

- Performance Testing
  - Based on performance requirements
  - Record performance under varying conditions and see if it matches requirements
- Stress Testing
  - Based on performance requirements
  - Place system under maximum expected load and observe behavior, especially, but not only, performance
- Recovery Testing
  - Based on reliability requirements
  - Force failures and observe system recovery
    - Equipment failures

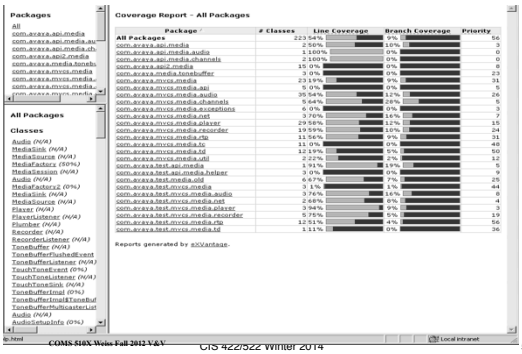
Other Types of Testing (2)

- Security Testing
  - Based on security requirements
  - Attempt to break in
- Mutation Testing
  - Introduce errors into your code and see if test cases discover them
  - A test for goodness of test data

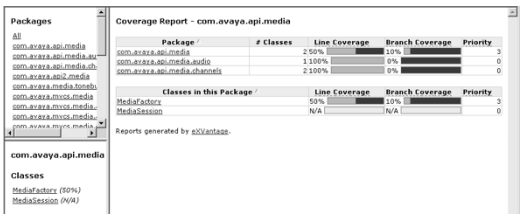
Testing Effectiveness

- Goal: Reduce errors found by customers
  - Question: For different types of coverage, what is the incidence of field errors?
    - Measures: For each type of coverage, number of field errors found for different normalizations as a function of percent coverage
      - Field errors per line of code
      - Field errors per change
      - Field errors per complexity
- Goal: Reduce lifecycle cost
  - Question: What is the cost to obtain different types of coverage during testing
    - Measures: For different types of coverage, what is the cost to obtain the coverage as a function of the percent of coverage

Coverage Report: Packages



Coverage Report: Classes



Coverage Report: Source Code

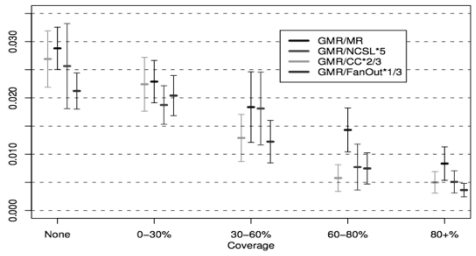
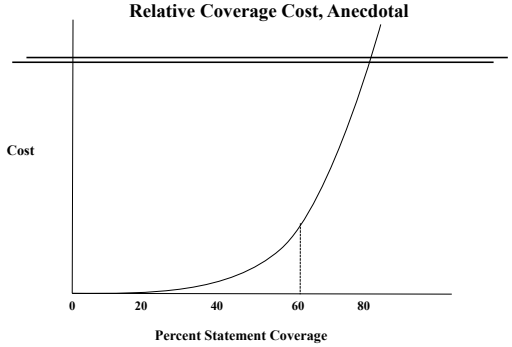


Figure 2. The averages and standard errors of field MR rates for the Avaya project

GMR: field error MR: Modification Report NCSL: Non-commentary source lines CC: cyclomatic complexity

Source: Mockus, Naggapan, Tinh-Drong; Test Coverage and Post-Verification Defects, A Multiple Case Study, 3rd Intntl Symposium on Empirical Software Engineering and Measurement, October, 2009

Relative Coverage Cost, Anecdotal



Outputs From Testing

- List of defects found
  - Modification Requests
- Actions to be taken
  - Usually, revisions of existing artifacts, especially code
- Measures of code quality
  - Normalized defects and MRs
- Measures of test quality and completeness
  - Coverage
  - Number and criticality of errors found
- Training of staff new to a project

Benefits of Software Testing

- Focuses attention on need for error-free software
- Helps ensure that the more obvious and critical kinds of errors will be detected
- Complements other forms of verification, such as design reviews, code inspections, etc.

Software Testing in Practice

- Most companies' new hires are testers.
- Most testing work is manual; help from tools is still limited.
- In many cases, testing is not performed using systematic testing methods or techniques.
- Because no systematic methods or techniques are used, testing is not very effective.
- Sometimes there are "conflicts of interest" between testers and developers.

The Psychology of Testing

- Discouraging factors:
  - a dirty business
  - dumb work/menial work
  - headache undertaking
  - under-funded & too much work
  - too little time & under pressure
  - no good tools
  - not rigorous, systematic
  - no generally accepted principles
  - requires a critic's mentality
  - destructive work

• Encouraging factors:

–Contributing to quality and customer satisfaction
–Tangible evidence of contribution
–Learning details of product operation that few others see
–Necessary part of the software development cycle

"Beware of bugs in the above code; I have only proved it correct, not tried it."
–Donald Knuth

Software Testing Strategies

- Integrate software test case design methods into a well-planned sequence of steps
- A road map for testing
  - test planning, test case design,
  - test execution,
  - result evaluation
  - amount of effort and time
  - resources required (human, equipment, computers, etc.)
- Testing strategies must be customizable
- Testing strategies must support planning and management
  - measurement

Summary: Reviews vs. Testing

- Reviews focus on static analysis of various artifacts
  - Exception: execution of code can be used as part of code reviews
    - As an aid: help to understand what the code does
    - As a precondition: code may be required to be unit tested before review
- Testing focuses on dynamic (execution-time) analysis of code
- Reviews and tests tend to find different types of errors
- Reviews and tests tend to be performed by different roles
- Each uses a variety of different techniques (different families)
- Each uses different skills
- Both are important
- Both have a common goal: develop the right software correctly