Quality Assurance: Building Quality In

The role of “testing”*

Active reviews

*From Prof. Michal Young

What

- Narrow view:
  - Testing is executing a program and comparing actual results to expected results
- Wider view:
  - “Testing” is shorthand for a variety of activities: *anything we can do to check for defects*
  - Dynamic program testing is the most common activity when the artifact is program code
  - Also, reviews, analysis of models, automated checks; we usually need several

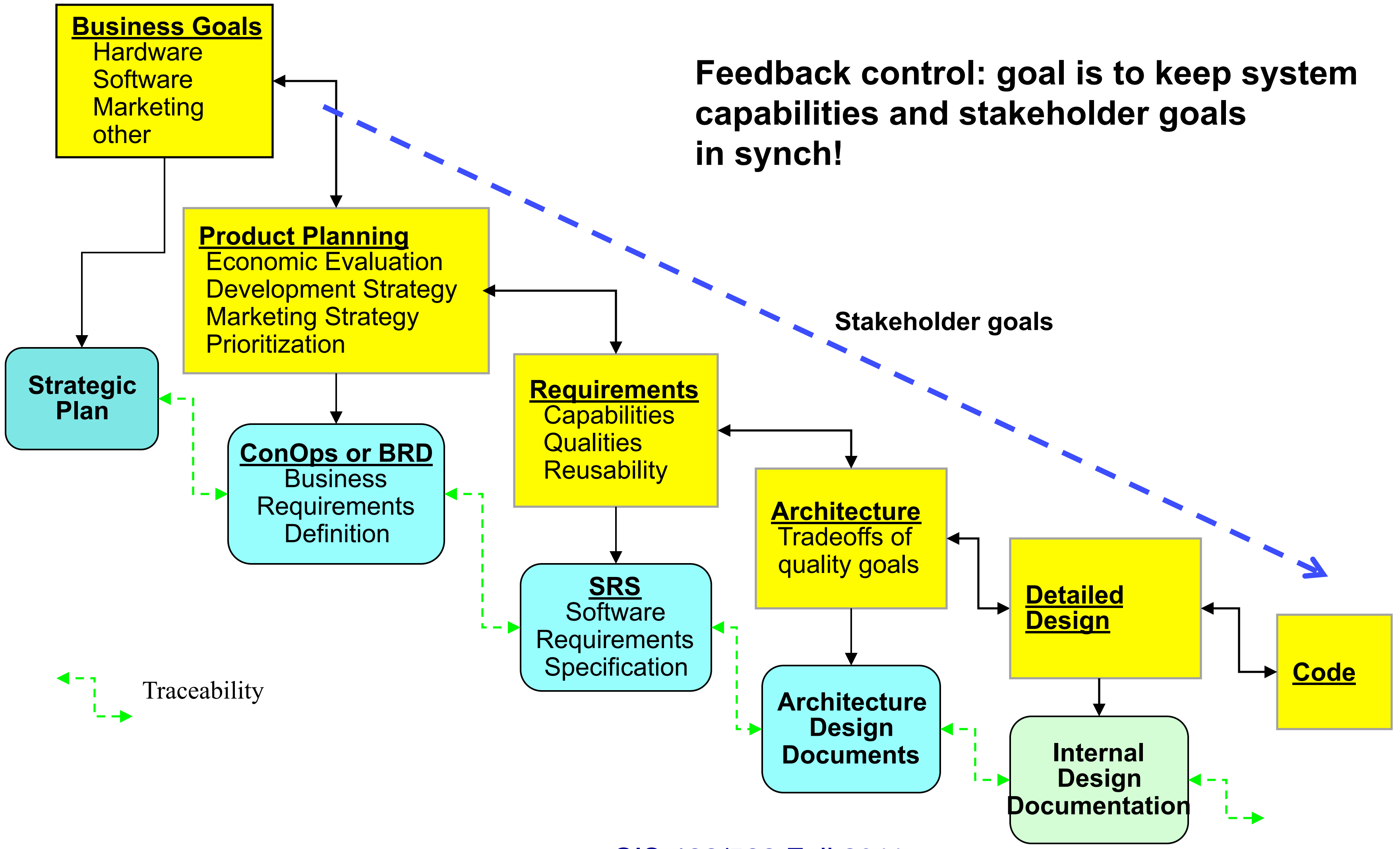
Why Test

- Stupid question?
    - But we need to be clear about goals before we can make reasoned choices regarding the other questions, *who, what, when*, and *how*
    - In general: testing provides the feedback in our "feedback control loop"
- We test to avoid costs
    - Costs during software development
    - Cost of defects in the final product
    - Implies cost/benefit is important

Real meaning of “control”

- What does “control” really mean?
- Can we really get everything under control then run on autopilot?
- Rather control requires continuous feedback loop
  1. Define ideal
  2. Make a step
  3. Measure deviation from idea
  4. Correct direction or redefine ideal and go back to 2

Feedback in the Product Development Cycle

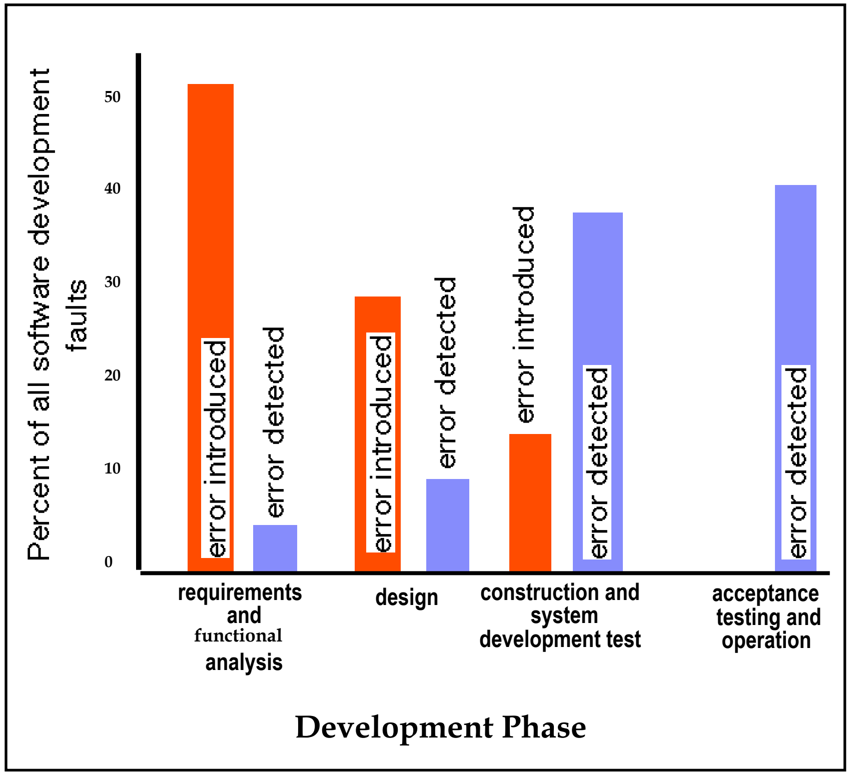


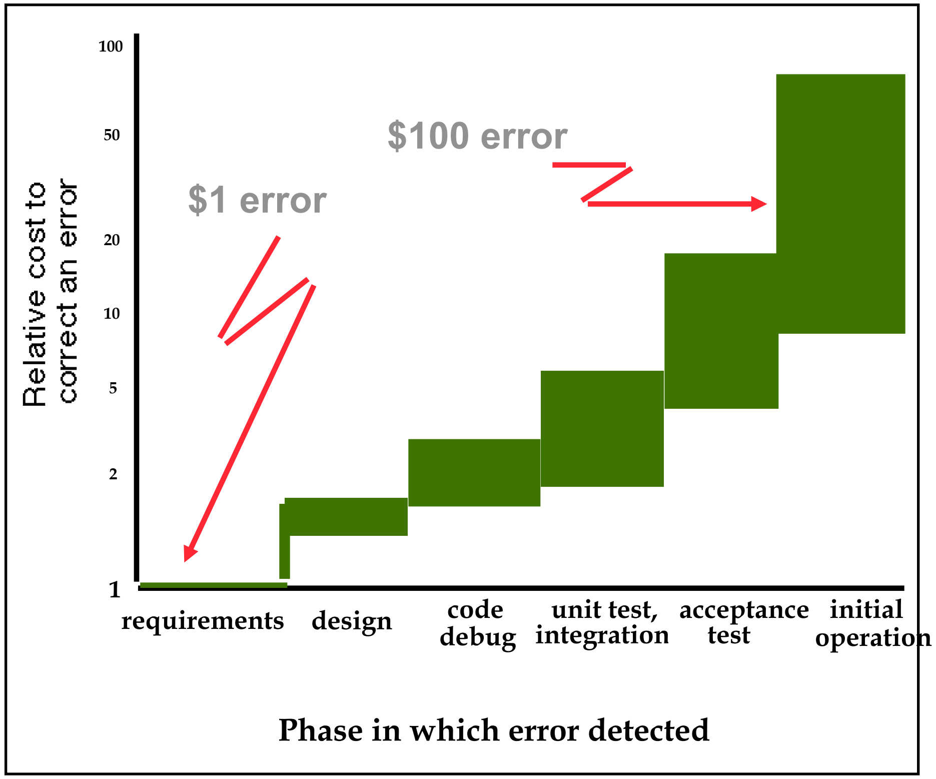
Feedback control: goal is to keep system capabilities and stakeholder goals in synch!

Costs: Importance of Early Defect Detection

1. The majority of software errors are introduced early in software development

2. The later that software errors are detected, the more costly they are to correct





Errors, Detection, and Repairs

- Basic observation:
  - Cost of a defect grows *quickly* with time between making an error and fixing it
    - Step function as defects cross scope walls: From programmer to sub-team, from close colleagues to larger team, from module to system, from developers to independent testers and from development to production
- “Early” errors are the most costly
    - Misunderstanding of requirements, architecture that does not support a needed change, ...

When

- As early as possible
  - Reduce the gap between making an error and fixing it
    - Ideally to “immediately” ... which we call “prevention” or “syntactic checking”
    - E.g., error detection/correction in Eclipse, other programming environments
- Throughout development
  - People make mistakes in every activity, so every work product should be tested as soon as possible
  - But should continue: different activities better detect different kinds of errors

Choosing What

- For every work product, we ask: *How can I find defects as early as possible?*
  - Ex: How can I find defects in software architecture before we've designed all the modules? How can I find defects in my module code before it's integrated into the system?
- Divide and conquer
  - What properties can be checked automatically?
  - What properties can be (effectively) tested dynamically?
  - How can I make reviews cost-effective?

Verification and Validation: Divide and Conquer

- Validation vs. Verification
  - Are we building what the stakeholder want? vs. Are we building according to spec?
  - Crossing from judgment to precise, checkable correctness property. Verification is at least partly automatable, validation is not.
- Correctness is a *relation* between spec and implementation
  - To make a property verifiable (testable, checkable, ...) we must capture the property in a spec

Divide and Conquer: Usability

- Real requirement:
  - Not “The product should be easy to use”
  - Rather: the product must be usable. Users with characteristics XXX should learn to use it effectively within 30 minutes, and should thereafter complete task T within S seconds with error rate E.
  - Hard and expensive (but important) to test. We probably can’t test it after every trivial change to the product.
- Divide and conquer:
  - Validate the user interface design.
  - Verify the user interface implementation: Is it consistent with the design? Does it violate any of the (precisely stated) requirements?

Who

- Cost of a defect rises dramatically at architectural and sub-team boundaries
  - It’s cheap for me to fix the bug I just created in my module. It’s much, much more expensive to find, understand, and fix a bug in a module made by a teammate who is sleeping 3000 miles away.
  - Implies local testing first

=> Test cases are part of good module interface designs

=> Module tests should be thorough and completed before a module (or revision) becomes part of the baseline used by others

The Long When

- Test execution is just one part of testing
  - And it needs to be a very cheap, automated part, because we should re-test the program over and over as it evolves
- Test design can often be done much earlier
  - Can begin building tests based on use cases and other requirements
  - Part of a good system design is devising acceptance test cases
- Test design is also a test of specifications
  - Is this specification precise, or ambiguous? Can I effectively check whether an implementation satisfies it?
  - What does it say about the SRS if I cannot write system test cases from the requirements?

How (from why, who, when, what)

- Reviews: evaluation by people
  - Situations requiring judgment
  - Where automation is expensive or impractical
- Execution testing
  - Where executable code exists
  - Black box: Test design is part of designing good specifications
  - White (or glass) box: Test design from program design
    - Allows more effective coverage of code
      - Executing every statement or branch does not guarantee good tests, but omitting a statement is a bad smell.
    - Many different approaches
- Less common
  - Formal models and proofs
  - Executable models, etc.

Testing Perspective

- Execution testing is the most common approach to establishing system quality
- What can be established by execution testing and what cannot?
  - Functional correctness?
  - Quality requirements?
  - What the stakeholders want?
- Implications

Summary: 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?

QA in Your Projects

- How do you plan to establish quality?
  - Capture QA planning in assembla pages
- Reviews (describe one)
  - What will be reviewed?
  - What kinds of reviews will be conducted and by whom?
  - What are the results
- Test plans
  - What is the testing strategy? (see CIS422W12_Team3)
  - How will tests be created and by whom?
    - Module tests, system tests, etc.
  - Which testing strategies will be used and why?
    - Black box, white box, coverage, etc.
    - Read Chapter 13 of text!