The Test Plan

Software Engineering
Computer Science 520/620
Spring 2012
Prof. Leon Osterweil

Today’s Problem

How to Build Something Like this

Focus on “How do you know”

Requirements
- Functional
- Robustness
- Performance
- Safety
- Accuracy

Testplan

Timing
Inputs
Outputs
Setup
Knockdown

Timing limit must meet performance requirement

Test input/output behavior must match functional requirements
Testing is too long and hard to do all at once at the end of development

- Divide the job into subtasks
- Do some activities during development
  - Can do test planning during development
  - And should do so
- Phase testing at the end
  - Using test plans previously developed

Testing Phases

- Unit/Module
  - Comparing a code unit or module with design specifications.
  - Planned during coding: done after coding
- Integration
  - Systematic combination of software components and modules
  - Planned during design: done after unit/module V&V
- Software System
  - Comparing entire software system with requirements
  - Planned during requirements: done after integration
- System
  - Comparing integrated hardware/software system to requirements
  - Planned during informal requirements: after SW System
Reasoning during development

- Should be incremental
- After each phase
- During each phase
- Testing not possible
  - As usually construed
  - What can be done?
**DEVELOPMENT PHASES**

- Requirements
- Specification
- Architecting
- Implementation Designing
- Coding

**TESTING PHASES**

- System Testing
- Unit Testing
- Integration Testing
- Unit Test Plan
- Software Sys Test Plan
- System Test Plan

**System Test Planning**

- Requirements Specification
- System Test Plan
- Analysis of requirements
- And System Test Planning
- happen here

**Test Plans and Test Planning**

- Goal: Determine if the product satisfies the requirements that spawned its development
- Testing is done after the product is built
- Test planning commences during requirements
  - Testing can be an elaborate process
    - Best to plan it out
  - Testing can require elaborate harnesses
  - Testing difficulties can shape product requirements

**Testing is Buying Knowledge**

- The testing costs resources
- It should result in knowledge that is worth the cost
- The value of the knowledge is up to the buyer
- The cost of the knowledge is something that software engineers should be able to estimate

All suggest the value of planning it out far in advance
Requirement vs. Test Plans

- Requirements specify "What"
- Test Plans specify "How would I know it if I were to see it?"
- Structure of one often strongly shapes the structure of the other

Test Plans

- The problem: How to devise a strategy for testing that is cost effective?
  - Meets knowledge acquisition objectives passably well
  - At a cost that is acceptable
- A Test Plan is a key part of an overall software product

Test Plans Are (Software) Objects Themselves

- They have (knowledge acquisition) goals/requirements
- They have an architectural structure
- They have specific implementations in
  - Individual testcases
  - Instructions (i.e. code) for how to perform them
- They require evaluation at the end
  - How many cases failed?
  - And how?
  - What does this tell us
  - About the software being tested
  - The testplan itself

Testplan Structure MAY Mirror Structure of Requirements Specification

- A hierarchical decomposition
  - Maybe functional
  - Maybe some other aspect
- But logically it is a separate and distinct entity
- But with many relations shared with requirements
- Or the testplan might be structured differently
  - Depends upon requirements, architecture...
Possible conceptual approach

- Test plan and requirements are separate, isomorphic DAGs (side-by-side?)
- "Test Plan" field in each requirements element node
  - Points to corresponding test plan element
  - Which is the plan for testing that requirement element
- But Test Plan DAG is separate object

And also….

“Testable Requirements”

- Being sure requirements can be verified is a very important goal for requirements development
- How to be sure a requirement is testable?
- Try to build a testplan for it
- Important to worry about the cost of running the tests, though
Easy Example: A Square Root function

- **Goal:** Determine that SQRT produces correct values quickly enough
  - **Structure of the Test Plan**
    - SQRT for positive numbers
    - SQRT for negative numbers
    - SQRT for very large numbers
    - SQRT for very small numbers
    - SQRT for inputs of various types
      - Reals
      - Integers
      - Character strings
  - **Resources**
    - Computer time
  - **Failure response:** log it and continue

Some Kinds of Test Element Goals

- Is the functionality correct?
- Does the software execute fast enough?
- Is the software easy enough to use to satisfy a particular class of stakeholders?
- Does the software *fail safe* under certain specific circumstances?

Example Structure of a Testplan Element Specification

- Goals/Requirements for this test case
- Requirements element (e.g. function) or aspect (e.g. security) being tested
- Needed resources (e.g. databases, users, computers, software)
- Setup procedure
- Input data, which may be
  - fixed, randomly selected, selected from a list
- Output results required
  - Speed required
  - Definition of what is "correct" output
    - fixed number, range, formula
  - Response to failure(s)
    - Cleanup/knockdown
  - Evaluation: turning this data into information, maybe into knowledge.

Example Structure of a Testplan Element Specification

- Goals/Requirements for this test case
- Requirements element (e.g. function) or aspect (e.g. security) being tested
- Needed resources (e.g. databases, users, computers, software)
- Setup procedure
- Input data, which may be
  - fixed, randomly selected, selected from a list
- Output results required
  - Speed required
  - Definition of what is "correct" output
    - fixed number, range, formula
  - Response to failure(s)
  - Cleanup/knockdown
  - Evaluation: turning this data into information, maybe into knowledge.

It can be hard to observe results; specify how to observe the results

- Is the functionality correct?
  - May require specification of tolerance
- Does the software execute fast enough?
  - What does "fast enough" mean
- Is the software easy enough to use to satisfy a particular class of stakeholders?
  - Do you need to bring in some stakeholders??
- Does the software *fail safe* under certain specific circumstances?
  - Have to cause the software to fail
  - What are the circumstances; how specified

Some test plan elements can potentially test more than one of these

More detail

- **Goal:** Show that SQRT produces accurate results in acceptable time, when applied to positive numbers
  - Set time bound at 2 ms.
  - Establish accuracy as up to 7 decimal places
- **Testcases**
  - For 1000 randomly generated numbers (0, 10**25)
    - Generate input number
    - Start the clock
    - Apply SQRT function
    - Capture the output
      - Numerical result
      - Elapsed time
    - Evaluate the numeric result
      - Compare it to a table
      - Square it and compare it to input
    - Generate report
    - Produce summary of generated reports
    - Evaluate summary with respect to goal
Non-Trivial Programs Are Much Harder

- Some Problems:
  - How to phase test case execution
  - Which characteristics to focus on when
  - Combining types of tests
  - Some tests require elaborate setup
  - Sometimes failures are hard to detect
  - What to do when failures are detected

Financial issues may be the most important here.
Testing is BUYING knowledge. Do what is most cost effective first

More Realistic Example:
Student Online Course Registration

- Consider all of the functions
- Timing requirements are variable
- Accuracy may be hard to determine
- Interfaces to other systems must be tested
- Robustness testing/stress testing
- Etc.

Example: “Select a Course”

- Goal: Request for a course is handled correctly, quickly
- Requirements element: (point to function in rqts. Spec.)
- Needed resources
  - Course availability database, student status database
- Setup procedure: link to needed databases
- Input data, which may be
  - Choose some available courses
  - Choose some that are not available
  - Choose some that the student needs permission for
  - Some where students is not eligible to register
- Output results required
  - Speed must be < 1 second
  - Register only eligible students in available courses....
- Response to failure
  - Ineligible student, illegal course number, course full,