Lecture 19: Functional Testing

Kenneth M. Anderson
Software Methods and Tools
CSCI 3308 - Fall Semester, 2004

Brief Review

- Program Verification
  - A program is correct if it meets its requirements specification
- Requirements Specs
  - F(input) = output
  - Functional Contract, should be as specific as possible
- Test Cases
  - Input, Documentation, and Expected Output;
  - Test Suite - a collection of test cases
- Test Run
  - Run each test case and record pass/fail
  - repeat until all tests pass

Major Problem

- How do you pick test cases?
- Two main approaches
  - Functional Testing
    - a.k.a. Black Box Testing
  - Structural Testing
    - a.k.a. White Box Testing
- Note: current testing research has moved beyond these concepts…
  - folding and sampling techniques are current
- …but they are used in this class as an introduction

Functional Testing

- In functional testing, we test the functionality of the system without regard to its implementation
  - The system is, in a sense, a black box
    - because we cannot look inside to see how it computes its output
  - We provide input and receive output

Functional Testing, continued

- Functional Testing is a strategy for helping a software engineer pick test cases
- This is useful, since selecting test cases is a tricky problem
  - A test suite should be **"complete"**…
    - with respect to the program's specification
    - but how many test cases do you need to be complete?
  - A test suite should be **precise**
    - no duplicate test cases
    - if a test suite takes too long to run, then it will get run less often (which increases the chance that a fault goes undetected)

Functional Testing, continued

- Functional testing helps create test suites by providing a criterion for selecting test cases:
  - The requirements specification of a program lists functions that the program must perform
  - A test suite is complete when it tests every function
  - For each function, determine "categories" of input that a function should treat equivalently
    - boundary conditions can be useful guides
    - test both "typical" input and error conditions
    - a test suite will need at least **one test case** for **each category** associated with **each function**

Functional Testing: Step 1

- Identify functional categories in the requirements specification that broadly classifies functions the program must perform
- Example: A database of cars (for a car dealer)
  - Persistence of Information
  - Generation of Reports
  - Sorting

Functional Testing: Step 2

- Identify specification items in the spec that correspond to functions the program must perform
- Each item should be assignable to one of your functional categories
  - Could be an iterative process, in which a specification item identifies a new functional category
- Car Database Example:
  - Generate a report listing all cars in inventory by their identification number from smallest to largest (report generation, sorting)
  - Generate a report listing all cars in inventory by the time a car has been in inventory from longest to shortest (report generation, sorting)
  - Information on car sales must be stored for at least two years (persistence of information)

Functional Testing: Step 3

- Identify functional equivalence classes for each specification item (like the GCD example in lecture 17)
- Consider the first function of the car database
  - List cars in inventory by identification number
- The functional classes might be
  - Database has zero cars
  - Database has one car
  - Database has many cars
  - Cars have only been entered into the database
  - Cars have been entered and then deleted

Step 3, continued

- The functional classes might be… (continued)
  - Cars have been entered, deleted, and then re-entered into database
  - Cars were entered in the order that they should be printed
  - Cars were entered in the opposite order that they should be printed
  - Cars were entered in a random order
  - Database has two cars with the same identification number
- Discussion
  - This is **way more functional equivalence classes than normal**, in fact, when you find a item like this it might be good to split the specification item like
    - List cars in inventory by id
    - Sort cars in inventory by id

Functional Testing: Step 4

- Determine inputs for each functional class
  - e.g. pick test cases!
  - Each class should have its boundaries tested along with some “middle” case
- Identifying test cases
  - Database has zero cars: one test case
  - Database has one car: one test case
  - Database has many cars: two test cases
    - one with two cars (a boundary condition) and one with more than two cars
    - If a maximum had been specified we would test that too

Step 4, continued

- Identifying test cases
  - Cars have only been entered into the database: one test case
  - Cars have been entered and then deleted: two test cases
    - one extra car entered and then deleted and more than one extra car entered and then deleted
  - Cars have been entered, deleted, and re-entered: three test cases
    - one was deleted and re-entered
    - more than one was deleted, and one was re-entered
    - more than one was deleted and re-entered

Functional Testing: Step 5

- Determine the number of test cases for each function
- There are two ways to do this
  - Add each of the test cases for each equivalence class together, or
  - Look for orthogonal sets of equivalence classes that can be combined and multiply their test cases
    - We will be using the “addition” method for the testing notebook
  - Orthogonal here means equivalence classes that test distinctly different things
    - in our example, we have three equivalence classes that deal with the number of cars, and three which deal with how they were entered into the database

Method 1

- Adding test cases
  - `zero cars : 1`
  - `one car : 1`
  - `more than one car : 2`
  - `entered only : 1`
  - `deleted : 2`
  - `deleted and re-entered: 3`
  - `Total : 10`

Method 2

- Multiplying Test Cases
  - (1 + 1 +2) * (1 + 2 + 3) = 4 * 6 = 24
- Think about it like this (12 of the 24 shown):

	entered only	one deleted	multiple deleted	one deleted one re-entered
zero cars				
one car				
two cars				

Functional Testing: Step 6

- Eliminate redundant test cases
  - For example zero cars in the database will probably be a functional equivalence class for several different spec. items;
    - A single test will cover that functional class for all such items
- Prioritize test cases
  - You may not have the time or budget to test them all
  - As such, give critical test cases higher priority…
  - …while test cases that test obscure or uncommon errors can be given lower priority
- You now have your test suite!