Test Case Selection

Classifier

Constraint

Body

Object-oriented Programing

Selection of Appropriate Test Data

- Fundamental questions:
  - What are the possible sources of failure in a program?
  - What test data should be selected to demonstrate that failures do not arise from these sources?
  - Alternate question: What and how many test cases are *good enough*?

Test Data Selection

- Definitions
  - Let *F* be a program, subsystem, class, method, procedure, or some other work artifact
  - Let *D* be the domain of possible inputs to that program and *d* be an instance of *D*
  - Let *F(d)* be the results of executing *F* with input *d*
  - Let *OUT* be a predicate between *D* and *F(d)*, such that *OUT(d, F(d))* is true if and only if *F(d)* is the expected result for executing *F* with input *d*
  - Let T be set of test cases, where $T \subseteq D$, and is called a test suite

Convenient Notations

- Shorthand for success/failure of a test case, when *F* is understood:

$$OK(d) = OUT(d, F(d))$$

- Shorthand for success/failure of a whole test suite, *T*:

$$SUCCESSFUL(T) = \forall t \in T(OK(t))$$

Test Data Selection

- Ideal test suite, *T*, is a set of test cases such that it succeeds if and only if there are no errors in the program

$$\forall t \in T(OK(t)) \Leftrightarrow \forall d \in D(OK(d))$$

How do we go about creating an ideal test or at least get something that is close to one?

Test-Data Selection

- Exhaustive testing: $T = D$
- Thorough testing: T satisfies some set of test-selection criteria, C
  - *COMPLETE(T, C)*
- The trick is now to choose the criteria, C, so the testing results are consistent and meaningful, and common close to being ideal
  - *RELIABLE(C)*
  - *VALID(C)*

Definitions about Criteria

- Criteria reliability:

$$RELIABLE(C) = \forall T_1 \subseteq D \forall T_2 \subseteq D$$
$$(COMPLETE(T_1, C) \wedge COMPLETE(T_2, C)) \Rightarrow$$
$$(SUCCESSFUL(T_1) \Leftrightarrow SUCCESSFUL(T_2))$$

- Criteria validity:

$$VALID(C) = \forall d \in D$$
$$\neg OK(d) \Rightarrow (\exists T \subseteq D(COMPLETE(T, C) \wedge$$
$$\neg SUCCESSFUL(T)))$$

Fundamental Theorem of Testing

Theorem:

$$\exists C \exists T \subseteq D$$
$$(RELIABLE(C) \wedge VALID(C)$$
$$COMPLETE(T, C) \wedge$$
$$SUCCESSFUL(T)) \Rightarrow \forall d \in D(OK(d))$$

- The theorem says that if there exists some criteria that is reliable and valid, a test suite that satisfies the criteria, and the test suite is successful, then all of D should be successful.
- In this case, the test suite is “thorough”

Issues Regarding Test-data Selection Criteria

- Choosing *C*
  - The criteria depends on the testing technique
  - Therefore, choose techniques first and then choose selection criteria for each technique
- Proving RELIABLE(*C*)
- Proving VALID(*C*)
- To get a handle on these three issues, we must understand the nature and causes of software errors

Sample Criteria for Modified Path Testing

- Choose T such that
  - Every line of code is executed
  - Every possible branch of a condition tried
  - Boundary conditions for loops are tried
  - If concurrent execute is possible, also
    - Every possibility on entry conditions of every critical section are tried

Sample Criteria for Input Validation Testing

- Partition D into meaning sub-domains
  - Consider representative values of D, independent of any constraints or embedded language
  - Determine if there are constraints on D
    - If there is, consider valid and invalid values
  - Determine if there is an embedded language
    - If there is, consider legal and illegal value with respect to the language
- Choose T such that
  - There is one test case for each sub-domain

Nature and Cause of Software Errors

- Terminology
  - Failures - observed errors or inconsistencies
  - Faults - errors in the software, documentation, or other work product
- What's a "bug"?
  - This is an overloaded terms that people informally use for both failures and faults

Sources of Errors

- Errors can occur in any phase or activity of the development process
- Many of these errors may manifested themselves during implementation
  - In general, the earlier you can detect a fault, the cheaper and easier it is to fix
  - For discussion purposes, we will refer to any fault that causes a failure in the software system as a *software error*, regardless of the fault's source.
- We will focus on testing techniques that uncover software errors

Sources of Software Errors

- Three sources of software errors mentioned by Goodenough and Gerhart
  - Missing Control Flow Paths
  - Inappropriate Path Selection
  - Inappropriate or Missing Action
- Others?
  - What are some of the typically mistakes that you make?
  - What are some of the hardest to find?
  - Do the types of mistakes that you make depend on the development environment?

Sources of Software Errors

- Memory management errors
  - Confusing references/pointers
  - Out-of-bounds references
  - Initialization
- Understanding semantics of other components
- Configuration and environment
- Timing errors