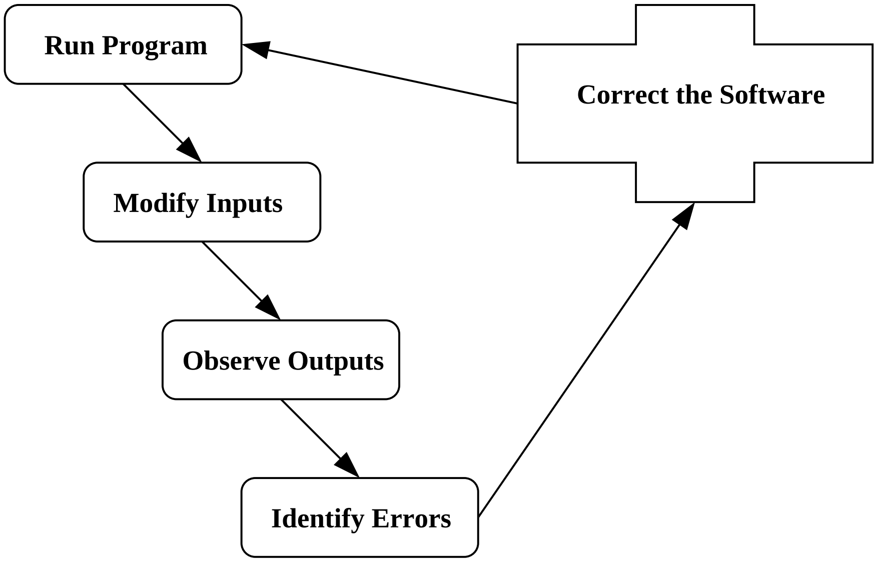
Testing

CS3 / SEOC1

Note 9

The Testing Process



Run Program: how, if implementation is incomplete?

Modify Inputs: what *range* of inputs sufficient?

Observe Outputs: of function, component, interface, system. . . ?

Identify Errors: what is an error?

Correct S/W: non-trivial. . .

When to stop?

Testing Principles

The goal of testing is not to prove that software is error-free, but rather just to find what errors we can

- All tests should be traceable to customer requirements
- Tests should be planned long before testing begins
- Testing should begin “in the small” and progress towards testing “in the large”
- Exhaustive testing is not possible
- Testing should be conducted by a third party
- The Pareto Principle applies to software testing

The Pareto Principle:

20% of causes responsible for 80% of effect

- proposed by Dr. Joseph Juran (of Total Quality Management fame), after Wilfredo Pareto – C19th economist and sociologist.
- Some examples:
  - Addressing the most troublesome 20% of the problem will solve 80% of it.
  - 20% of individuals will cause 80% of your headaches.
  - In public involvement, 20% of the people will command 80% of your time.
  - Of proposed solutions, about 20% likely to remain viable after adequate screening.
- in testing:
  - 20% of bugs cause 80% of visible errors
  - 20% of errors require 80% of time to fix
  - …

Testing Strategies

- Behavioural Testing:
  - Test that *expected* system behaviour is observed
  - Top-down versus bottom-up
  - Black-box versus white/glass-box testing
- Stress Testing:
  - Place an unnatural load on the system
  - Test performance, system limits
  - Stress until program breaks down

Behavioural Testing Strategies

- Top-down Testing (prototyping):
  - Start at subsystem level - replace modules with stubs
  - Modules can be tested as soon as they are coded
  - Top-down detects design errors early
  - A working system exists at all times
  - *Issue:* Test output is artificial
- Bottom-up Testing:
  - Modules at the lowest level of the hierarchy are tested first
  - Parent modules are replaced by drivers
  - Easier to create test cases, real input
  - Can determine performance
  - *Issue:* No demonstrable program exists until all modules have been developed

Functional versus Structural

- Functional, or “black-box”, testing:
  - Tester does not have code for routine, only a functional description of it
  - Test inputs determined by requirements
  - *Equivalence Partitioning:*
    - Determine which classes of input data have common properties
    - Test a sample from each class
- Structural, or “glass-box”, testing:
  - Tester sees source code of routine
  - Does not need to understand the program as a whole, only the module being tested
  - Hard to get clues about which test inputs best exercise the program
  - Techniques: Control Structure Testing
    - Basic Path Testing
    - Condition Testing
    - Data Flow Testing
    - Loop Testing

Categories of Testing

Unit: discovers defects in individual procedures and functions.

Integration: tests at module, sub-system and system level.

Acceptance: validates design; at early stages can include prototyping or simulation.

Which is most important?

Verifiers argue that unit testing is "*first* and *most exhaustive* test"

- Nothing else will work right unless this is done well

Validators argue acceptance testing only real test of design, thus should not be deferred until delivery.

Test Categories (1)

- Unit Test:
  - Individual components tested in isolation
    - Procedure,
    - function,
    - object
  - Stand-alone entities
  - Check that component meets spec
- (Integration Test 1) Subsystem Test:
  - Combine related modules
    - Modules identified during system design
    - Combine interdependent components (initially, tested units)
    - Test interaction of related components
    - Modules are stand-alone, entities
  - Rigorously exercise interfaces
  - Detect interface mismatches

Test Categories (2)

- (Integration Test 2) System Test:
  - Combine (potentially unrelated) subsystems
  - Find unanticipated interactions between components of subsystems
  - Validate the overall functionality of the system
- Acceptance Test:
  - Test the program with real data
    - (but not in the field)
  - Handles both verification and validation
  - Can detect errors in the requirements
  - Tests performance and functionality
  - Stages:
    - Alpha Testing
    - Beta Testing

Acceptance Testing

- Alpha Testing
  - First stage of Acceptance testing
  - System developer tests in the presence of the customer
  - Real data
  - Developer and customer reach an agreement about adequacy of the system
  - Delivered product deemed acceptable in quality and functionality
- Beta Testing
  - System is distributed to real customer site
  - Testing under actual working conditions
  - Subset of the real users
  - Training program also tested
  - Somewhat controlled environment
  - Customer agrees to report problems to developers

Regression Testing

- Corrections to errors found may introduce new errors
- Can't assume that unrelated features will not be affected after changes
- Can't just re-test modules that have been modified
- Could Test entire system after changes
  - maintain full test suite
  - costly, impractical
- Need to partition system design to limit propagation of error effects
- Develop test subsets which stand alone

Test Plans

- Testing can consume half of the overall development costs
- Test plans describe the testing process
- Components of a test plan:
  - Major phases of testing
  - Traceability to requirements
  - Schedule and resource allocation
  - Relationship between test plan and other documents
  - Test auditing

Test Cases

- Not the same thing as test data
- Test Cases:
  - input and output specifications
  - statement of the function under test
  - mapping to requirements
- Example: *Program to determine whether a triangle is isosceles*

```
function Is_Isosceles
    (Side1, Side2, Side3 : Integer)
return boolean
```

How many test cases are there?

- A triangle that is isosceles (2,2,3)
- Reorder the equivalent sides (2,3,2) (3,2,2)
- Triangle that is equilateral (2,2,2)
- Triangle that is not isosceles (1,2,3)
- Reorder numbers (2,3,1)
- Boundary conditions (1,2,0)
- Reorder boundaries (1,0,2) (0,1,2)
- Multiple boundaries (0,0,1)
- All boundaries (0,0,0)
- Large numbers (6500001, 4, 35467843)
- Floating point (1.3454, 42, 7654.245)
- Scientific notation (42e-5, 36e79, 46.3e9)
- Less than three sides (1,2) (1)
- Non-numeric characters (A, B, 42)

Testing: OO and Component Issues

- Use-cases help:
  - structure and plan testing
  - link testing to user requirements
  - plan acceptance testing
- Objects and Components:
  - **Have:** abstraction, encapsulation, interfaces, *context* (for components)
  - naturally supports top-down or bottom-up approach
  - black-box natural; glass-box supported
  - subsystem and integration testing (should be) easier
  - low-coupling makes dealing with *regression* issues easier