LIMKOKWING UNIVERSITY OF CREATIVE TECHNOLOGY

BOTSWANA

BTECH3618 • Software Testing and Reliability

CHAPTER 7

Integration / System Testing

Motivation

- During unit testing, all components work
- individually, why do you doubt that they'll
- work when we put them together?
- Interfacing

Potential Problem

- Global data structures can present problems
- Data can be lost across an interface
- One module can have an inadvertent, adverse affect on another
- Incompatibility
- Distributed features
- Individually acceptable imprecision may be magnified to unacceptable levels.

Integration Testing Hierarchy

- Big – bang Integration
  - Bottom – up Integration
  - Top – down Integration
  - Sandwich Integration

Big-bang Testing

- All components are tested in isolation, and will be mixed together when we first test the final system.
- Disadvantages:
  - Requires both stubs and drivers to test the independent components.
  - When failure occurs, it is very difficult to locate the faults. After the modification, we have to go through the testing, locating faults, modifying faults again.

Bottom-up testing

Major steps

1. Low-level components will be tested individually first.
2. A driver(a control program for testing) is written to coordinate test case input and output.
3. The driver is removed and integration moves upward in the program structure.
4. Repeat the process until all components are included in the test

Adv: Compared with stubs, drivers are much easier to develop.

Disadv: Major control and decision problems will be identified later in the
testing process.

Top-down testing

1. The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main module.

2. depending on integration approach, subordinate stubs are replaced once a time with actual components.

3. Tests are conducted as each component is integrated.

4. Stubs are removed and integration moves downward in the program structure.

Adv: Can verify major control or decision points early in the testing process.

Disadv: Stubs are required when perform the integration testing, and generally, develop stubs is very difficult.

Sandwich Integration

Uses top-down tests for upper levels of program structure, coupled with bottom-up tests for subordinate levels.

Coupling-based Integration Testing

- Cohesion – cohesion is a qualitative indication of the degree to which a module focuses on just one thing.

- Coupling – coupling is a qualitative indication of the degree to which a module is connected to other modules and to the outside world.

Coupling Types

- Independent coupling
- Call coupling
- Scalar data coupling
- Stamp data coupling
- Scalar control coupling
- Stamp control coupling
- Scalar data/control coupling
- Stamp data/control coupling

Coupling Types

- External coupling
- Non local coupling
- Global coupling
- Tramp coupling

Coupling based Testing Criteria

- Last-def-before-call(P1, call_site, *x*) – set of nodes that defines *x* and for which there is a def-clear path from the node to the call_site in P1.

- Last def-before-return(P2, y) – set of nodes htat define the returned variable y, and for which there is a def-clear path from the node to the return statement.

- Shared-data-def – a set of nodes that define a nonlocal or global variable *g* in P3 that is used in P4.

Coupling based Testing Criteria

- First-use-after-call (P1, call_site, x) – set of nodes in P1 that have uses of x and for which there exists a def-clear path with no other uses between the call statement for P1 and these nodes.
- First-use-in-callee(P2, *y*) – set of nodes for which parameter *y* in P2 has a use, and there is at least one defclear path with no other uses from the start statement to this use.
- Shared-data-use(P4,*g*) – set of nodes that use a nonlocal or global variable *g*.
- External-references(*i, j*) – *i* and *j* reference to the same external file.

Example

- • 1 main {
- 2 read(control_flag);
- 3 if (control_flag == 1)
- 4 read(x,y,z);
- 5 else {
- 6 x = 10;y = 0;z = 12;
- 7 }
- 8 ok = true;
- 9 root(x,y,z,r_1,r_2,ok);
- 10 if (ok)
- 11 write(r_1, r_2);
- 12 else
- 13 write(“No solution”);
- 14 root(float a,b,c,
- root1,root2, int
- result) {

Example

- Test case 1 = {(1, 1, 1, 1)} (Control_flag,x,y,z)
- Test case 2 = {(1, 1, 2, 1)}
- Test case 3 = {(0, 1, 1, 1)}
- {T1} satisfies call-coupling
- {T2,T3} satisfies all-coupling-defs , allcoupling-
- uses, all-coupling-paths

System Testing

- Objective: To ensure that the system does
- what the customer wants it to do.
- Objective of unit and integration testing was
- to ensure that the code implemented the
- design properly.

System Testing

- Functional Testing – Black-box testing techniques
- Category-Partitioning Testing
- Transaction Flow Testing
- Specification-based Testing

Non Functional Testing

- Stress tests – test the system in the context where the quantity or rate of input exceeds design limits.
- Performance tests – test whether the system produce results within acceptable time intervals.
- Configuration and compatibility tests – test the system in different target environment configurations, which may required to interoperate with many combinations of hardware and software.
- Security tests – evaluates the ability of the system to prevent unauthorized use, computer crime, or sabotage.
- Recovery tests – test the system automatically or manually recovery from a failure mode to normal operation.
- Usability tests – evaluates the ergonomics of an human-computer interaction design.

Acceptance Testing

- Benchmark test
- Pilot test
  – Alpha test
  – Beta test
- Parallel testing