Software Testing

COM 3220

Three meanings of bug

- error: mistake made by a developer. Mostly located in people's head.
- fault: an error may lead to one or more faults. Faults are located in text of program.
- failure: execution of faulty code may lead to one or more failures. A failure occurs when there is a difference between the results of the correct and incorrect programs.

Failure detection

- Compare actual output to expected output.
- Expected output is from specification.
- Specification: any external, independent description of the program, including user documentation.
- Are often incomplete, incorrect, ambiguous or contradictory. Specification may be wrong, not the program!

Motivation

- Derive tests from both the specification and the program.
- Derivation is done by ”predicting” likely programmer errors or likely program faults.
- Use general rules, e.g., *always test boundary conditions.*

Motivation

- Check for faults of omission: missed special cases.
- Most common type of fault according to a study by Glass.
- Experienced testers have a catalog of programming cliches and associated errors available. See Test Requirement Catalog (low-level omissions).

Motivation

- First requirement of test design: Be methodical. Three stages:
  - Finding clues
    - sources for test requirements
  - Expanding them into test requirements
    - useful sets of inputs that should be considered
  - Writing test specifications
    - exact inputs and expected outputs

Clues

- What needs testing? Collect from specification, program, bug reports, etc.
- Create a checklist.

Test requirements

- Create a test requirement catalog

Test specifications

- Describes input and exact expected output.

Supplementary code inspections

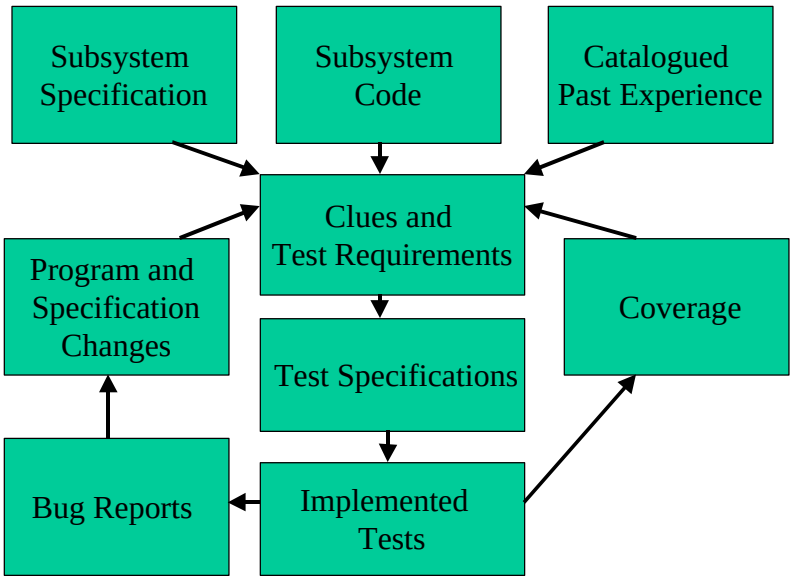
- Some faults that testing is poor at detecting.

Test implementation

- Avoid having to write a lot of support code.
- It is better to test larger subsystems because less support code needs to be written.
- Individual routines are exercised more.
- Testing the tests: test coverage as a crude measure.
- During test design do not pay attention to coverage criteria.

Test implementation

- During test design do not pay attention to coverage criteria. Test requirements from other sources should do that anyway.
- Complete subsystem testing will usually result in high coverage.
- Treat missed branches as clues about weaknesses in the test design.



A broader view: dependability

•

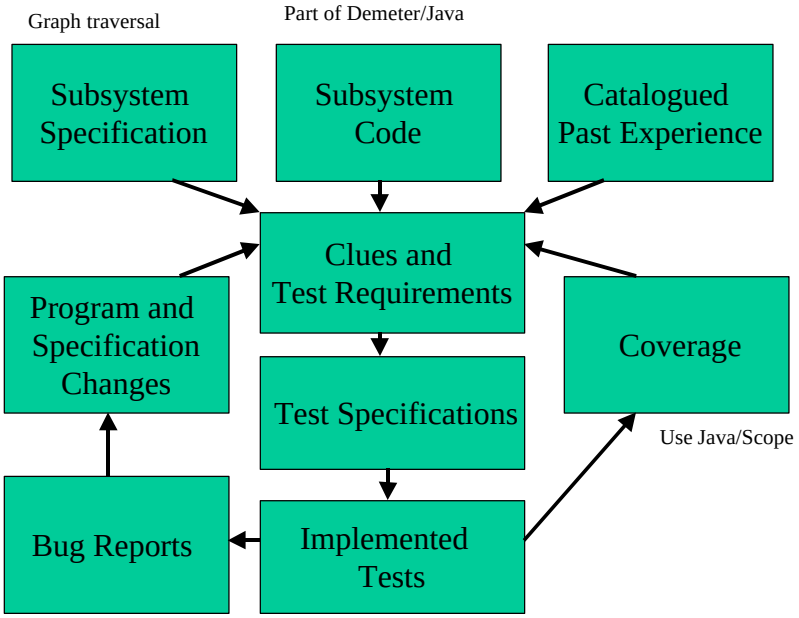
Microsoft PowerPoint Presentation

Application

- Graph algorithms:
  - Depth-first traversal
  - Finding all paths satisfying some restrictions.
- Happens to be be a subsystem of Demeter/Java.
- You don’t have to know much about Demeter. You will learn the minimal things you need.

Use Java to write testing code

- You will need to write some Java code for testing.



What we want to test

- Given a directed acyclic graph G (no multi-edges), traverse all paths from A via B to C.
- Given a directed acyclic graph G (no multiedges), traverse all paths from A bypassing B to C.

Notation for describing graphs

- A = B C D. // node A has three successors
- B = E. // node B has only one successor
- E = . // E has no successor
- This information is put into a file program.cd.
- Two files program.beh are given. Contains the traversal specification. Counts visits of C.

How to call the program

- demjava test
- The program will print the paths it traversed and print how often it visits C.

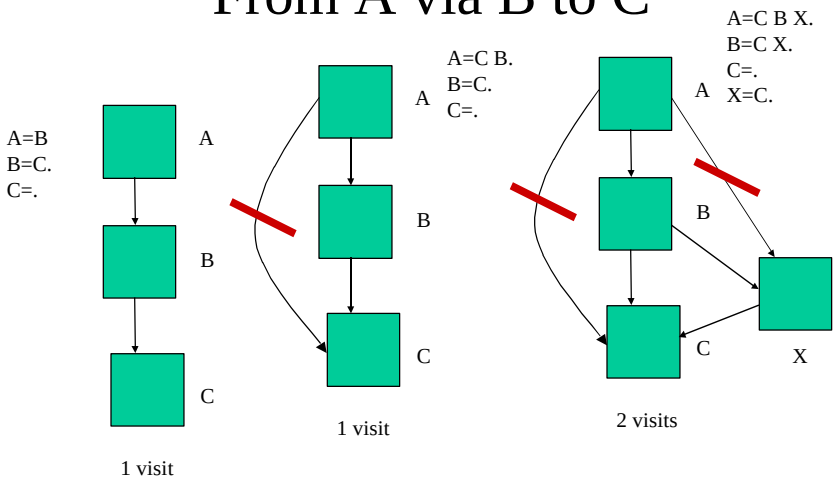
Clue list: from A via B to C

- What does program do if there is no path from A via B to C?
- What if A or B or C do not appear in the graph.
- Check that paths from A to C not going through B are excluded: paths of length 1, 2 or 3.

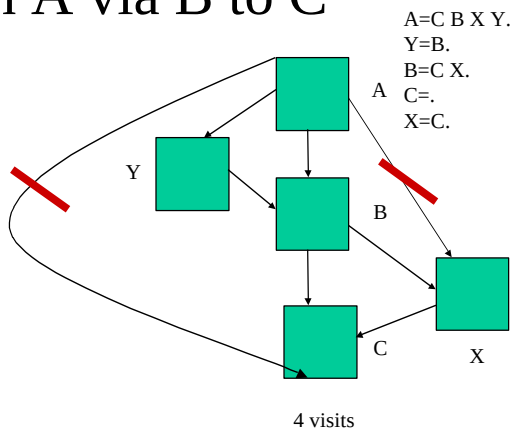
Clue list: From A bypassing B to C

- What does program do if there is no path from A bypassing B to C?
- What if A or B or C do not appear in the graph. Is it ok if B does not appear?
- Check that paths from A to C going through B are excluded: paths of length 1, 2 or 3.

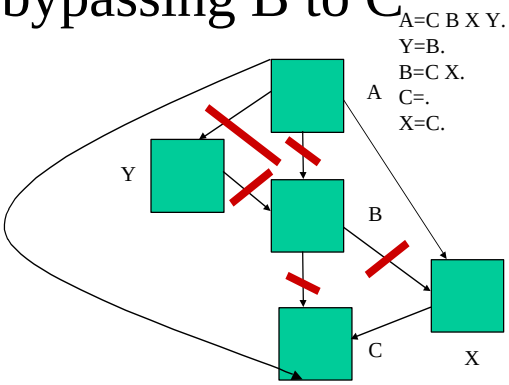
Test specifications: From A via B to C



Test specifications: From A via B to C



Test specifications: From A bypassing B to C



2 visits

Fundamental Assumptions of Subsystem Testing

- Most errors are not very creative. Methodological checklist-based approaches will have a high payoff.
- Faults of omission, those caused by a failure to anticipate special cases, are the most important and most difficult type.
- Specification faults, especially omissions, are more dangerous than code faults.

Fundamental Assumptions of Subsystem Testing

- At every stage of testing, mistakes are inevitable. Later stages should compensate for them.
- Code coverage is a good approximate measure of test quality. Must be used with extreme care.

A summary of subsystem testing

- Build the test requirement checklist
  - Find clues
  - Expand clues into test requirements
- Design the tests
  - Combine requirements into tests
  - Check tests for common testing mistakes
- Supplement testing with code inspections

A summary of subsystem testing

- Implement test support code
- Implement tests
- Evaluate and improve tests
  - use code coverage tool
  - find undertested or missing clues
  - find more test requirements
  - write more test requirements

Test strategies

- a systematic method used to select and/or generate tests to be included in a test suite.
- effective: likely to reveal bugs
- Kinds
  - behavioral = black-box = functional
  - structural = white-box = glass-box testing
  - hybrid

Testing strategies

- behavioral = black-box = functional
  - based on requirements
- structural = white-box = glass-box testing
  - based on program (coverages)
- hybrid
  - use combination

Classification of bugs

- unit/component bugs
- integration bugs
- system bugs

Generic Testing Principles

- Define the graph
- Design node-cover tests (tests that confirm that the nodes are there)
- Design edge-cover tests (that confirm all required links and no more)
- Design loop tests

Generic Testing Principles: Example

- Define the graph
  - UML class diagram
- Design node-cover tests (tests that confirm that the nodes are there)
  - Build at least one object of each class
- Design edge-cover tests (that confirm all required links)
  - use each inheritance edge and association

Generic Testing Principles: Example

- Define the graph
  - Finite state machine
- Design node-cover tests (tests that confirm that the nodes are there)
  - Use each state at least once
- Design edge-cover tests (that confirm all required links)
  - use each state transition at least once

Quality factors

- Correctness
  - conform to specification
- Maintainability
  - ease with which software can be changed
    - corrective: error fixing
    - adaptive: requirement changes MAJORITY
    - perfective: improve system
- Portability

Quality factors

- Testability
  - how easy to test? Are requirements clear?
- Usability
  - effort required to learn and operate system
- Reliability: mean-time between failures
- Efficiency: use of resources
- Integrity, Security

Quality factors

- Reusability
- Interoperability

- Write *Quality Manual* to address those issues

ISO 9000 Series of Standards (5 years old)

- How can customers judge the competence of a software developer?
- Adopted by 130 countries.
- ISO 9001: Quality Systems - Model for Quality Assurance in Design, Development, Production, Installation and Servicing. (general design)

ISO 9000 Series of Standards (5 years old)

- ISO 9000-3 Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software.
- ISO 9004-2 Quality Management and Quality System Elements

The end

Test coverage tool

- For example: For each traversal, which fraction of traversal methods are used?
- How often is each adaptive method called?
- Define global counters in Main class.
- Use aspect language to instrument code. Generate code.
- Testing tool development.

Course ideas

Advanced OO systems develops testing tools for testing class?

Test UML graphical editor.