

OO Quality

- Inspections.
- Testing.
- Metrics.
- Pitfalls and Anti-Patterns.

1

Reviews and Inspections (1 of 2)

- The sooner you detect defects, the easier it is to fix them.
 - Self inspection is best.
 - Peer reviews are next best.
 - Formal inspections are the last chance during development.
 - Worst to wait until testing to find defects.

2

Reviews and Inspections (2 of 2)

- Peer reviews – Because OO projects are better implemented with small teams, there is more emphasis on peer (1 on 1) reviews, rather than the more formal inspection process.
- The advantage of formal inspections is that they yield management metrics.
 - Inspections are project milestones.
 - Tracking defect data can lead to predictions of software quality.
 - Defect data are also process metrics.

3

Things To Look For

- Correctness – software should do the right thing.
 - Preconditions on operations should conform to the system operation models.
 - Constraints must be preserved.
 - Software should do what the requirements say.
- Completeness – software should do the complete job. No missing parts.
- Consistency – all models should be consistent with each other.

4

OO Testing (1 of 3)

- Testing should be planned in parallel with development.
 - Use cases lead to acceptance tests.
 - Class specifications lead to black box tests.
 - Dynamic models lead to white box tests.

5

OO Testing (2 of 3)

- Each class should have a static self test operation.
 - It should test out all of the operations defined in the class.
- The test method should:
 - Create one or more instances of the class.
 - Send messages to the instance(s) in the right sequence using a test script.
 - Check results.

6

OO Testing (3 of 3)

- The order in which classes should be tested depends on structure:
 - Test servers before clients. Follow the dependency arrows backwards.
 - Test superclasses before subclasses.
 - A subclass test method should invoke test methods for its ancestor classes.

7

Checking Constraints

- Ensure that the constraints are preserved. Four possibilities:
 - Check before updating state.
 - Check after updating state.
 - Delegate to another class.
 - If embedding, delegate.
 - If inheriting, put check in ancestor.
 - Delegate to client.
 - Publish constraint in class specification.

8

Pareto Effect

- 20% of the code is executed 80% of the time.
- We spend 80% of our time developing code that will be executed 20% of the time.
- The basic course scenarios account for 20% of the code.
- Error checking and handling accounts for 80% of the code.
- Bottom line: Don't plan development effort based on just the basic course scenarios.

9

Testing Strategy

- In the waterfall life cycle model, all test code is written at one time.
- In the incremental model, the test code is written along with the production code.
 - Since classes evolve, test code evolves, too.
 - Hint: write the test code at the same time as the production code.
- Caution: Extra testing time will be necessary for reusable code.
 - Hint: Test early and often.

10

White Box vs. Black Box

- White box testing – The test effectiveness is measured in terms of *coverage*.
 - Statement coverage – each statement is executed by at least one test case.
 - Branch coverage – each branch is taken each way by at least one test case.
- Black box testing – Treat the software like a black box. You can only test to the requirements specification (use cases).

11

White Box Testing

- Polymorphism can lead to additional coverage paths in white box testing.
 - An invocation of a polymorphic operation is equivalent to a multi-way branch in a traditional language.
 - If there are n possible polymorphic operation that may be invoked with a single message, then you need n test cases (one for each polymorphic receiver).

12

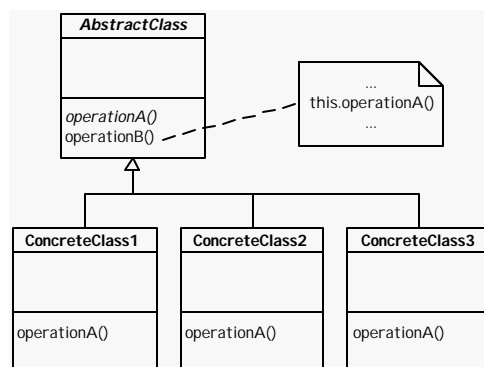
Black Box Testing

- Use the dynamic models to build and check the test cases.
 - Use cases and scenarios.
 - System operations.
 - Interaction diagrams.
 - Activity diagrams.

13

Overridden Operations

- You must test operationA() for every possible subclass definition for operationB().



14

Metrics

- Metric – the definition of some aspect whose measurement will be helpful in the management of a project.
 - Process metrics can be used to determine the effectiveness of processes.
 - Product metrics can be used to measure the quality of a product.
- Measurement – the value of a metric that applies to a particular process or product.

15

Process Metrics

- Effort (e.g., person-hours) spent during each task.
- Effectiveness (e.g., defects found per person-hour of inspections, lines of code produced per person-month).
- Cost (personnel, machines, overhead).
- Time (days, weeks, months).

16

Goal-Question-Metric (GQM) Method

- GQM is a technique for determining which metrics to take.
 - Step 1: Using a hierarchical structure, divide each goal into sub-goals.
 - Step 2: For each sub-goal, develop a set of questions whose answers will measure the effectiveness of meeting that goal.
 - Step 3: For each question, develop a set of metrics that will help answer the question.
- Reference: Basili and Rombach, *IEEE Transactions on Software Engineering*, 14(6), 1988.

17

Analysis Metrics

- Dynamic model metrics:
 - Number of use cases.
 - Number of sentences per use case.
 - Number of system operations.
- Static model metrics:
 - Number of classes.
 - Number of associations.
 - Levels of inheritance.
 - Fan-out in inheritance tree.
 - Number of attributes per class.
 - Number of operations per class.

18

Design Metrics

- Dynamic model metrics:
 - Number of messages per system operation.
 - Number of objects involved in system operation.
- Static model metrics:
 - Number of dependencies.
 - Cohesion.
 - Coupling.

19

Implementation Metrics

- Operations:
 - Number of lines of code.
 - Number of parameters.
 - Number of attributes used.
 - Number of local variables.
 - Halstead metrics.
 - McCabe cyclomatic number.
 - Number of comments.
- Between classes:
 - Fan-out.
 - Fan-in.

20

OO Pitfalls (1 of 3)

- Pitfalls are discussed in the book by Bruce Webster, *Pitfalls of Object-Oriented Development*, M&T Books, New York: 1995.
- If something can go wrong, it will (Murphy's law).
- Risk management: Figure out what can go wrong, and take steps early to either reduce the likelihood of it happening, or reduce the impact when it does happen.

21

OO Pitfalls (2 of 3)

- Kinds of pitfalls discussed in Webster's book:
 - Conceptual.
 - Political.
 - Management.
 - Analysis and Design.
 - Environment, Language and Tools.
 - Implementation.
 - Class and Object.
 - Coding.
 - Quality Assurance.
 - Reuse.

22

OO Pitfalls (3 of 3)

- Each pitfall discussion has the following structure:
 - Discussion.
 - Symptoms.
 - Consequences.
 - Detection.
 - Extraction.
 - Prevention.

23

AntiPatterns

- Brown, et. al., *Antipatterns*, Wiley, 1998, describes what they call anti-patterns.
- A pattern is something you want to repeat. An anti-pattern is something you want to avoid.
- They describe anti-patterns for refactoring software, architectures, and projects in crisis.
 - Examples: The Blob, Lava Flow, Poltergeists, Boat Anchor, Golden Hammer, Warm Bodies, Swiss Army Knife, Death by Planning, Corncob.

24