Lesson 26-
Non-Execution-Based Testing

October 9, 2015

Non-Execution-Based Testing (Schach Chap6)

- Goals of Testing:
  - Does Program Conform to Specification? Does It Meet Customer Expectations?
  - Testing Incorporated Into Every Life-Cycle Phase.
- Distinguish between:
  - *Non-Execution-Based (Static)* - Walkthroughs/Inspections
  - *Execution Based (Dynamic)* Testing - Run Test Data Through Prototype/Implementation Software (Chap 15).

*Non-Execution-Based (Static) Testing

- Underlying Principle, Developer Cannot Adequately Review own Work. Why?

- Static Testing: Don't Execute Program, Instead, Examine Code
  - Detect Faults Before Execution. Consider Errors As Isolated, ie., Error *Interactions* Not Significant.
  - Less Time Required to Find Each Error - More Cost-Effective Than Dynamic Testing.
- Two Kinds of *Static* Testing:

Walkthroughs

- A walkthrough team consists of from four to six members
- It includes representatives of
  - The team responsible for the current workflow
  - The team responsible for the next workflow
  - The SQA group
- The walkthrough is preceded by preparation
  - Lists of items
    - Items not understood
    - Items that appear to be incorrect

*Managing Walkthroughs

- The walkthrough team is chaired by the SQA representative
- In a walkthrough we detect faults, not correct them. Why not?

Managing Walkthroughs (cont.)

- A walkthrough should be document-driven, rather than participant-driven
  - Participants trace workflow artifact (specification, design, code, etc.) line by line, explain logical flow of document
- Verbalization leads to fault finding
- A walkthrough should never be used for performance appraisal!

*Inspections

- An inspection has five formal steps
  - Overview
  - Preparation
  - Inspection
  - Rework
  - Follow-up

*Inspections (cont.)

- An inspection team has four members (Schach) *(IEEE standard recommends 3 – 6)*

*Recording Faults

- Checklist should be used for uncovering potential faults
- Faults are recorded by severity
  - Example:
    - Major or minor
- Faults are recorded by fault type
  - Examples of design faults:

Getting it Right- How Do We Know?

- **Verification:**
  - At the end of each phase
  - Determine if phase **properly carried out**.

- **Validation**:
  - Before delivering final product to client
  - Is end product **what the client specified**?

Requirements Verification

- Q: How can we verify that requirements have been met?

Requirements Validation

- Q: How can we validate that requirements have been met?

Analysis Verification

- Every item in the analysis artifacts must be traceable to an item in the requirements artifacts
  - Similarly for the design and implementation artifacts
- Automated tools (such as Telelogic DOORS) used to index and cross-reference requirements document and specification
  - Can use Excel for providing traceability on course project

Design Verification

- How to verify Design Workflow?

Implementation Verification

- Q: How to verify component implementation?

Integration Verification

- How to verify System Integration?

- Traceability of artifacts is important requirement for successful testing

*Fault Metrics

- What kinds of Metrics are meaningful for Code Inspections?

- What could a 50% increase in the fault detection efficiency mean?

Use of Fault Metrics

- For a given workflow, we compare current fault rates with those of previous products
- We take action if there are a disproportionate number of faults in an artifact
- We carry forward fault statistics to the next workflow

*Statistics for Non-Execution-Based Testing->Inspection

- 93% of all detected faults found during inspection (IBM, 1986)
- Cost of detecting fault decreased by 90% (Switching sys 1986)
- 4 major faults, 14 minor faults per 2 hours (JPL, 1990). Savings of $25,000 *per inspection*
- # faults detected decreased exponentially by phase (JPL, 1992)
- Any special considerations for use of fault statistics?

~*Focus on Value of Code Inspections

- Average effort to detect defects in hours per defect.

Distribution of Average Effort To Detect Defects [Briand, et. al.,1998]

Defect Detection Technique	Minimum Value	Most Likely Value	Maximum Value
Design Inspections	0.58	1.58	2.9
Code Inspections	0.67	1.46	2.7
Testing	4.5	6	17

- Increasing implementation of inspections has moved rework to earlier in the lifecycle

Hewlett Packard Experience [Grady and Van Slack 1994]

Work Product	Estimated Starting Point (Extent of Adoption)	Estimated 1993 (Extent of Adoption)	Percentage Total Cost Saved	Rework Percentage	Estimated $ Savings Per Year
Specification	1%	29.5%	28.5%	17%	$10,175,000
Design	1%	34.8%	33.8%	11%	$7,808,000
Code	5%	42.3%	37.3%	4%	$3,133,000
Test Plan	1%	17.1%	16.1%	1%	$338,000
			TOTAL	33%	$21,454,000

Summary

- Goals of Testing
- Non-Execution vs. Execution-Based Testing
- Two Types of Static Testing-
  - Code Walkthroughs
  - Inspections
- Verification vs. Validation
- Fault Metrics
- Non-Execution-Based Testing Value