Inspection

219343: Software Testing

Lesson 09-1 v1.0

Jittat Fakcharoenphol

Fall 2007/ 2008

Inspection

- According to Parnas and Lawford:
  - Systematic approach to examining a program in detail
  - To assess the quality of the software product in question, *not* the quality of the process used to develop the product

Handling complexities

- Use divide and conquer
- Examine small parts while making sure that
  - nothing is overlooked
  - the correctness of all inspected components implies the correctness of the whole product

Goal

- Inspection lies in between testing and formal verification
- Inspection seeks to compliment testing
  - Make sure program corrects
  - Also, checking coding style, naming conventions, maintainability

Three reading techniques

- From [Dunsmore et al.]
  - Checklist
  - Use-case driven
  - Abstraction-driven

Checklist

- Been around since 1970s
- Problems [Laitenberger & DeBaud]
  - questions too general
  - instruction missing
  - detection biased towards previous defects
- See handout

Use case

- Check that each object is capable of responding correctly to all situations
- Force the inspector to consider the context in which an object is used

Use case: steps

- Creating scenarios
  - Take each use case, derive a series of brief scenarios
- Using the scenarios
  - trace on the sequence diagram
  - when encountering the class under inspection, switch to code
  - check decisions and state changes when inspecting the method
  - verify the final state after the scenario is finished

Abstraction-driven

- "Localizing the delocalization."
- Create natural language abstraction for each part (method) of the program (while you go)
  - should make later inspection easier, i.e., no further analysis of these is needed – the delocalization has been localized in the abstraction

Abstraction-driven: steps

- Analyze the interdependencies between classes
- Analyze the classes – starting with classes with the least dependencies
- Dependencies between methods within classes are analyzed
- Inspect the methods with least dependencies first
- Reverse engineer an abstract specification for each method
  - understand external references
  - state changes
  - outputs