SED
Static Testing Methods

Static Testing Methods

- Code Walkthroughs
- Code Inspections
- Semantic Analysis of Code
- Fagan Inspections
- Code Metrics
- Static Testing Tools

Unit Test Methods
Static & Dynamic Testing

- Tests which do not involve executing the program, are called **static test methods**
- Methods which do involve executing the code are called **dynamic test methods**

- *Note* Static methods may still use a computer. For example, compilation is a static test method.

Code Walkthroughs

- A **Code Walkthrough** is an *informal* meeting where the programmer leads the review team through his/her code and the reviewers try to identify faults
- You can do this with your own code but it is more effective if the reviewer is not the same person as the programmer. See *egoless programming* in many SE texts
- Example: perform a code walkthrough of the `getNumofDaysInMonth` method

```java
if (year<1) {
    // EXAMPLE
    throw new YearOutOfBounds(year);
} else if (month==1 || month==3 || month==5 || month==7 || month==10 || month==12) {
    numDays = 31;
} else if (month==4 || month==6 || month==9 || month==11) {
    numDays = 30;
} else if (month==2) {
    if (isLeapYear(year)) {
        numDays = 29;
    } else {
        numDays = 28;
    }
} else {
    throw new MonthOutOfBounds(month);
}
return numDays;
```

See B&D, 7th Ed. Page 457

Code Inspections

- Inspections are a formal, efficient and economical method of finding faults in design and code [Fagan,76]
- Code inspection amounts to "executing the code in your head" or on paper
- Code inspections are very effective at finding faults [statistics from Pfleuger p.291]
  - detect 67% of faults [Fagan 76]
  - detect 85% of faults [Jones 77]
  - detect 93% of errors [Ackerman et al 86]
Code Inspections (cont.)

- A component is selected for inspection by a review team
- The formal meeting is chaired by a moderator
- The code developers are not allowed to present any artefacts – this is done by the review team
- Things to be inspected may include
  - Efficiency of algorithm w.r.t. non-functional requirements
  - Correctness of code
  - Comments and consistency with code
- Code developers do not need to be present

Code Inspections (cont.)

- (In general) Four people constitute a good-sized code inspection team:
  1. Moderator
  2. Designer
  3. Coder/Implementor
  4. Tester

Checklists (used in code inspections)

- A set of questions to stimulate critical appraisal of all aspects of the system
- Questions are usually general in nature and thus applicable to many types of system
- Examples in Fagan (see SED Readings)
- Java Code Inspection Checklist (see SE Readings), e.g.
  - Variable and constant declaration defects (VC)
  - Computation/numeric defects (CN)

Unit Testing by Code Inspection

- Test Purpose:
  - to detect logical faults in implemented units
- Evidence:
  - program code and its specification
- Assumptions:
  - correct operating environment for calling the unit
- Deduction:
  - Testers reason about the behaviour of the code to determine whether the code produces the correct results.
- Verdict:
  - Code ready (for other tests) OR code revision required

Semantic Analysis

- Analysis based on a model of the meaning (ie semantics) of a program
- Formal Proofs: prove a given program (model) satisfies a required property
- Control Flow Analysis: analysis of the directed graph of the control structure of a program to identify inaccessible code, infinite loops and poor structure
- Data Flow Analysis: analysis of a diagrammatic representation of the flow of data throughout a program
- Symbolic Execution: check for agreement between code and specification using algebraic variables in place of input data. Assignment statements produce algebraic output which can be compared with expected results.
Code Metrics

- Measures of properties of code which may predict how likely the code is to contain errors
- For example,
  - graph theoretic complexity (of the program’s control graph)
  - module accessibility (how many ways a module may be accessed)
  - number of entry and exit points per module
  - for some Object Oriented Metrics see http://yunus.hun.edu.tr/~sencer/oom.html

Static Testing Tools

- Lint – the original static code checker for C. Checks for suspicious constructs, unreachable code etc.
- Checkstyle – checks the coding style of java source.
- PMD – analyzes java code, identifying inefficiencies.
- BLAST (Berkley Lazy Abstraction Software verification Tool) – a software model checker for C that examines all possible executions of the programme for faults.