同济大学软件学院
School of Software Engineering, Tongji University

Software Testing Foundation Level

Chapter 3: Static techniques

刘琴 Qin Liu

3 Static techniques

3.1 Static techniques and the test process

3.2 Review process

3.3 Static analysis by tools

3.1 Static Test

- Test object is not provided with test data, not executed
  - Analyzed, by persons and tools
  - Used for all documents relevant to software development and maintenance
    - Tool-supported static analysis is only for documents with a formal structure
- Goal: find defects and deviations, optimizing the development process

3.1.1 Structured Group Evaluations

- Reviews, applying human analytical capabilities
- The only possibility to check the semantics of a document
- Relying on colleagues of the author to provide mutual feedback: peer reviews
- A means for quality assurance
  - Eliminating defects and inconsistencies

3.1.2 Positive Effects

- Cheaper defect elimination
- Shortened development time
- Decreased costs and time for dynamic tests
- Cost reduction during the whole product life
- Reduced failure rate during operation
- Mutual learning
- Helping the author find forgotten issues
- Whole team feeling responsible for the quality

3.1.3 Some Issues

- Potential problem: in a badly moderated review session
  - Author feeling that he (not the document) is subject to critical scrutiny
- Cost of reviews: 10-15% of development budget
- Savings of reviews: 14-25%
- If reviews are systematically used and efficiently run, more than 70% of defects can be found and repaired
- Success factors
  - Every review has a clear goal, formulated beforehand
  - Right people are chosen as review participants

3 Static techniques

3.1 Static techniques and the test process

3.2 Review process

3.3 Static analysis by tools

3.2.1 Types of Reviews

- Informal: no real process (hallway chats, buddy tests, pair programming), yet useful, cheap, popular
- Technical: documented and defined defect removal process, involving technical experts but not managers
- Walkthrough: author “walks through” review item
- Inspection: a trained moderator (other than the author) leads the inspection team (with defined roles) through a formal inspection process (rules, checklists, entry and exit criteria), which includes gathering defect removal metrics
- When walkthroughs, technical reviews or inspections are performed by a peer group, the review may be called a peer review

3.2.1.1 Walkthrough

- Manual, informal
- Less focus on preparation
- Typical usage situations (scenarios) are discussed
- Test cases may be played through
- Reviewers try to find possible errors/defects by spontaneously asking questions
- Useful for small teams (up to 5 persons), for checking noncritical documents
- Objective: mutual learning, development of understanding of review object, error detection

3.2.1.2 Inspection

- The most formal review
- Reviews use checklists with criteria for checking different aspects
- Goal: finding unclear items and possible defects, measuring quality, improving quality
- Called design inspection or code/software inspection
- Inspection meeting
- Pair Programming, two programmers work together at one workstation ?
- In addition: optimizing the development process

3.2.1.3 Technical Review

- Focus: compliance of the document with the specification, fitness for its intended purpose, and compliance to standards
- Technical experts as reviewers (some should not be project participants for avoiding project blindness)
- High preparation effort

3.2.1.4 Informal Review

- A light version of review
- Initiated by the author in most cases
- No meeting: just a simple author-reader-cycle
  - Cross-reading by colleagues
- Results need not be explicitly documented
  - A list of remarks or the revised document is enough
- Types of informal review
  - Pair programming, buddy testing, code swapping
- Very common and highly accepted due to minimal effort required

3.2.1.5 Pair Programming

- An agile software development technique
- Two programmers work together at one workstation
  - Driver: writing the source code
  - Observer/navigator: reviewing each line of code
  - The two programmers switch roles frequently
- Advantages
  - Economics
  - Design quality
  - Satisfaction
  - Learning
  - Team-building and communication
- Pairing variations
  - Expert-expert, expert-novice, novice-novice

3.2.1.5 Pair Programming (cont.)

- Remote pair programming (virtual pair programming, distributed pair programming)
  - Two programmers are in different locations
  - Working via a collaborative real-time editor, shared desktop, or a remote pair programming IDE plugin

3.2.2 Consensus and Understanding

- Incompleteness and ambiguity can hide the real meaning of the specifications
- Agreement and uniform understanding of the specifications

Long before any code exists, the specification must be handed to an outside testing group to be scrutinized for completeness and clarity. As [V.A.] Vyssotsky [of Bell Lab's Safeguard Project] says, the developers themselves cannot do this: "They won't tell you they don't understand it; they will happily invent their way through the gaps and obscurities."

– Fred Brooks
The Mythical Man-Month
1975

3.2.3 A Generic Review Process

1. Planning
2. Kick-off
3. Preparation
4. Review meeting
5. Rework/repair
6. Follow-up

Includes estimating and planning, training participants, etc.

These steps of the process repeat per each item reviewed. Preparation is usually one to two hours alone. Meeting is one to two hours together. Rework/repair is fixing the bugs found.

Follow-up includes on individual items as well as overall process improvement analysis, evaluation of defect (bug) removal at phase exit reviews (exit meetings), etc.

The details of the review process depend on the specific review type used on the project

3.2.4 Roles and Responsibilities

- Moderator: Lead the review meetings
- Scribe or secretary: Gather information on findings
- Author: Describe, explain, answer questions on item
- Reviewer/inspector: Find defects (bugs) in item
- Manager: Plan, arrange resources and training, support, analyze process metrics
- In some cases, one person may play multiple roles
  - Authors sometimes act as moderators
  - One of the reviewers can act as the secretary
  - The specifics are determined by the type of review

3.2.5 Suggestions for Successful Reviews

- Provide training
- Review the product, not the producer
- Set and follow agenda and objectives
- Limit debate
- Focusing on finding, not fixing, problems
- Take written notes
- Limit and carefully select participants
- Insist on preparation (e.g., by having people submit notes)
- Develop a checklist for each type of item that is reviewed
- Review the reviews
- Use the right techniques
- Ensure management support
- Learn and get better!

3.2.6 Common Requirements and Design Bugs

- Ambiguities: What exactly does that mean?
  - E.g.: System shall allow user to read ISP e-mail
  - What ISPs? What size e-mails? Attachments?
- Incompleteness: Okay, and then what?
  - E.g.: Upon three invalid passwords, system shall lock user's account…
  - For how long? How to unlock? Who can unlock?
- Untestability: How can I check this item?
  - E.g.: System shall provide 100% availability
  - No known test technique to demonstrate perfect availability
- Excessive dependencies, coupling and complexity
  - Look for ugly design diagrams and confusing requirements

3.2.7 IEEE 1028 Software Review Standard

1. Overview
   - Purpose, scope, conformance, organization, application
2. References
3. Definitions
4. Management reviews
   - Responsibilities, inputs/outputs, entry/exit criteria, procedures
5. Technical reviews
   - Responsibilities, inputs/outputs, entry/exit criteria, procedures

3.2.7 IEEE 1028 Software Review Standard

6. Inspections
   - Responsibilities, inputs/outputs, entry/exit criteria, procedures, data collection, process improvement
7. Walkthroughs
   - Responsibilities, inputs/outputs, entry/exit criteria, procedures, data collection, process improvement
8. Audits
   - Responsibilities, inputs/outputs, entry/exit criteria, procedures

3.2.8 Selection Criteria

- Depending on how thorough the review needs to be, and effort that can be spent
  - The form of review results (documentation, or presented informally)
  - Date and time for the review
  - Technical knowledge from different disciplines or not
  - Level of technical knowledge required
  - Presentation effort appropriate or not
  - Is review object formally written
  - How much management support

3.2.9 Success Factors

- Reviews help improve the examined documents
- Human and psychological factors have a strong influence
- Testers should be used as reviewers
- Considering type and level of examined document, and state of knowledge of participants
- Checklists and guidelines should be used
- Training
- Management can support
- Continuous learning

3 Static techniques

3.1 Static techniques and the test process

3.2 Review process

3.3 Static analysis by tools

3.3 Static Analysis

- Objective similar to reviews, but tools do the analysis
  - Example: spell check is a form of static analyzer
- Document to be analyzed must follow a formal structure, in order to be checked by a tool
  - Technical requirements
  - Software architecture
  - Software design (e.g. UML diagrams)
  - HTML/XML documents
  - Formal models
  - Program code

3.3 Static Analysis (cont.)

- Analysis tools often produce a long list of warnings and comments
  - Information must be handled intelligently
- Static analysis and reviews are closely related
  - Static analysis should be performed before review, if documents are formal enough
- Not all defects can be found using static testing
- Some inconsistencies and defect-prone areas are difficult to find by dynamic testing
  - Detecting violation of programming standards or use of forbidden error-prone program constructs is only possible with static analysis (or reviews)

3.3 Static Analysis (cont.)

- Defects and dangerous constructions that can be detected by static analysis
  - Syntax violations
  - Deviations from conventions and standards
  - Control flow anomalies
  - Data flow anomalies
- In addition, static analysis can detect security problems
  - E.g., lack of buffer flow protection, failing to check whether input data out of bounds

3.3.1 Compiler as a Static Analysis Tool

- Violation of syntax can be detected by the compiler, and reported as fault or warning
- Additional information/checks
  - Generating a cross-reference list
  - Checking for correct data type usage
  - Detecting undeclared variables
  - Detecting dead code
  - Detecting overflow/underflow of field boundaries
  - Checking interface consistency
  - Detecting the use of all labels

3.3.2 Conventions and Standards

- Compliance to conventions and standards can be checked with tools
- Only guidelines that can be verified by tools should be accepted in a project
- An additional advantage
  - If the programmers know that the program code is checked for compliance to the programming guidelines, their willingness to work according to the guidelines is much higher

3.3.3 Data Flow Analysis

- Checking the usage of data on paths through the program code
  - An anomaly is an inconsistency that can lead to failure (but does not necessarily do so)
  - May be flagged as a risk
- The analysis checks usage of every variable
  - **Defined (d)**: the variable is assigned a value
  - **Referenced (r)**: the value of the variable is read and/or used
  - **Undefined (u)**: the variable has no defined value

3.3.4 Data Flow Analysis (cont.)

- Data flow anomalies
  - **ur-anomaly**: an undefined value (u) of a variable is read on a program path (r)
  - **du-anomaly**: the variable is assigned a value (d) that becomes invalid/undefined (u) without having been used in the meantime
  - **dd-anomaly**: the variable receives a value for the second time (d) and the first value had not been used (d)

3.3.5 Data Flow Analysis (cont.)

- Example of data flow anomalies

```
void exchange (int& Min, int& Max) {
   int Help;
       if (Min > Max) {
       Max = Help;
       Max = Min;
       Help = Min;
       }
   }
```

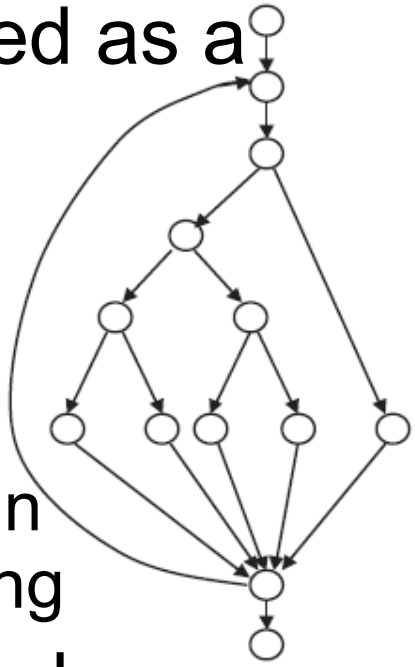
- ur-anomaly of the variable *Help*
- dd-anomaly of the variable *Max*
- du-anomaly of the variable *Help*

3.3.5 Data Flow Analysis (cont.)

- Not every anomaly leads directly to an incorrect behavior (e.g. du-anomaly)
- An exact examination of the program parts is worthwhile, and further inconsistencies can be discovered

3.3.6 Control Flow Analysis

- Program structure can be represented as a control flow graph
- Possible anomalies can be detected
  - Jumps out of a loop body
  - A structure with several exits
  - Not necessarily lead to failure, but not in accordance with structured programming
- If parts of graph are very complex and relations are not understandable, the program should be revised

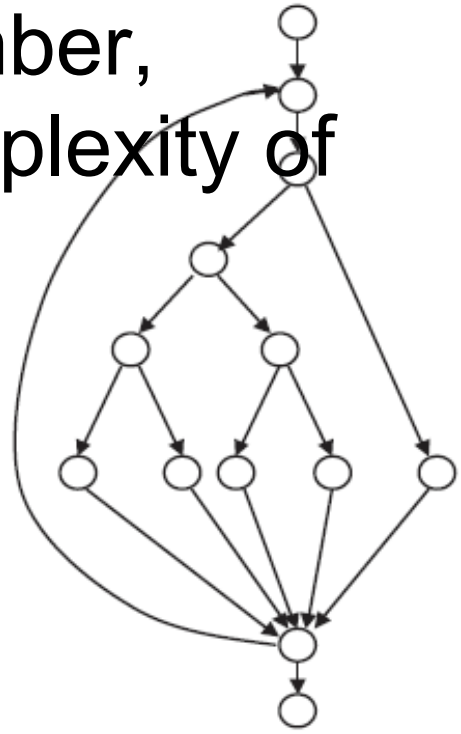


3.3.6 Control Flow Analysis (cont.)

- Predecessor-successor table
  - Showing how every statement is related to others
- If a statement has no predecessor
  - Unreachable (dead code)
- The only exceptions: first/last statements

3.3.7 Determining Metrics

- Static analysis tools also provide measurement values (metrics), for measuring quality characteristics
- Example: the *cyclomatic* number, measuring the structural complexity of code
  - Definition: v(G) = e - n + 2
  - Testability and maintainability



3 Static techniques

3.1 Static techniques and the test process

3.2 Review process （A2.1.5 A live inspection)

3.3 Static analysis by tools (A1 Assignment 1)

上海市嘉定区曹安公路4800号，同济大学嘉定校区软件学院