STATIC TESTING

Chapter 3

Topics to be covered

I. **Static Testing:**
   1. Structured Group Examinations- Reviews

II. **Static Analysis:**
   1. Data Flow Analysis
   2. Control Flow Analysis
   3. Tools For Static Testing

Objectives

- Static examinations, like **reviews**, **tool supported document** and **code analyses**, can be successfully used for quality improvement.
- ***The test object*** is not executed with test data, but is analyzed instead.
- Able to statically analyze program code using compilers, data flow analysis, and control flow analysis.

Objectives …

- **The main goal of examination** is to find defects and deviations from the existing specifications, defined standards, or even the project plan.
- The **results** of these examinations are additionally used to optimize the development process.
- **The basic idea is defect prevention at earlier stage**

3.1 Structured Group Examination

- Systematic use of human capability to think and analyze
  - Apply human analytical capability to check and evaluate complex issues
  - This done by through intensive reading and trying to understand the documents that are examined
  - Techniques : Reviews
  - Another terms : inspection
  - Peer reviews : if colleagues provide feedback
  - Types of reviews
    - **Walkthrough**
    - **Inspection**
    - **Technical review**
    - **Informal review**

3.1.1 Reviews

- Review is a common generic term for all the human **static analysis** techniques, as well as the term for a specific **document examination** technique.
- Reviews rely on the colleagues of the author to provide feedback. they are also called **peer reviews.**

1.Means to assure quality

  - Reviews are an efficient means to assure the quality of the examined documents.
  - Eliminating defects leads to improved quality of the documents and has a positive influence on the whole development process.

Reviews…

2. Potential problem for reviews (what problems?)

3. Reviews cost and savings
   - ➢ The cost for reviews are estimated 10 -15% of the development cost.
   - ➢ Savings are estimated to be about 14-25% .

□ 70% defects in documents can be found in systematic reviews.

Positive Effects Of Reviews

1. It results in cheaper defect elimination.
2. It results in shortened development time.
3. If defects are recognized and corrected early, costs and time needed for execution of dynamic tests decrease.
4. cost reduction can be expected during the whole lifecycle of a product.
5. A reduced failure rate during operation of the system can be expected.
6. As the examinations is done using a team of people, reviews lead to mutual learning.
7. As several persons are involved in a review, a clear and understandable description of the facts is required.
8. The whole team feels responsible for the quality of the examined object and the group will gain a common understanding of it.

3.1.3. The General Process

A review requires six work steps:

1. Planning :reviews must certainly be planned &put the view points.
2. Overview: if an overview meeting is considered necessary, time and place must be chosen.
3. Preparation: intensively study of the review object
4. Review meeting: the review meeting is led by a review leader or moderator.
5. Rework :the manager decides whether to follow the recommendation or to select a different approach
6. Follow-up: the correction of the defects must be followed up, usually by the manager, moderator

General Rules For Review Meeting

1. It should be limited to 2hrs.
2. The moderator has a right to cancel or continue.
3. The test object must be the Document ,not the author.
4. The moderator shouldn't be a reviewer.
5. General style Q’s shall not be discussed.
6. Developing solution is not the task.
7. Issues must not written as commands to the author.
8. Issues must be weighted.
9. All the participants should sign on the protocol.

3.1.4. Roles and Responsibilities

1. **Manager:** The development manager selects the objects to be reviewed
2. **Moderator:** The moderator is responsible for administrative task & the moderator is crucial for the success of the review.
3. **Author:** The author is the creator of the document
4. **Reviewer:** They shall identify and describe problems in the review object. They shall represent different viewpoints.
5. **Recorder:** The recorder (or scribe) shall document the findings

Possible Difficulties

- Reasons for reviews to fail

1. The required persons are not available or do not have the required qualification or technical aptitude.
2. Inaccurate estimates during resource planning
3. lack of preparation.
4. missing or insufficient documentation.
5. lack of management support

Types Of Reviews

1. **Walkthrough**
2. **Inspection**
3. **Technical review**
4. **Informal review**

Walkthroughs

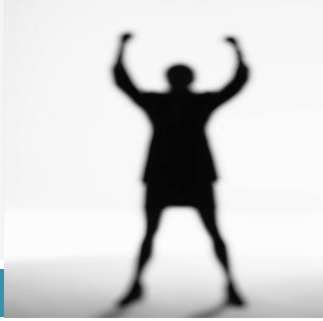
- Informal examination of a product (document)
- Made up of:
  - Developers
  - Client
  - Next phase developers
  - Software quality assurance group leader
  - Suitable for small development teams.

- Produces:
  - List of items not understood
  - List of items thought to be incorrect

Software Inspections

- Involve people examining the source representation
- with the aim of discovering anomalies and defects
- Do not require execution of a system so may be used before implementation
- May be applied to any representation of the system (requirements, design, test data, etc.)
- Very effective technique for discovering errors

Inspection Success



- Many different defects may be discovered in a single inspection.
  - In testing, one defect may mask another so several executions are required.

- The reuse domain and programming knowledge so reviewers are likely to have seen the types of error that commonly arise

Inspections and Testing

1. Inspections and testing are complementary and not opposing verification techniques.

2. Both should be used during the V & V process.

3. Inspections can check conformance with a specification but not conformance with the customer's real requirements.

4. Inspections cannot check non-functional characteristics such as performance, usability, etc.

Program Inspections

1. Formalised approach to document reviews
2. Intended explicitly for defect DETECTION (not correction)
3. Defects may be logical errors, anomalies in the code that might indicate an erroneous condition (e.g. an un-initialised variable) or non-compliance with standards

Inspection Procedure

- System overview presented to inspection team.
- Code and associated documents are distributed to inspection team in advance.
- Inspection takes place and discovered errors are noted.
- Modifications are made to repair discovered errors.
- Re-inspection may or may not be required.

Inspection Teams

- Made up of at least 4 members
  - **Author** of the code being inspected
  - **Inspector** who finds errors, omissions and inconsistencies
  - **Reader** who reads the code to the team
  - **Moderator** who chairs the meeting and notes discovered errors
  - Other roles are Scribe and Chief moderator

Inspection Checklists

- Checklist of common errors should be used to drive the inspection
- Error checklist is programming language dependent
- The 'weaker' the type checking, the larger the checklist
  - Examples: Initialization, Constant naming, loop termination, array bounds, etc.

Technical Review

- □ Focus is on compliance of document with specification, fitness for it’s intended purpose, compliance to standards.
- □ Technical experts as reviewers.
- □ High preparation efforts are required.

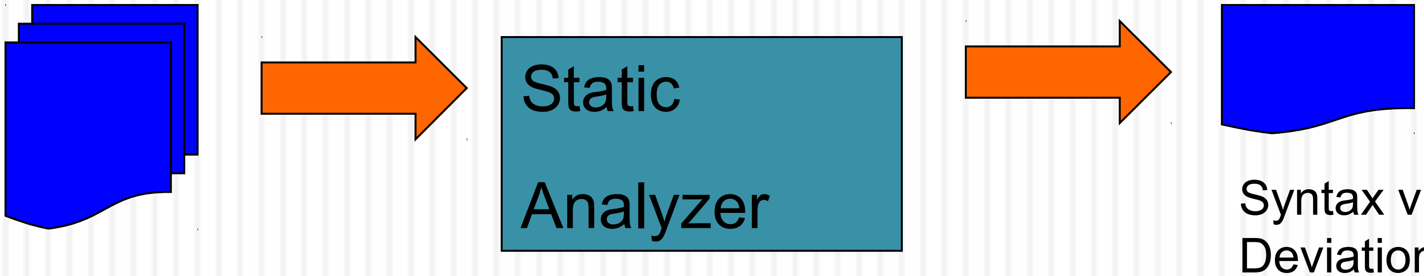
Informal Reviews

- Light version of review.
- High acceptance

3.2.Static Analysis

- What is static analysis?
  - Analysis of software artifacts e.g., requirements or code, carried out without execution of these software artifacts
- Objective of static analysis
  - To reveal defects or parts that are defect-prone in a document
  - Additional objective
    - To derive metrics in order to measure and prove the quality of the object
- How is static analysis done?
  - Static analysis tools known as static analyzers
- Objects to be analyzed
  - Formal documents that must follow a certain formal structure

Static Analyzers: General Form



Static

Analyzer

Document
Eg. Source code

Syntax violation
Deviation
Data flow info
Control flow info

Static Analysis

- Who and when used static analysis tools?
  - **Developers**
  - Before and during component or integration testing
    - To check if guidelines or programming conventions are adhered to
  - During integration testing : analysis adherence to interface guidelines
- What are produced by static analysis tools?
  - List of warnings and comments
    - Syntax violation
    - Deviation from conventions and standards
    - Control flow anomalies
    - Data flow anomalies
    - Metrics

Static Analysis (cont'd…)

- If a static analysis is performed before a review…
  - number of defects can be found.
  - number of the aspects to be checked in the review decreases
  - Thus much less effort in a review
- Not all defects can be found using static testing
  - Some defects become apparent only when the program is executed (runtime)
    - Example: division by zero valued variable

3.2.1.Compiler as Static Analysis Tool

- Detection of violation of the programming language syntax.
- Further information and other checks
  - Generating a cross reference list of the different program elements (eg: variables, functions)
  - Checking for **correct data type** usage by data and variables in programming languages with strict typing
  - Detecting **undeclared variables**
  - Detecting code that is **not reachable**
  - Detecting **overflow or underflow** of field boundaries
  - Checking of **interface consistency**
  - Detecting the **use of all labels** as jump start or jump target

3.2.2.Data Flow Analysis

- What is it?
  - A form of static analysis based on the definition and usage of variables
- How it is performed?
  - Analysis of data use
    - The usage of data on paths through the program code is checked
- Use to detect data flow anomalies
  - Unintended or unexpected sequence of operations on a variable
- What is an anomaly?
  - An inconsistency that can lead to failure, but does not necessarily so.
  - May be flagged as a risk

The General Idea

- Data flow testing can be performed at two conceptual levels.
  1. Static data flow testing
  2. Dynamic data flow testing
- Static data flow testing
  - Identify potential defects, commonly known as **data flow anomaly.**
  - Analyze source code.
  - Do not execute code.
- Dynamic data flow testing
  - Involves actual program execution.
  - Bears similarity with control flow testing.
    - Identify paths to execute them.
    - Paths are identified based on **data flow testing criteria**.

Data Flow Anomaly

- Anomaly: It is an abnormal way of doing something.
  - Example 1: The second definition of x overrides the first.

```
x = f1(y);
x = f2(z);
```

- Three types of abnormal situations with using variable.
  - Type 1: Defined and then defined again
  - Type 2: Undefined but referenced
  - Type 3: Defined but not referenced

Data Flow Anomaly

- Anomaly: It is an abnormal way of doing something.
  - Example 1: The second definition of x overrides the first.

```
x = f1(y);
x = f2(z);
```

- Three types of abnormal situations with using variable.
  - Type 1: Defined and then defined again
  - Type 2: Undefined but referenced
  - Type 3: Defined but not referenced

Data Flow Anomaly

- Type 1: Defined and then defined again (Example 1 above)
  - Four interpretations of Example 1
    - The first statement is redundant.
    - The first statement has a fault -- the intended one might be: w = f1(y).
    - The second statement has a fault – the intended one might be: v = f2(z).
    - There is a missing statement in between the two: v = f3(x).
  - Note: It is for the programmer to make the desired interpretation.

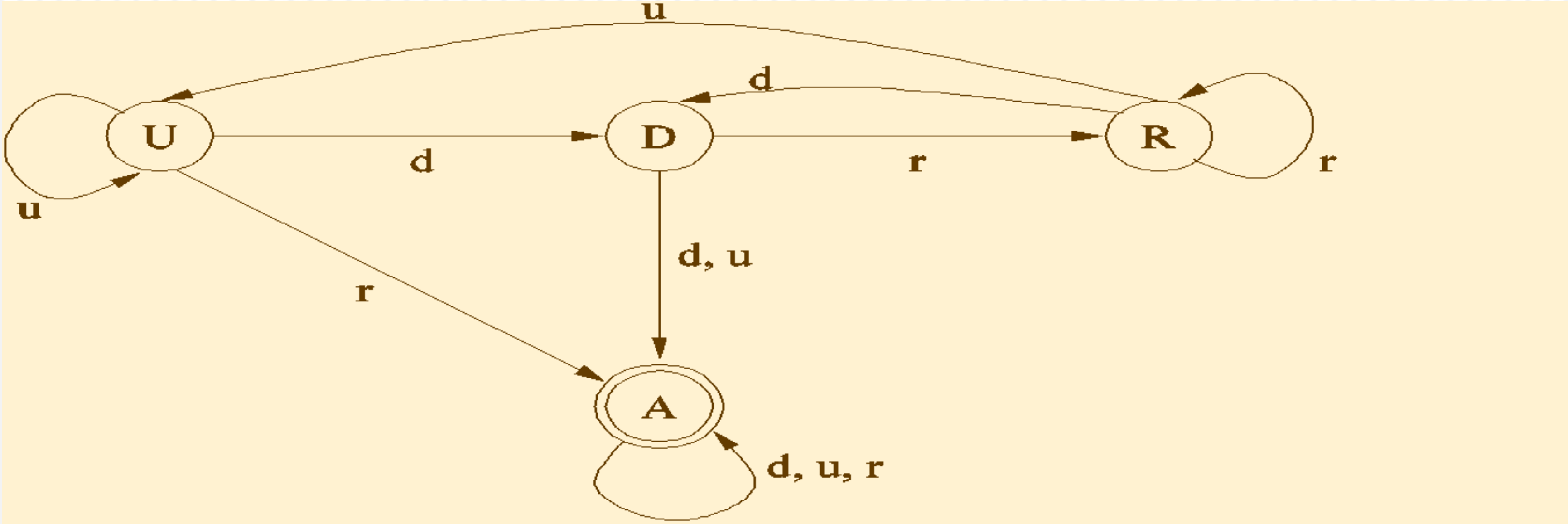
Data Flow Anomaly

- Type 2: Undefined but referenced
  - Example: x = x – y – w; /* w has not been defined by the programmer. */
  - Two interpretations
    - The programmer made a mistake in using w.
    - The programmer wants to use the compiler assigned value of w.
- Type 3: Defined but not referenced
  - Example: Consider x = f(x, y). If x is not used subsequently, we have a Type 3 anomaly.

Data Flow Anomaly

- The concept of **a state-transition diagram** is used to **model a program variable** to identify data flow anomaly.
- Components of the state-transition diagrams
  - The states
    - U: Undefined
    - D: Defined but not referenced
    - R: Defined and referenced
    - A: Abnormal
  - The actions
    - *d*: define the variable
    - *r*: reference (or, read) the variable
    - *u*: undefine the variable

State Transition Diagram Of A Program Variable



Legends:

States

U: Undefined
D: Defined but not referenced
R: Defined and referenced
A: Abnormal

Actions

d: Define
r: Reference
u: Undefine

Data Flow Anomaly

- Examples of data flow anomalies
  - Reading variables without previous initialization
  - Not using the values of a variable at all
- The usage of every single variable is inspected
  - (Type 1: *dd*), (Type 2: *ur*), (Type 3, *du*)
- Three types of usage or states of variables:
  - **Defined (d) : the variable is assigned a value**
  - **Reference (r) : the value of the variable is read and/or used**
  - **Undefined (u) : the variable has no defined value**

Data Flow Analysis

- Three types of data flow anomalies (Type 1: *dd*), (Type 2: *ur*), (Type 3, *du*)
  - **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) .

Data Flow Analysis: Example

- The following function is supposed to exchange the integer Value of the parameters Max and Min with the help of the variable Help,
- if the value of the variable Min is greater than the value of the variable Max

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

Data Flow Analysis: Example

- The following anomalies detected:
  - ur-anomaly of the variable Help
    - The domain of the variable is limited to the function.
    - The first usage of the variable is on the right side of an assignment.
    - At this time, the variable still has an undefined value, which is referenced .
    - There was no initialization of the variable when it was declared.

Data Flow Analysis: Example

- The following anomalies detected:
  - dd-anomaly of variable Max
    - The variable is used twice consecutively on the left side of an assignment and therefore is assigned a value twice.
    - Either the first assignment can be omitted or the use of the first value has been forgotten

Data Flow Analysis: Example

- The following anomalies detected:
  - du-anomaly of the variable Help
    - In the last assignment of the function the variable Help is assigned another value that cannot be used anywhere.
    - This is because the variable is only valid inside the function

The right one is…

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

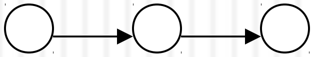
```
Help = Max;
Max = Min;
Min = Help;
```

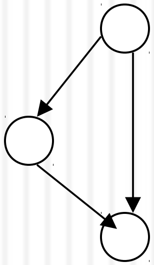
3.2.3.Control Flow Analysis

- What is control flow?
  - An abstract representation of all possible sequences of events (paths) in the execution of a component or system.
- A program structure is represented (modeled) by a control flow graph (CFG).
- CFG is a directed graph that shows a sequence of events (paths) in the execution through a component or system.
- CFG consists of nodes ○ and edges →
  - Node represents a statement or a sequence of statements
  - Edge represents control flows from one statement to another

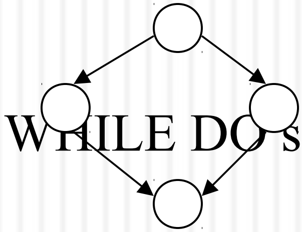
Control Flow Analysis

- Basic constructs of CFG: Sequence of assignment statements
  - IF … THEN … ELSE statement

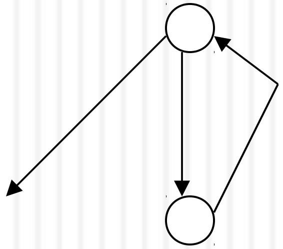


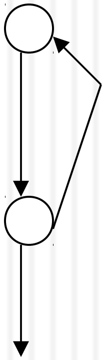


IF … THEN



  - WHILE DO statement



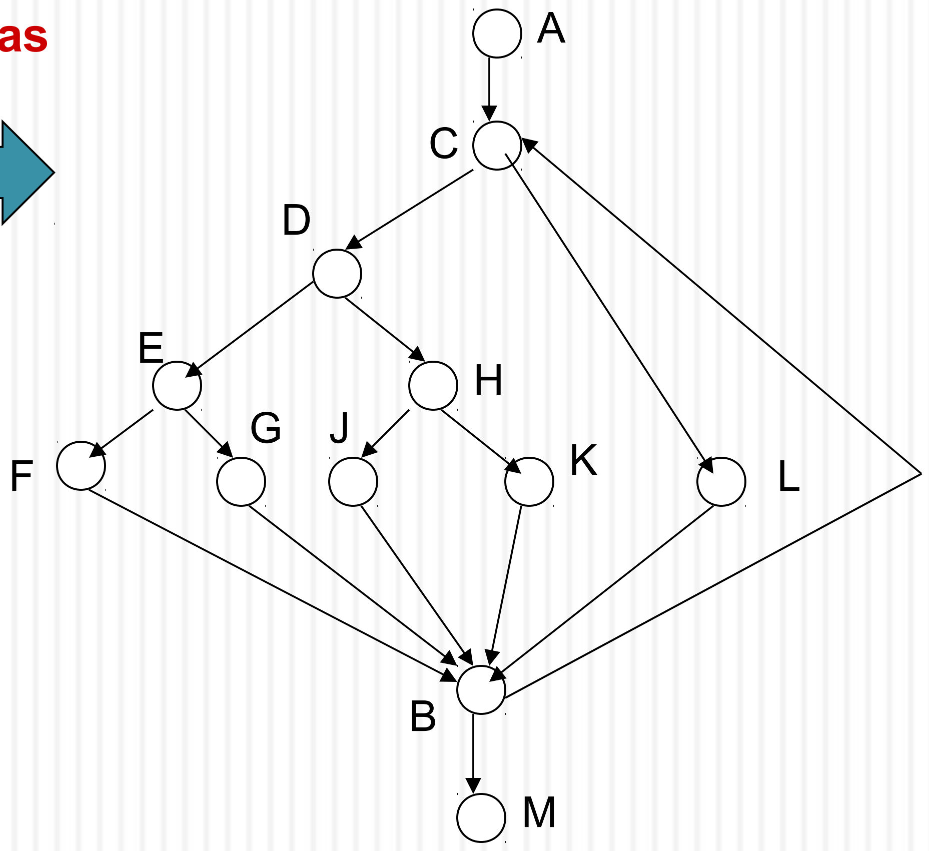


DO WHILE

Control Flow : example

```
A
DO
      IF C THEN
        IF D THEN
          IF E THEN F
          ELSE G
        ELSE IF H THEN J
               ELSE K
     ELSE L
WHILE B
M
```

Modeled as



Control Flow Anomalies

- Statically detected anomaly in the control flow of a test object.
- *Example*
  - Jumps out of a loop body
  - Program structure has many exits

3.2.4.Determining Metrics

- Quality characteristics can be measured with metrics.
- The intention is to gain a quantitative measure of software whose nature is abstract.
- Example:
  - McCabe's metric or cyclomatic complexity, V
  - Measures the structural complexity of program code
  - Based on CFG
  - $V(G) = e - n + 2$

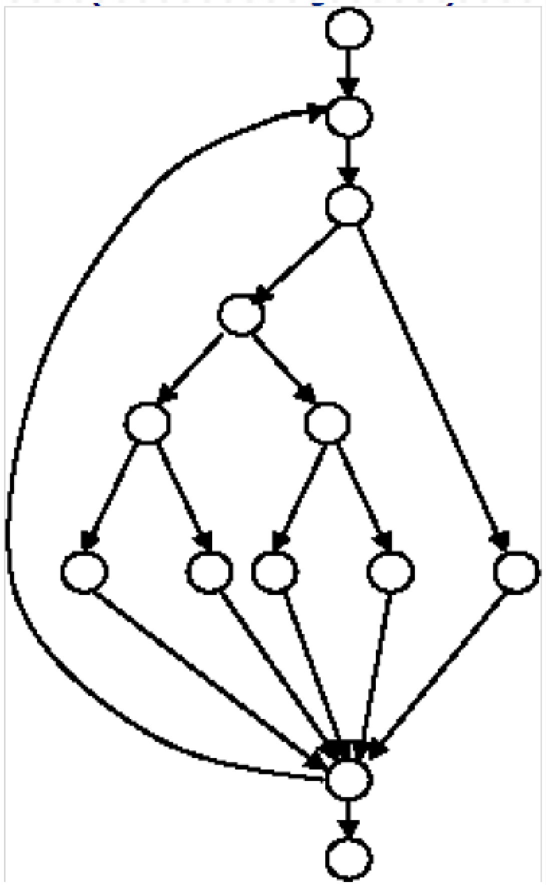
    where $V(G)$ is cyclomatic number of graph G

    e = number of edges in G

    n = number of nodes in G

Control flow graph for the calculation of the cyclomatic number

What is the cyclomatic number for this CFG ?



Determining Metrics

Example: for CFG in previous slide

V(G) = e - n + 2 = 16 – 12 + 2 = 6

V(G) higher than 10 can not be tolerated and rework of the source code has to take place

- □ *V(G) can be used to estimate the testability and maintainability.*
- □ *V(G) specifies the number of linearly independent paths in the program.*

Summary

- Static testing can be done to find defect and deviation using:
  - Structured group examinations
    - Reviews
      - Inspection, walkthrough, technical review, informal review
  - Static analysis using static analyzers
    - Compiler
    - Data flow analysis
    - Control flow analysis

Thank you

???