UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

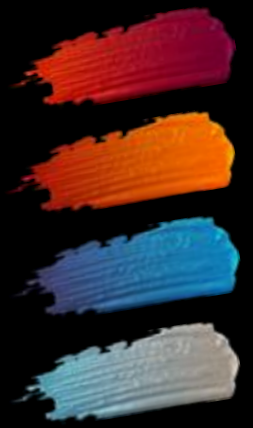
SERLAB
Software Engineering Research

Integrazione e Test di Sistemi Software

Static Testing

Azzurra Ragone

Dipartimento di Informatica - Università degli Studi di Bari
Via Orabona, 4 - 70125 - Bari
Tel: +39.080.5443270 | Fax: +39.080.5442536
serlab.di.uniba.it



What is Static Testing?

Static Testing is a type of a Software Testing method performed to check the defects in software without actually executing the code of the application (like in **Dynamic Testing**)

Static testing as «human testing»

Static testing (aka Software reviews) can be seen as "**human testing**".

Why?

- **Reading code** is part of a comprehensive testing.
- Software testing seeks to identify faults by causing failures.
- Some kinds of software reviews (e.g., code inspection) try to identify faults that, when executed, cause failures.

When?

- Before to start the computer-based testing.

Testing in software development

Static testing:

It does not require executing the code
It can be performed manually or by a set of tools

vs

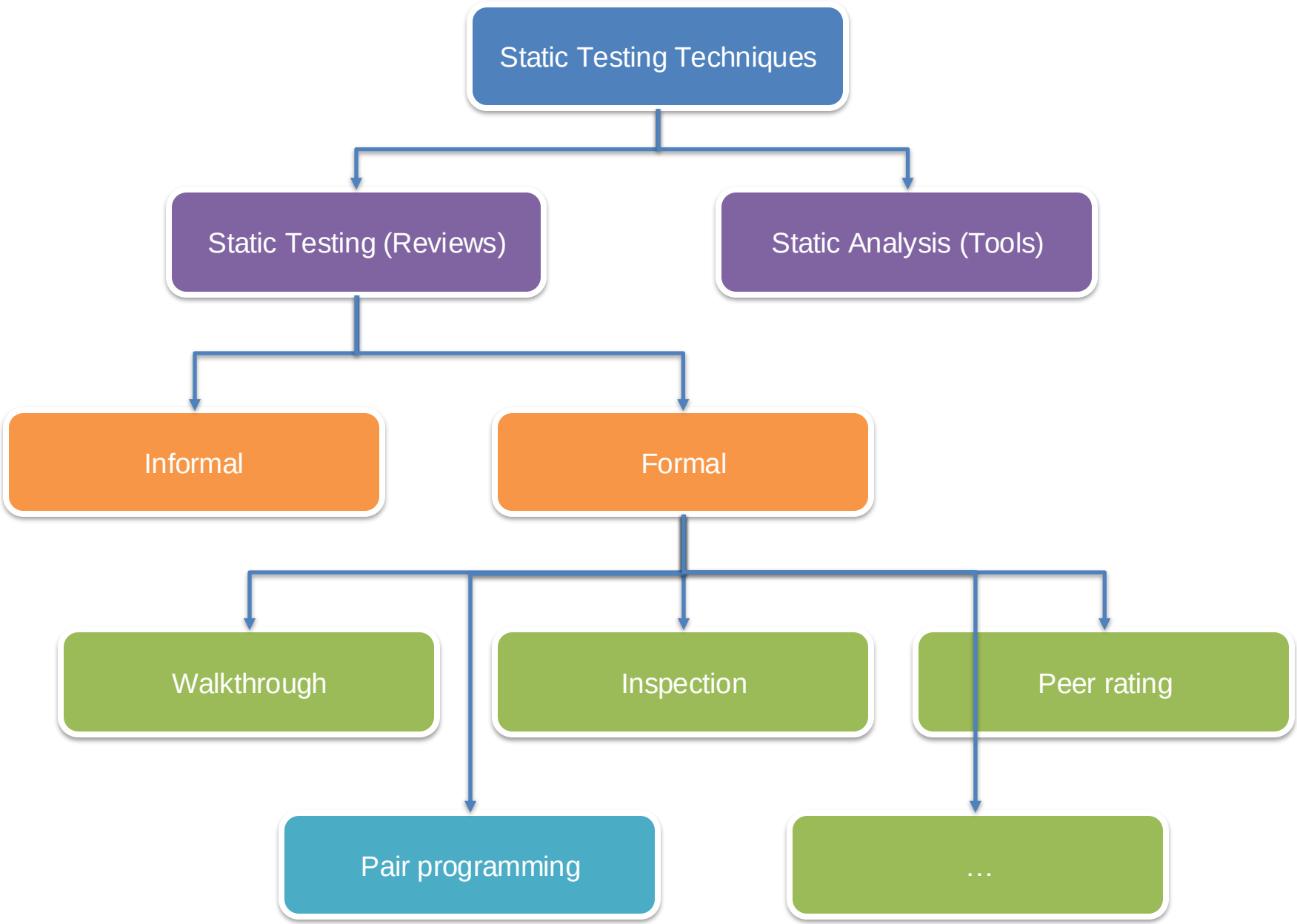
Dynamic testing:

It involves executing the software and evaluating its behavior during runtime (e.g. Unit, Integration, System and Acceptance Testing)

Benefit of «human testing»

Errors are found earlier (lower the cost)

At the beginning there is less pressure on the programmers: programmers make more mistakes if an error is found later


Static Testing Techniques
Static Testing (Reviews)
Static Analysis (Tools)
Informal
Formal
Walkthrough
Inspection
Peer rating
Pair programming
…

Static Analysis

Static Analysis includes the evaluation of the code quality. Different tools are used to do the analysis of the code and comparison of the same with the standard. It also helps in following identification of following defects:

(a) Unused variables
(b) Dead code
(c) Infinite loops
(d) Variable with undefined value
(e) Wrong syntax

Why static testing (reviews)?

K. Wiegers [1995] reports that, in an unnamed company, correcting a fault found by testing was **15 times** the cost to fix a fault found in an inspection---an example of static testing---, and this cost grew to **68 times** if the fault was reported by a customer

K. Wiegers [1995] also reports that, in another unnamed company, the cost to fix a fault found by inspection was **$146** while the cost to fix a fault found by customer was **$2900**

Slide courtesy of Simone Romano

Types of software reviews

Code inspection

Code walkthrough

Desk checking

Peer rating

Audit

Pair programming

Slide courtesy of Simone Romano

Software review

A test during which a work product (e.g., piece of **documentation** or **code**) is evaluated by one or more individuals to detect **issues** like code smells, faults, etc. (of course, we are interested in detecting faults)

Two primary approaches to code reviews:

- **Code inspection**
- **Code walkthrough**

Slide courtesy of Simone Romano

Code inspections and walkthroughs

They can be used at virtually any stage of software development, after a system or a component/unit is deemed to be complete

They both involve a team of people reading or visually inspecting code

This team participates in a meeting with the **objective to find issues** but **not to find solutions** to these issues (it's test not debug!)

Slide courtesy of Simone Romano

Code inspections and walkthroughs

Only one participant is the author of the program (it is ineffective testing our own program)

These human testing methods generally are effective in finding from **30 to 70** percent of the **logic-design** and **coding errors** in typical programs.

They are not effective, however, in detecting high-level design errors, such as errors made in the **requirements analysis** process.

Code inspections

A code inspection is a set of procedures and issue-detection techniques for group code reading

- **Peculiar** to code inspections is the use of **issue checklists**

It consists of an interrupted meeting, which should last from **90 to 120 minutes**

- The rate of evaluated code usually is **150 statements** per hour
- If a software system is too large for a code inspection meeting of 90-120 minutes, more code inspection meetings can be planned

Slide courtesy of Simone Romano

Code inspection team

It usually consists of four people:

- The **moderator:**
  - It is expected to be a competent programmer, but he/she is not the author of the code to be evaluated, and need not be aware of all the details of the system
  - Moderator duties include:
    - Distributing materials for, and scheduling, the inspection meeting
    - Leading the meeting
    - Recording all issues found
    - Ensuring that the issues are subsequently fixed
- The **producer**---the author of the code to be evaluated
- The other two people usually are the system's **designer** and a **test specialist**---he/she should be familiar with the most common programming issues (not only faults)

Slide courtesy of Simone Romano

Code inspection agenda

Some days before the inspection meeting, the moderator distributes some material (e.g., program listing and design specification) to the other participants, which should familiarize themselves with that material prior the meeting

During the inspection meeting, two activities occur:

1. The **producer** narrates, statement by statement, the logic of the code
   - During the discourse, other participants should raise questions to determine if issues exist
   - It is likely that the producer, rather than the other team members, will find most of the issues just because he/she is reading aloud the code to an audience
2. The code is analyzed with respect to a checklist of common programming issues

Slide courtesy of Simone Romano

Code inspection agenda

The moderator leads the discussion during the meeting and ensures that the participants focus their attention on **finding issues**, **not correcting them**

The goal is to find issues!

At the end of the meeting, the producer is given a **list of the issues uncovered**, which he/she has to fix after the meeting

If more issues are uncovered, or they might require substantial corrections, the moderator might plan another inspection meeting to re-evaluate the code

The list of issues is used to **update the issue checklist** to improve the effectiveness of future inspections

Slide courtesy of Simone Romano

Issue checklist

Any software company has its own issue checklist
- Some issues are concerned with **common errors**, others can be concerned with readability issues, etc.
- Issue checklists usually have categories

Example checklist:
- *Have all files been closed after use?*
- *Will every loop eventually terminate?*
- *Is each variable assigned the correct length and data type?*
- *Are comments accurate and meaningful?*
- *Does a reference variable have a value uninitialized?*
- *In loops, are there too many or too few iterations?*
- …

Slide courtesy of Simone Romano

Human aspects in code inspections

If the producer views the inspection as an attack on him/her and adopts a defensive posture, then the process will be ineffective

The results of an inspection should be confidential, shared only among the participants

E.g., *do not use those results to assume that a programmer is inefficient/incompetent*

Slide courtesy of Simone Romano

Side benefits of code inspections

- The producer receive valuable **feedback** concerning *programming style, choice of algorithms, and programming techniques*
- Any participant can gain knowledge from code inspections
  - E.g., programming styles, common programming issues, etc.
- Since the code of a producer is subject of evaluation, he/she is **fostered to write clean code** (e.g., readable code)

Inspection favors **cooperative behavior** and identification of the most **error-prone sections** of the program (more attention to these sections will be devoted in the automation testing phase)

Slide courtesy of Simone Romano

Code walkthroughs

Like a code inspection, a code walkthroughs is a set of procedures and issue-detection techniques for group code reading

However, **peculiar** to code walkthroughs is the use of **paper test cases**

It consists of an interrupted meeting, which should last from 1 to 2 hours

Slide courtesy of Simone Romano

Code walkthrough team

It usually consists of **three to five** people:

- The **moderator**
- The **secretary**
  - He/she records all issues found
- The **tester**
- The other two persons can be:
  - The **producer**
  - A highly experienced programmer
  - The person who will eventually maintain the system
  - A new programmer from a different project (to give a fresh, unbiased outlook)
  - A novice developer
  - …

Slide courtesy of Simone Romano

Code walkthrough agenda

Like in an inspection, the moderator provides the other participants with materials several days in advance, to allow them to familiarize themselves with it

During the inspection meeting:

1. The participants "play computer"
2. The tester brings a small set of **paper test cases**--- representative sets of inputs (and expected outputs) for the system (or component) to be evaluated
3. Each test case is **mentally executed**
   I.e., the test data are "walked through" the logic of the system
4. The state of the system (i.e., the values of the variables) is monitored on paper or a whiteboard

Slide courtesy of Simone Romano

Code walkthrough agenda

- Paper test cases must be **simple** and **few in number**
  - People execute systems at a rate that is much **slower** than a machine
  - Test cases themselves do not play a critical role
    - They are a vehicle for getting started and for questioning the producer about his or her **logic** and **assumptions**
- In most walkthroughs, the greater part of the issues are found during the process of **questioning** the producer, rather than by **executing** the test cases

Slide courtesy of Simone Romano

Human aspects in code walkthroughs and side benefits

Human aspects:

As in a code inspection, the attitude of the participants is critical

E.g., if the producer views the walkthrough as an attack on him/her and adopts a defensive posture, then the process will be ineffective

Side benefits:

The same as code inspection

E.g., participants can gain knowledge from code walkthrough (*identification of error-prone sections and education in errors, style, and techniques*)

Slide courtesy of Simone Romano

Desk checking

It can be viewed as a one-person inspection

- A person reads the code to be evaluated, checks it with respect to an issue list, and/or walks test data through it

To be effective, a person different from the producer should perform desk checking

- People are generally **ineffective** in testing their own programs

Less effective than code inspection and walkthrough

- In code inspection and walkthrough foster a **healthy, competitive environment** where participants try to demonstrate their **ability to identify issues**
- Such an environment is not present in desk checking

Slide courtesy of Simone Romano

Peer rating

It is a kind of review whose objective is **not to find errors**

It is a technique of **evaluating anonymous programs** in terms of their overall *quality, maintainability, extensibility, usability, and clarity*

A programmer is selected to serve as an **administrator** of the process

The administrator, in turn, selects approximately **6 to 20 participants** (namely programmers **with similar backgrounds** -- e.g., do not group Java programmers with C programmers)

Slide courtesy of Simone Romano

Peer rating

Each participant is asked to select **two** of his/her own programs to be **reviewed**

- One program should be representative of what the participant considers to be his/her **finest work**
- The other should be a program that the programmer considers to be **poorer in quality**

Once the programs have been collected, they are randomly distributed to the participants

- Each participant is given **four programs to review**---he/she does not receive his/her programs
- Two of the programs are the "finest" programs and two are "poorer" programs---the participant is not told which is which

Slide courtesy of Simone Romano

Peer rating

Each participant spends **30 minutes** reviewing each program and then completes an **evaluation form**

The evaluation form asks the reviewer to answer, on a scale from 1 to 10, questions like:

- *Was the program easy to understand?*
- *Was the high-level design visible and reasonable?*
- *Was the low-level design visible and reasonable?*
- *Would it be easy for you to modify this program?*
- *Would you be proud to have written this program?*

The reviewer also is asked for general comments and suggested improvements

Slide courtesy of Simone Romano

Peer rating

After the review, the participants are given the **anonymous evaluations** for their two contributed programs

- They also are given a **statistical summary** showing the overall and detailed ranking of their original programs across the entire set of programs
- And an analysis of how their ratings of other programs compared with those ratings of other reviewers of the same program

The purpose of the process is to allow programmers to **self-assess their programming skills**

- As such, the process appears to be useful in both industrial and classroom environments

Slide courtesy of Simone Romano

Audit

An independent examination of a work product that is performed by a **third party** to assess **compliance** with specifications, standards, contractual agreements, or other criteria

Slide courtesy of Simone Romano

Pair programming

A practice associated with agile processes like Extreme Programming (XP)

Two programmers write code together

These programmers alternate at the keyboard

- The programmer who is writing code is called **driver**
- The other programmer is called **navigator**

Pair programming

The **driver:**
- writes new code
- focus on detailed coding—syntax and structure

The **navigator** :
- directs the driver's work and reviews the written code
- highlights possible issues
- focuses on the flow of the work and reviews it in real-time

and then they discuss together alternative solutions

Pair programming

PP improves code **quality** and **readability**, and can speed up the **reviewing process**

PP **merges coding and inspection activities** so eliminating the gap between classic coding and inspection phases

The programmers frequently **alternate roles** so the written code is not "owned" by individual programmers, rather it a collective responsibility of both programmers

In XP, pair programming is usually carried out in normal (8-hour) workdays

To be effective, a constructive attitude of the programmers is needed, as well as a harmony between them

Require two programmers to work together **synchronously**

GitHub’s Copilot

GitHub's Copilot is an AI-powered developer tool to suggest code and entire functions in real-time, right from your editor

AI pair programming shifts the coding experience and reveals the importance of different skills: the emerging importance for developers to know how to **review** code as much as to **write** code

Less coding more reviewing

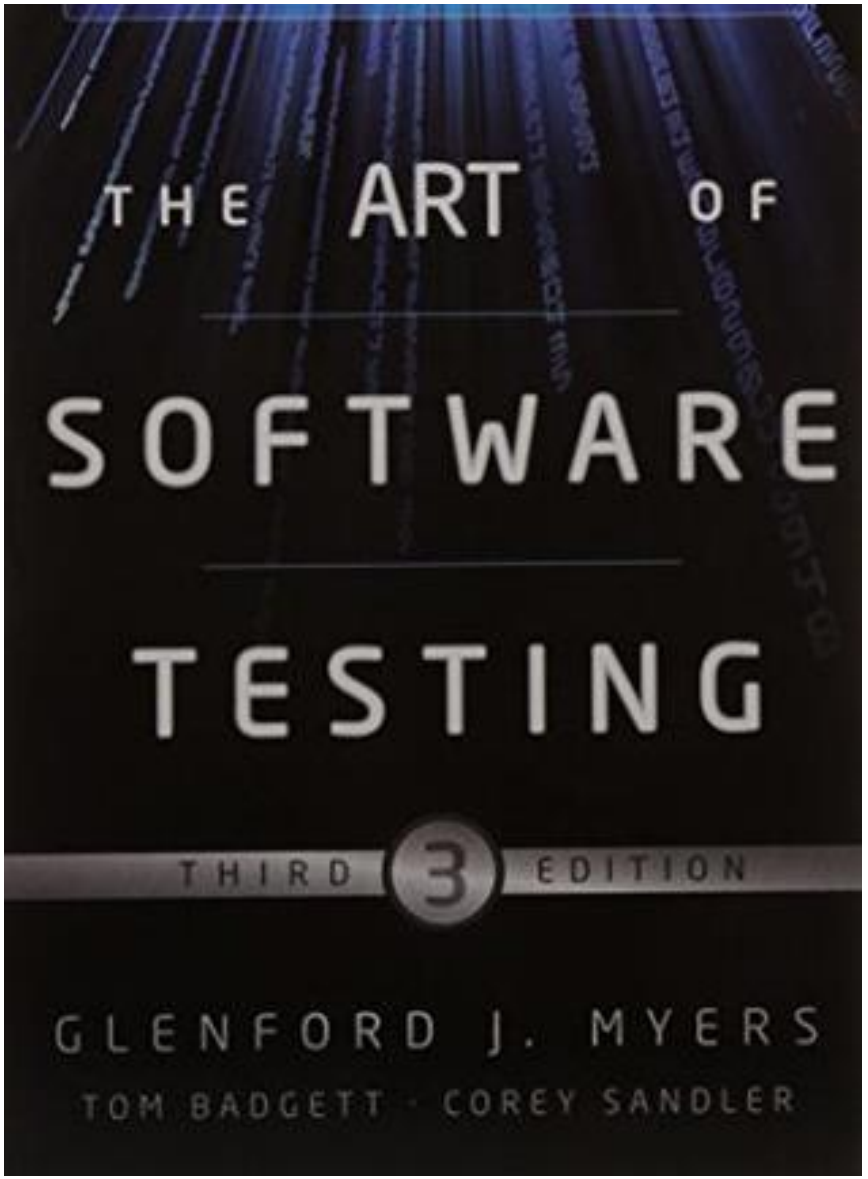
The risk is to understand less of how/why the code works

Reference book:

The Art of Software Testing

Chapter 3

Pdf available online



Interesting references

- **Taking Flight with Copilot.** Early insights and opportunities of AI-powered pair programming tools. https://queue.acm.org/detail.cfm?id=3582083

Azzurra Ragone

Department of Computer Science- Floor VI – Room 616
Email: azzurra.ragone@uniba.it