# Software Engineering 1

Validation And Verification

Arkadiusz Chrobot

Department of Information Systems, Kielce University of Technology

Kielce, January 11, 2026

Notes

## Outline

Notes

## Motto

”QA Engineer walks into a bar and he orders a beer. Orders 0 beers. Orders 99999999999 beers. Orders a lizard. Orders −1 beers. Orders a ueicbksjdhd. First real customer walks in and asks where the bathroom is. The bar bursts into flames, killing everyone.”

Source: Thomas' Digital Garden

Notes

## Introduction

*Validation and Verification* (*V&V*) is a software engineering process that aims to ensure the quality of developed product. The *Validation's* objective is to check if the software meets the clients' needs. The *Verification's* purpose is to make sure that the software adheres to its specification. The *Validation* answers the question [1]:

**Is the proper software being built?**

While the *Verification* gives an answer to the question:

**Is the software being built properly?**

Notes

## Software Quality

SWEBOK 3.0 defines the *Software Quality* as the capability of the product to satisfy stated and implied needs under specified conditions. It also describes the quality as the degree to which the software meets the established requirements, with the caveat, that these requirements represent the stakeholders' needs, wants, and expectations accurately.
The *Quality Control* (QC) is the process of checking the quality level, while the *Quality Assurance* (QA) is a set of all actions aiming to improve the quality in the software development cycle [2]. These terms are not interchangeable, so the term used in the motto is (probably) incorrect ☺

Notes

## Testing

The QA, QC and the V&V are performed with the use of *testing*. There are two kinds of testing: *static* and *dynamic*.
The *static testing* is a systematic examination of the software code and documentation, without running the code [1]. It may be performed manually or with the support of software tools. In the latter case, it is called *static analysis*. The most common form of static testing is a *review*, that can be either a *formal review* or an *informal review*, also known as a *peer review*. If the code execution is simulated during the review, then it is called a *desk checking*. The most formal types of reviews are the *inspection*, performed by the developers, and the *audit*, performed by a third party.
The *dynamic testing* requires an executable code to be available. In this case the test is an experiment where the software is run for some carefully chosen data sets and its behaviour is observed, as well its output data are verified.

Notes

## Scope of Static And Dynamic Testing

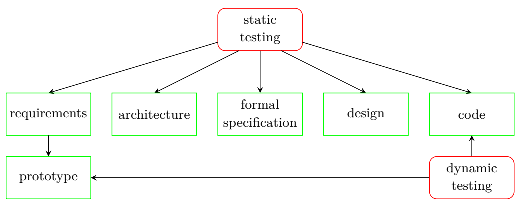

Figure: The scope of different testing forms

Notes

## Scope of Static And Dynamic Testing

As the figure 1 implies, the static testing may be applied to almost any artefact produced in the software development process. It allows the developers to discover defects earlier than the dynamic testing, which can be only applied to executable code. However, these two forms of testing are **not mutually exclusive**. Moreover, **they should be used together**. Only the dynamic testing is able to verify some software attributes, that can be examined at the runtime.

Notes

## Test Planning

Notes

As Edsger Dijkstra stated, tests may reveal defects, but they cannot ensure that there are no defects in the software. That is why testing is one of the most expensive activity in software development. The initial cost of static testing is high, but the benefits in later phases of development outweigh the expenses. On the other hand, the cost of dynamic testing in the early parts of software projects is almost non-existent (as there is nothing to test), but it grows rapidly later. One of the most important issues in testing is planning. In the waterfall model the tests were performed at the end of the project [3]. This quickly turn out to be wrong, because the programmers were lacking the feedback information provided by tests outcomes, to be able to remove defects as soon as possible. One of the earliest approaches to incorporate testing into the software development process is the V-Model (figure 2). In this model, with each artefact is associated a specific level of test.

## V-Model

Notes

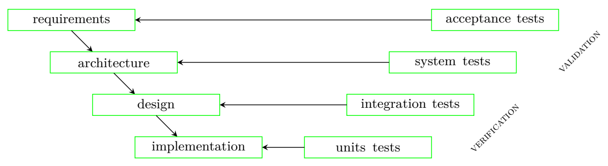

Figure: The V-Model

## Introduction to Dynamic Testing

Notes

The main goal of dynamic testing is detecting *defects*. The *defect* is a flaw in the software code that causes an *error* (also called a *failure* or *fault* or — colloquially — a *bug*) to occur. The *error* is an unexpected software behaviour or outcome. If a *test* reveals a defect, then its result is *positive*, otherwise it is *negative*.
The objective of dynamic testing may also be examination of the software runtime attributes, defined by nonfunctional requirements. These include efficiency, availability, and reliability. These kinds of dynamic tests are called *statistical testing*[1].
The distinction between statistical testing and testing for defects is fuzzy. Some defect could be found during statistical testing that were not detected by other tests. When testing for defects, experienced testers may estimate the efficiency and other attributes of tested software.

[1]Please don't confuse it with static testing. These are two different things.

## Finding Defects

Notes

Detecting a defect is a different process than finding its actual location in the code, although they are strictly interconnected. The outcomes of dynamic tests allow the programmer to analyse the symptoms of a defect. Locating it in the code is programming language and domain specific. It requires some experience and usage of such tools as debuggers and simulators. The figure 3 shows the connection between dynamic testing and finding and removing a defect.
After the defect is removed, the software should be tested again. The goal of this phase of testing is to ensure that the defect was really removed and not just covered up. In theory, all the tests that were performed until the detection of the defect should be repeated. It would be however too expensive. A better approach is to use the test case that allowed testers to detect the defect, and these test cases that relate to the software components that are affected by the repair. This method of is called *regression testing*.

## Finding Defects

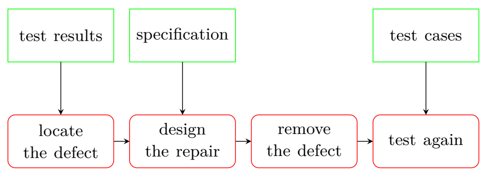

Figure: Testing, finding and removing defects

Notes

## Static Testing

### Comparison With Dynamic Testing

- + the total cost of static testing is less than the entire cost of dynamic testing,
- + in one session of a review, more defects are usually detected than in one session of dynamic testing,
- + the (possible) defects are not only detected but also located,
- + reviews make it possible to verify additional attributes of code, like readability, or discover domain-specific issues,
- + aside from the code, other artefacts may be reviewed,
- - the reviews cannot verify the runtime attributes of code (i.e. reliability, efficiency, and others),
- - manual static testing cannot be applied to a large code base.

Notes

## Inspections

The static testing, in a form of *inspections*, was proposed in the mid-1970s, by an IBM employee, Michael Fagan [1]. The inspection team is usually small (under 10 people). Each member has at least one of the following roles:

| **Role** | **Responsibility** |
|---|---|
| Author (Owner) | Provides the work product being inspected. Removes the discovered defects. |
| Inspector | Finds defects and other issues. |
| Reader | Interprets the inspected code or document. |
| Writer | Records the meeting outcomes. |
| Moderator | Plans and manages the inspection meeting. |
| Chief Moderator | Sets the inspection standards in the company. |

Notes

## Entry Criteria

Some of the inspection's *entry criteria* depend on the type of the inspected artefact. Others are the same in all possible cases. The *Moderator* is responsible for checking if all the entry criteria are met. Particularly, if the code is inspected she or he has to make sure that:

1. The current and correct specification of the reviewed code is available.
2. The team members know standards of the inspection.
3. The reviewed code development has been finished.
4. The defect types checklist is available.

Notes

## Inspection Process

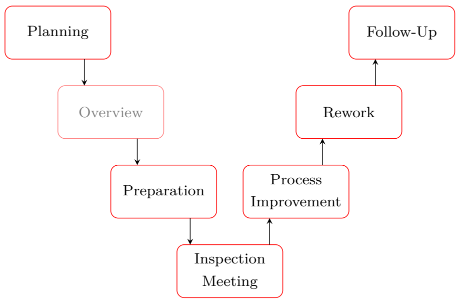

Figure: The inspection workflow

Notes

## Static Testing — Detected Defects

| Class of Defect | Example Control Check |
|---|---|
| Data Defects | Are all variables initialized before using? Do all constants have names? Is a buffer overflow possible? |
| Control Defects | Does every loop terminate? Are the conditions in conditional statements correct? |
| I/O Defects | Can unexpected input data cause a failure? |
| Interface Defects | Is the order of function's arguments correct? Do the data types of arguments and parameters match? |
| Memory Management Defects | Are the pointers used correctly? Is the allocated memory released when it is no longer used? |

Notes

## Static Testing — Summary

The reviews are not time-consuming. In case of the inspections the meeting takes about one hour and the preparations 1–2 hours. In each such a session, up to 500 lines of code may be analysed. Research shows that during the review, up to 60% of defects can be found, that would have to be detected latter, in the (much more expensive) dynamic testing process. If the static testing is supported by formal methods, then the score of discovered defects is even higher, up to 90%. Reviews are difficult to apply in companies with a highly competitive work culture.

Notes

## Static Code Analysis

The static testing of code can be preformed with the use of *Static Code Analysis Tools*. Such tools don't execute the software, but instead they examine its source code and find locations of possible defects. **They should be used with caution, because these tools don't know the context of the code, and may yield both false positive and false negative results.** Usually, Static Code Analysis Tools are stricter than compilers. They discover the same classes of defects as the manual review, but also they may check some additional code properties, like for example its structure and style. Those tools that are used for verifying the software security are called *SAST (Static Application Security Testing) Tools*.

Notes

## Examples of SCA Tools

LINT is the first ever developed tool for Static Code Analysis. It is dedicated for code written in the C language. Next, some other SCA tools were developed for other languages. Some of them supported more than one programming language. The first representative of the SAST tools is *splint*. The *checkstyle* is as an example of Static Code Analysis tool that checks if the code adheres to coding (including formatting) standards. Another free SCA tool is *ikos*, developed by NASA and dedicated for software written in C/C++.
Nowadays, the SCA tools are usually used in the CI/CD pipelines, for checking the code committed by programmers, before adding it to the repository. Examples of such tools include SonarQube (open source tool), Coverity (commercial tool by Synopsys), Klocwork (commercial tool by Perforce). All of them are multilingual and multipurpose.

Notes

## SCA — Detected Defects

The SCA Tools discover the same classes of defects as manual reviews do. However, they can also be applied to calculate some *code metrics* and to perform the *Data Flow Analysis* and the *Code Path Analysis*. The former traces the dependencies between input and output data. The latter finds all the code paths in the analysed software and statements that constitute these paths. The Code Path Analysis is especially useful in a dynamic testing method called *structural testing* (also *White-Box Testing* or *Transparent-Box Testing*).

Notes

## Bibliography

- Gerard O'Regan. *Concise Guide to Software Testing.* Cham, Switzerland: Springer, 2019.
- Adam Roman. *Thinking-Driven Testing: The Most Reasonable Approach to Quality Control.* Cham, Switzerland: Springer International Publishing AG, 2017.
- Neil Walkinshaw. *Software Quality Assurance: Consistency in the Face of Complexity and Change.* Cham, Switzerland: Springer International Publishing AG, 2017.

Notes

## Questions

?

Notes

## The End

Thank You for Your attention!

Notes

Notes

Notes

Notes