

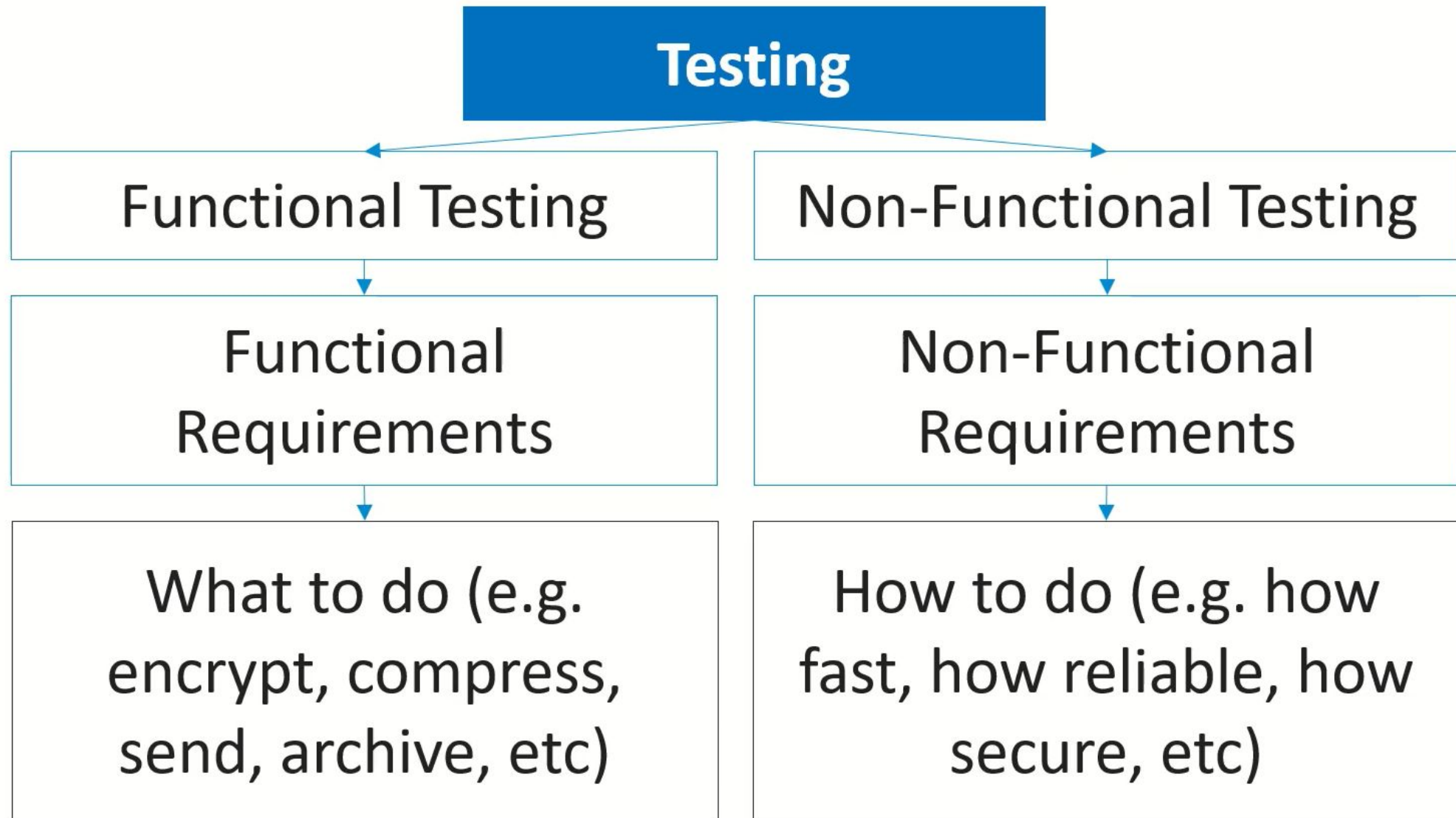
# SOFTWARE TESTING CLASSIFICATION

ENGLISH WITH JULLY

May 26, 2021

# TESTING CLASSIFICATION

# Two Fundamental Approaches: Functional and Non-Functional Testing



**Functional testing** – testing conducted to evaluate the compliance of a component or system with functional requirements.

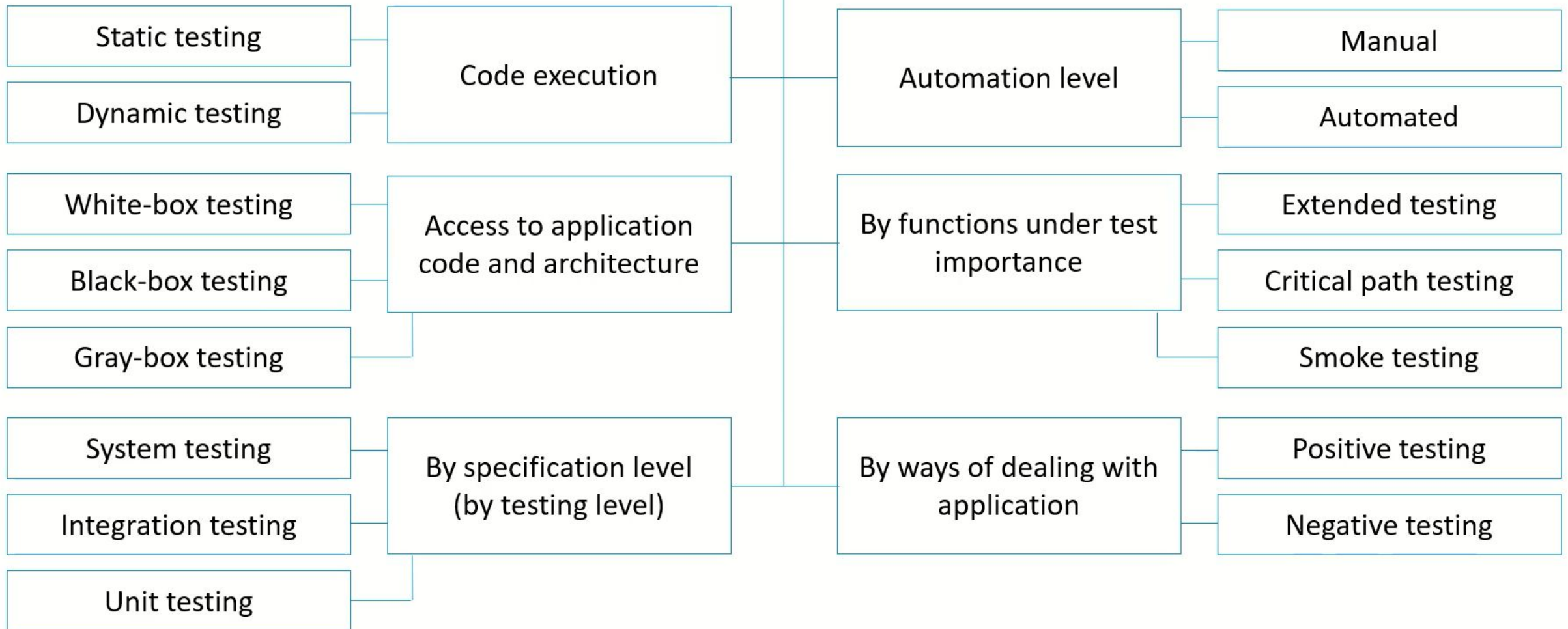
**Functional requirement** – a requirement that specifies a function that a component or system must be able to perform.

**Non-functional testing** – testing conducted to evaluate the compliance of a component or system with non-functional requirements.

**Non-functional requirement** – a requirement that describes how the component or system will do what it is intended to do.

# General Classification Approach

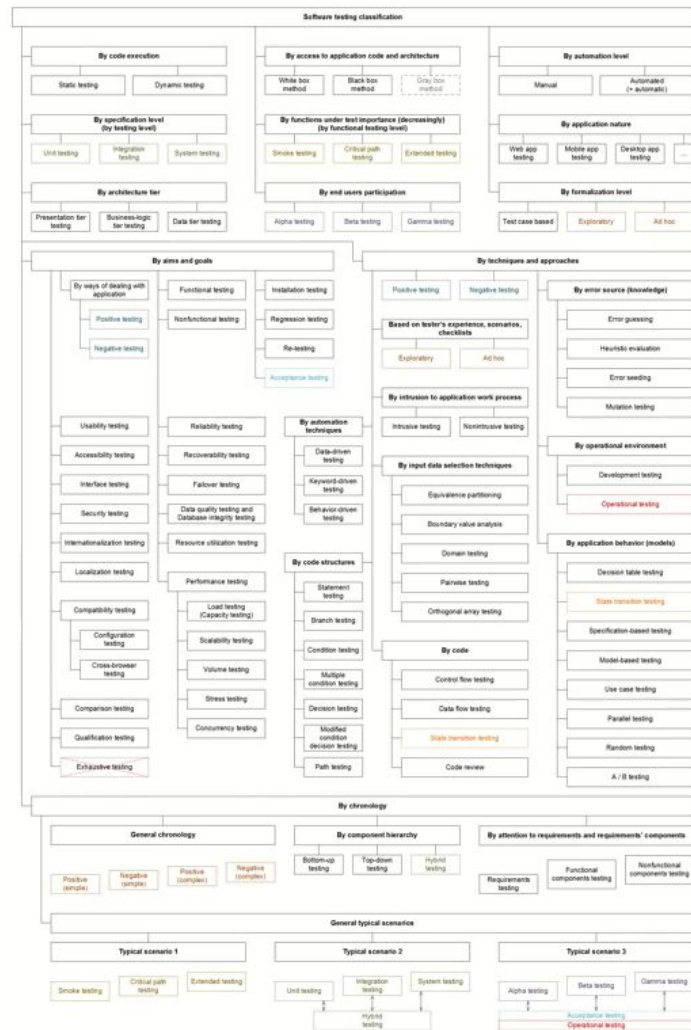
## Classification by...



# General Classification Approach: There Is Much More!

Please, visit:  
[http://svyatoslav.biz/urls/stc\\_en/](http://svyatoslav.biz/urls/stc_en/)

[http://svyatoslav.biz/wp-pics/software\\_testing\\_classification\\_en.png](http://svyatoslav.biz/wp-pics/software_testing_classification_en.png)





## Classification by code execution

**Read and remember!**

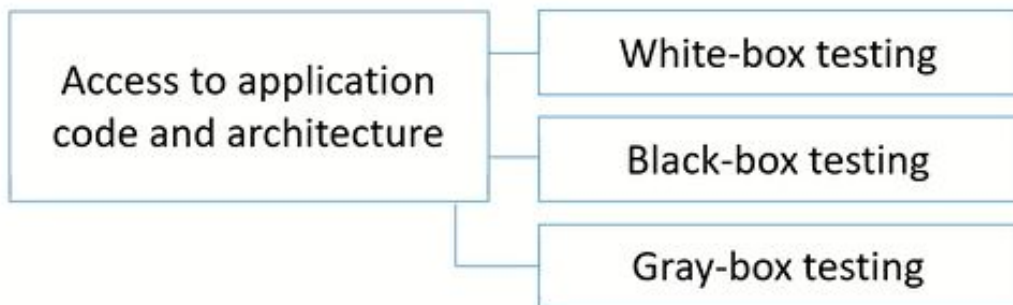


**Static testing** – testing a work product without code being executed.

**Dynamic testing** – testing that involves the execution of the software of a component or system.

... by access to application code ...

**Read and remember!**

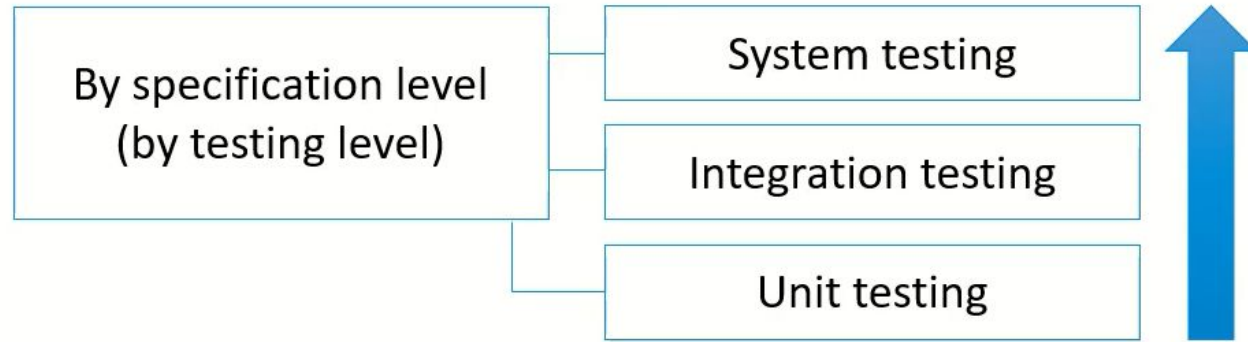


**White-box testing** – testing based on an analysis of the internal structure of the component or system.

**Black-box testing** – testing, either functional or non-functional, without reference to the internal structure of the component or system.

**Gray-box testing** – testing, which is a combination of Black-box and White-box (i.e. the internal structure is partially known).

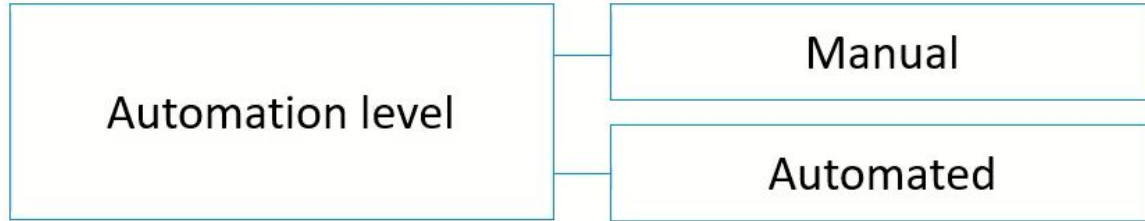




**Unit testing** – the testing of individual hardware or software components.

**Integration testing** – testing performed to expose defects in the interfaces and interactions between integrated components or systems.

**System testing** – testing an integrated system to verify that it meets specified requirements.

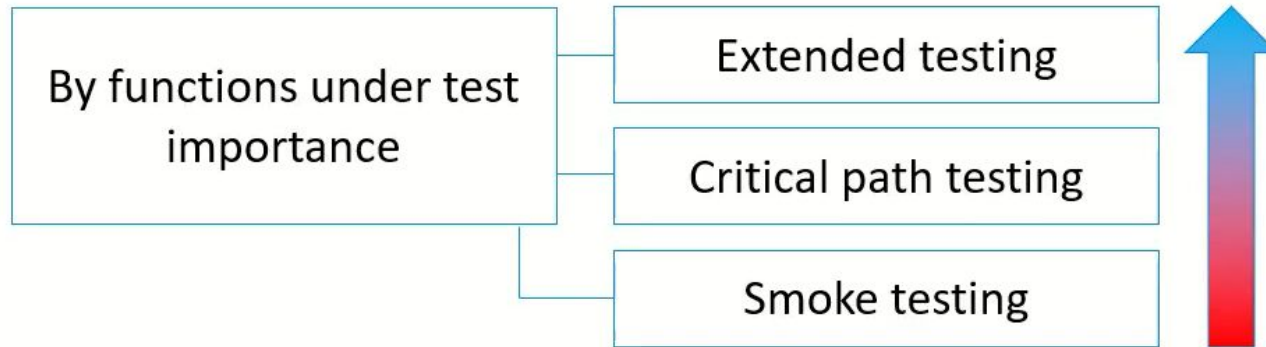


**Manual testing** – testing performed by the tester who carries out all the actions on the tested application manually.

**Automated testing** – the use of software to control the execution of tests, the comparison of actual outcomes to predicted outcomes, the setting up of test preconditions, and other test control and test reporting functions.

... by functions under test importance

**Read and remember!**



**Smoke test** – a subset of all defined test cases that cover the main functionality of a component or system, the most crucial functions.

**Critical path test** – test cases that cover the functionality used most of the time by the majority of users.

**Extended test** – test cases that cover the “nice-to-have” functionality (not used most of the time by the majority of users).

... by ways of dealing with application

**Read and remember!**

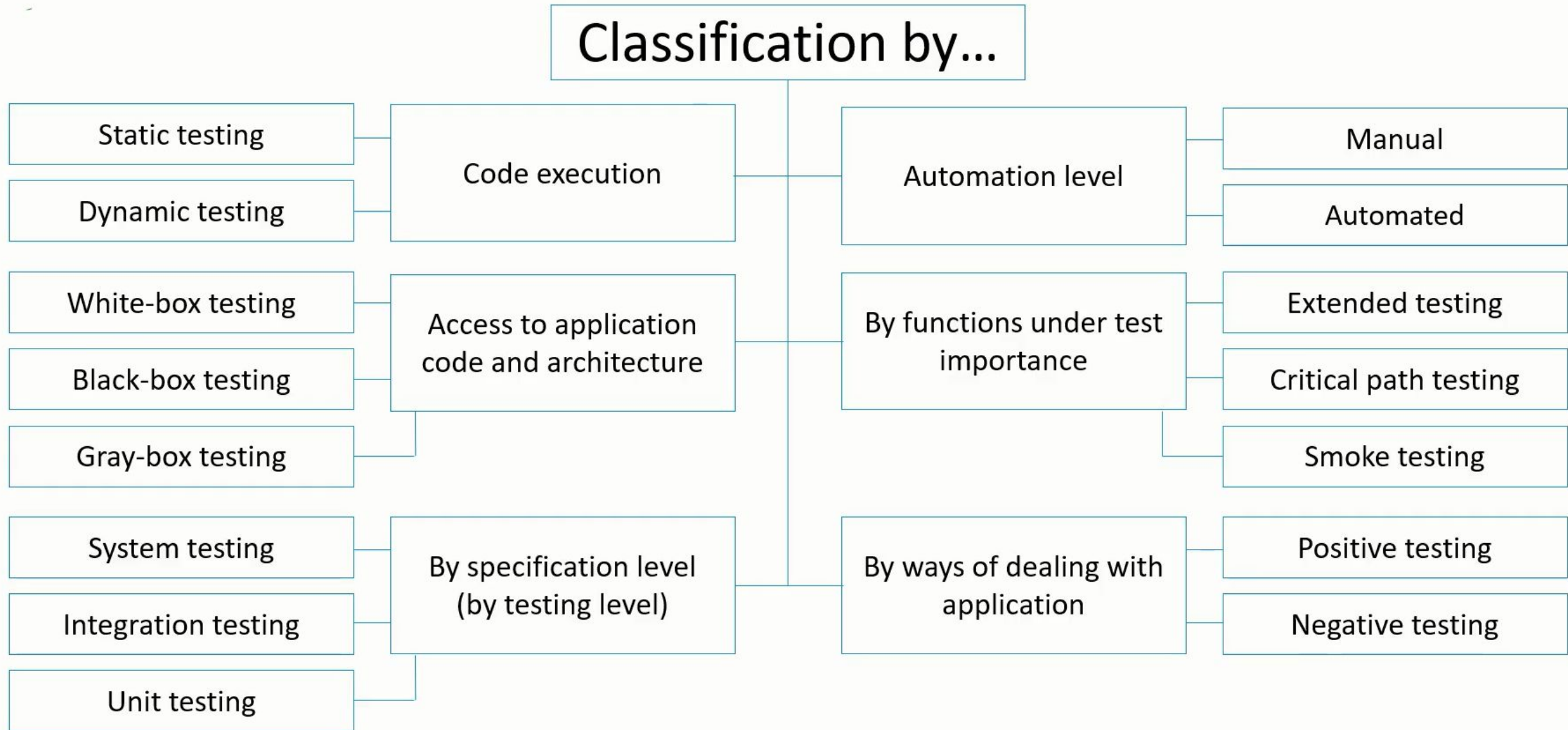


**Positive testing** – testing process where the system validated against the valid input data and valid set of actions.

**Negative testing** – tests aimed at showing that a component or system does not work.



Let's look at the big picture one more time...



# TEST PLANNING

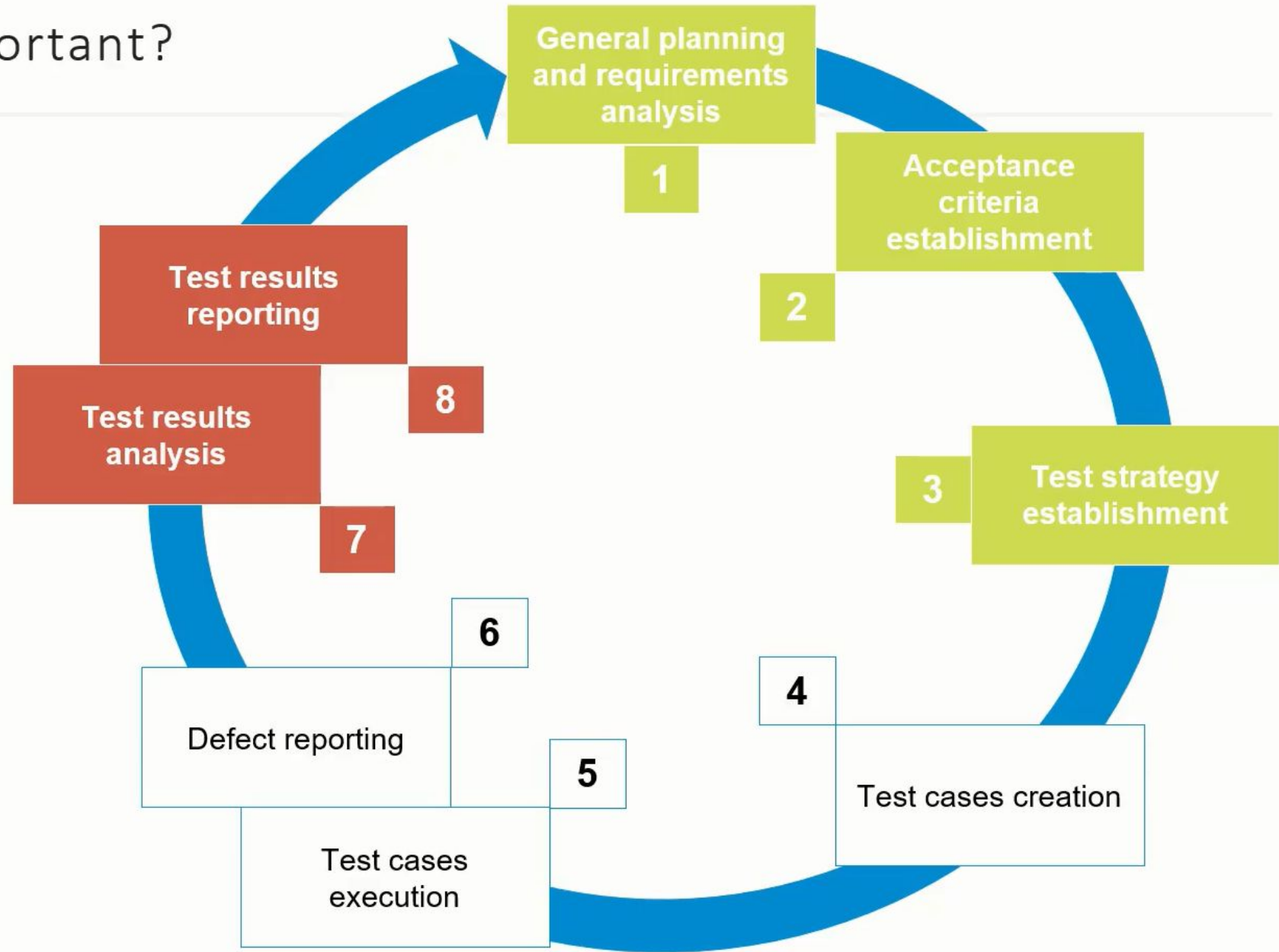


**Test planning** – the activity of establishing or updating a test plan.

**Test plan** – documentation describing the test objectives to be achieved and the means and the schedule for achieving them, organized to coordinate testing activities.

# Why is Test Plan so important?

**Test Plan** is the basis for all further activities, it is a map to the defined destination, a starting point to the reporting and so on and so forth...



## Planning Tasks and Goals

Estimation of work scope and complexity

Designation of resources and ways to acquire them

Definition of schedule and milestones

Risk mitigation and countermeasures definition

Distribution of duties and responsibilities

Coordination of team-effort between teams/projects/groups

### Test Plan helps us to...

Get better results with  
less effort

Optimize resources  
usage, avoid waste

See the progress at any  
given moment

Understand what to do  
in any situation

Get better  
understanding between  
people

Cooperate better

# TEST PLAN SECTIONS

**Test plan** – documentation describing the test objectives to be achieved and the means and the schedule for achieving them, organized to coordinate testing activities.



# General Test Plan Sections Overview

Project scope and main goals

Requirements to be tested

Requirements NOT to be tested

Test strategy and approach

Criteria

Resources

Schedule

Roles and responsibilities

Risk evaluation

Documentation

Metrics

...

**Project scope and main goals** – a very brief description of the purpose of the application development.

This section is reminiscent of business requirements, but here the information is presented in an even more condensed form with an emphasis on the most important tasks.

**E.g.:** *Correct automated conversion of text documents in different source encodings to one destination encoding with performance significantly higher than human performance during the same actions.*

**Requirements to be tested** – the list of functional and / or non-functional requirements to be tested.

Not each and every requirement should be tested. Here we list those ones that should be covered with test-cases.

E.g.:

- *UR-1.\*: smoke test.*
- *UR-2.\*: smoke test, critical path test.*
- *UR-3.\*: critical path test.*
- *BR-1.\*: smoke test, critical path test.*
- *QA-2.\*: smoke test, critical path test*

**Requirements NOT to be tested** – the list of functional and / or non-functional requirements NOT to be tested.

As not each and every requirement should be tested, here we list those ones that should NOT be covered with test-cases. For each requirement we write the reason why shouldn't we test it.

**E.g.:**

- *SC-1: the application is a console one by design.*
- *SC-2, L-1: the application is developed with proper PHP version.*
- *QA-1.1: this performance characteristic is at the bottom border of typical operations performance for such applications.*



**Test strategy and approach** – the description of the testing process in terms of the methods used, approaches, types of testing, technologies, tools, etc.

This description allows us to use the most effective and efficient way to achieve project goals in terms of quality.

**E.g.:**

*Due to the team cross-functionality, a significant contribution to quality improvement can be expected from the code review combined with manual testing using the white box method. Unit-testing will not be applied due to extreme time limitations.*

**Criteria** – the list of miscellaneous testing criteria, such as acceptance criteria, entry criteria, suspension criteria, resumption criteria, exit criteria, etc.

Usually, each criterion has a reference to proper metric. Otherwise the decision on criterion fulfillment may be too subjective.

**E.g.:** *Testing resumption criteria: more than 50% of bugs found during the previous iteration are fixed (see “Ongoing defects fixed percentage” metric).*



**Resources** – the list of miscellaneous project resources, such as software, hardware, staff, time, finance, etc.

Usually, finance resources are too confidential to be included in test-plan, so only a reference to the budget is mentioned here.

E.g.:

- *Software: four virtual machines (two with Windows 10 Ent x64, two with Linux Ubuntu 18 LTS x64), two PHP Storm licenses (latest version available).*
- *Hardware: two standard workstations (8GB RAM, i7 3GHz).*

**Schedule** – in fact, this is a kind of calendar with milestones and periods marked on it.

When the project is long enough (months and years long) there is no need to have a detailed schedule for the far future: the schedule may be adjusted and filled with details once we reach one milestone or another.

- E.g.:
- *25.05 – requirements testing and finalizing.*
  - *26.05 – test-cases and scripts for automated testing creation.*
  - *27.05-28.05 – main testing stage.*
  - *29.05 – testing finalization, reporting.*

**Roles and responsibilities** – the list of all key project roles (related to the testing process) with their key areas of responsibility.

Though this list is rather unified for most projects, there may be some special cases, so this section usually is not omitted.

- E.g.:**
- *Senior developer: participation in requirements testing and code review.*
  - *Tester: documentation creation, test-cases execution, participation in code-review.*



**Risk evaluation** – the list of risks that are likely to arise during the project. For each risk there is a probability evaluation and some options for resolving the situation.

There are a lot of typical risks (actual for every project) so no need to list such risks here. This section is for risks specific (or even unique) to the project.

- E.g.:
- *Personnel (low probability): if any team member is inaccessible, we can contact the representatives of the “Cataloger” project to get a temporary replacement (the commitment from the “Cataloger” PM John Smith was received).*

**Documentation** – the list of test documentation with details on who should prepare it, when, how, etc.

This section is extremely useful when onboarding a new project member. As sometimes documentation list may consist of dozens of items, it's good to have them listed in one place.

E.g.:

- *Requirements. Responsible person – tester, deadline – 25.05.*
- *Test cases and defect reports. Responsible – tester, creation period – 26.05-28.05.*
- *Test result report. Responsible person – tester, deadline – 29.05.*



**Metrics** – the list of numerical characteristics of quality indicators, methods for their evaluation, formulas, etc.

This section is actively referenced by “Criteria” section. Metrics have their dark side too, but in general it is better to have some objective way to tell the current project situation.

E.g.:

*Test cases success percentage:*

$$T^{SP} = \frac{T^{Success}}{T^{Total}} \cdot 100\%, \text{ where}$$

*$T^{SP}$  – percentage of successfully passed test cases,*

*$T^{Success}$  – quantity of successfully passed test cases,*

*$T^{Total}$  – total quantity of executed test cases.*

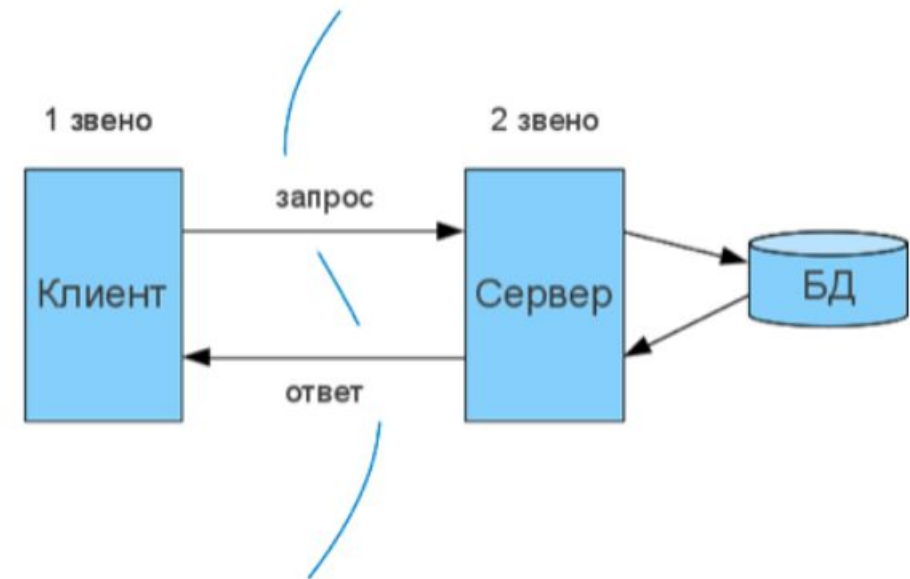
# CLIENT-SERVER ARCHITECTURE

## Two-tier architecture - the distribution of three basic components between two nodes (client and server).

**A web application** is a client-server application in which the browser is the client and the web server (broadly speaking) as the server.

**The client** is a browser, **a web server** is a server that accepts HTTP requests from clients and issues HTTP responses to them.

**A database** is an information model that allows you to orderly store data about an object or a group of objects that have a set of properties that can be categorized.



[more information here](#)

ESTIMATI

ON

**Estimation** - the score determines how much money, effort, resources, and time it will take to build a particular system or product.

## Method in Agile – Story points

**Story points** measure the effort it takes to complete a product backlog item or any other piece of work.

<b>How much is known about the task</b>	Everything	Almost everything	Something	Almost nothing	Nothing	Nothing
<b>Dependencies</b>	None	Almost none	Some	Few	More than few	Unknown
<b>How much work effort</b>	Less than 2 hours	Half a day	Up to two days	Few days	Around a week	More than one week
<b>Story Points</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>5</b>	<b>8</b> Should be split into smaller items	<b>13</b> Must be split into smaller items



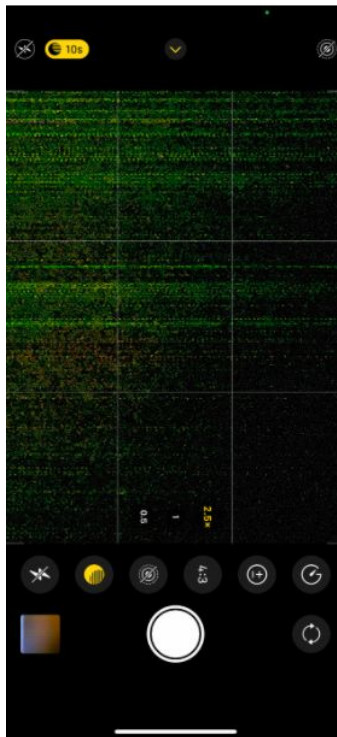
# CONFUSING MOMENTS

# Feature of Bug?

**A feature** is slang, the name of certain signs of an object or phenomenon. In other words, these are its features, unusual properties.

**The bug** means an issues in a program or application.

## BUG



## FEATURE



# ERROR/MISTAKE/FAULT/ BUG/DEFECT/FAILURE?

If someone makes **an error** or **mistake** in using the software, this may lead directly to a problem - the software is used incorrectly and so does not behave as we expected.

**A Bug** is the result of a coding Error or Fault in the program which causes the program to behave in an unintended or unanticipated manner. **Bugs** arise from mistakes and errors, made by people, in either a program's source code or its design

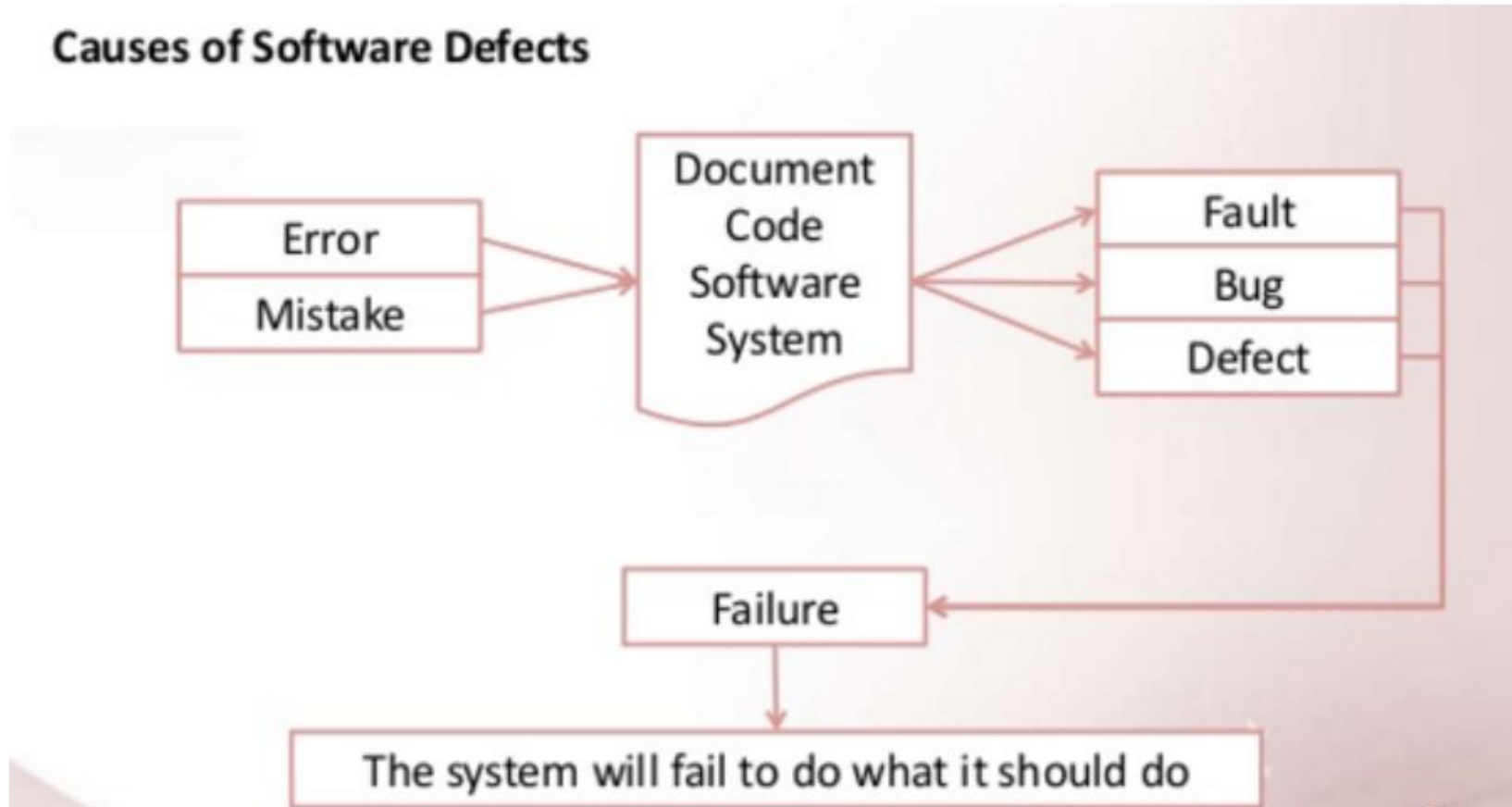
**A Defect** is a deviation from the Requirements.

**Failure** is a deviation of the software from its intended purpose. It is the inability of a system or a component to perform its required functions within specified performance requirements

# Causes of Software Defects

IMPORTANT!

## Causes of Software Defects



|

# THANK YOU!

**ENGLISH WITH JULLY**

May 26, 2021