QA Basics Revision

Software testing, Test scenarios and Test Cases, Bugs and Bug Tracking, Test Levels, Test Types

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

http://softuni.bg

Table of Contents

You Have Questions?

Software University

sli.do

#QA-Fund

Understanding Testing

What is Testing?

- **Exercising** software
  - To **verify** that it meets **specified requirements** and to **identify** any **errors**
- **Analyzing** a software item
  - To **detect discrepancies** between existing conditions and required conditions
- **Evaluating** the various features of the software

Software Testing

- Software Testing is a **way to**:
  - **Assess** and **ensure** the quality of software
  - **Minimize** the **risk** of software **failures** during operation
- The typical software testing process **includes**:
  - Test **Planning** and **Analysis**
  - Test **Design** and **Execution**
  - Test **Reporting** and **Evaluation**
  - Test **Maintenance**

Testing Objectives

- The **main objectives** of software testing include
  - **Preventing defects** in the software
  - **Verifying** that all specified **requirements** are met
  - **Confirming** the expected **behavior** of the software
  - **Reducing** the risk of inadequate software functionality
  - **Providing** valuable **information** for stakeholders
  - **Ensuring compliance** with contractual, legal, and regulatory requirements

Importance of Software Testing

Importance of Software Testing

Software University

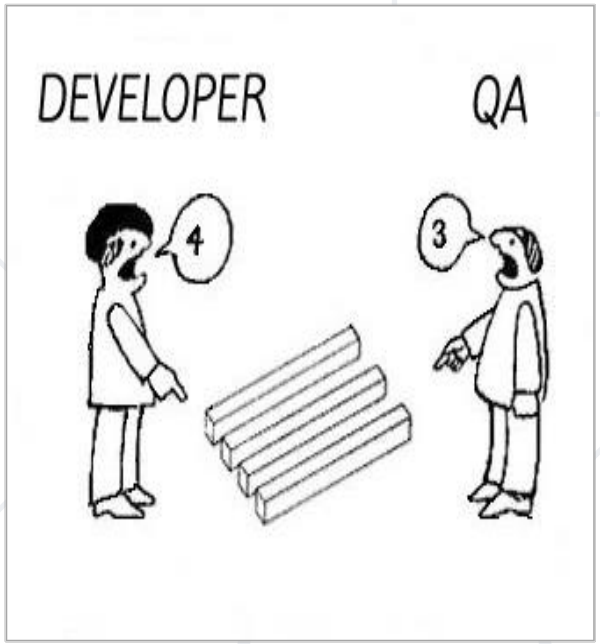
- Software Testing plays a **vital role** in:
  - **Ensuring the quality** of individual components and entire systems
  - **Verifying** that the software meets all contractual and legal **requirements**
  - **Reducing** overall **costs** significantly through early identification and fixing of issues
  - In critical applications (e.g., healthcare, aviation, etc.), it can **save lives** by preventing harmful software errors

Psychology of Testing

Human Psychology in Testing

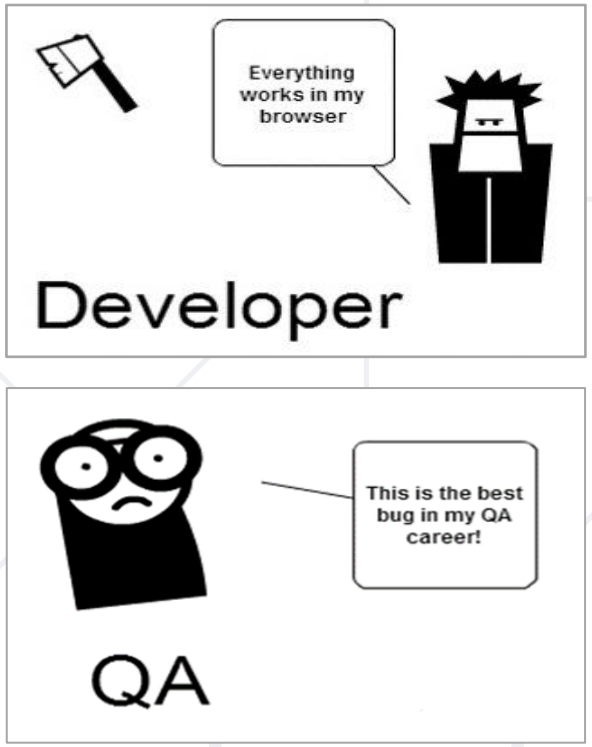
- Identifying defects may be perceived as **criticism**
- **Confirmation bias** can make it challenging to **accept** feedback
- As a result, testing can be viewed as a **destructive** activity
- Good **communication skills** are a must in order to **avoid conflict** between developers and QA

QA vs Devs



- **QA testers**
  - Face a perception of being 'destructive' – only happy when finding faults
  - Require excellent communication skills, tact, and diplomacy
  - Need to be multi-talented, balancing technical, testing and team skills

Devs vs QA



- **Developers**
  - Perceived as highly creative - their code is fundamental to the creation of the system
  - Not stereotypically strong communicators
  - Often specialize in one or two skills (VB, C++, JAVA, Python)

Seven Testing Principles

The Philosophy of Software Testing

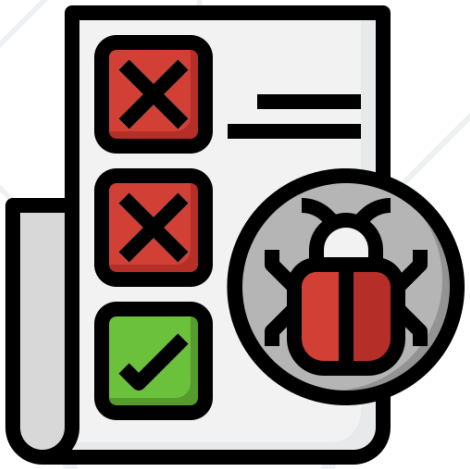
Seven Testing Principles (1)

"Testing shows presence of defects, not their absence"

- Testing can show that **defects** are **present**
- Cannot prove that there are **no defects**
- Appropriate testing **reduces** the **probability** for defects

Seven Testing Principles (2)

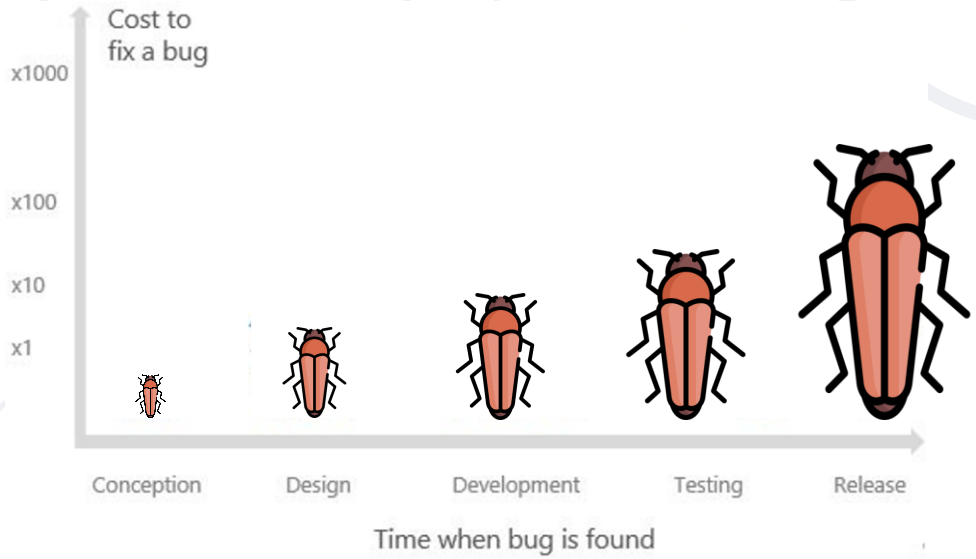
"Exhaustive testing is impossible"



- **All combinations** of inputs and preconditions are usually almost **infinite** number
- Testing everything is **not feasible**
- **Risk analysis** and **priorities** should be used to focus testing efforts

Seven Testing Principles (3)

"Early testing saves time and money"



- Testing activities shall be started as **early as possible**
  - And shall be focused on predefined objectives
- The **later** a bug is found – the **more** it **costs**!

Seven Testing Principles (4)

"Defects cluster together"

- Testing **efforts** should be focused **proportionally**
  - To the expected and later observed defect density of modules
- A **small** number of modules usually **contains most** of the **defects** discovered

Seven Testing Principles (5)

"Beware of the pesticide paradox"

- Same tests **repeated** over and over again, tend to **lose** their **effectiveness**
  - **Previously undetected** defects remain **undiscovered**
- **New** and **modified** test cases should be developed

Seven Testing Principles (6)

"Testing is context dependent"

- Testing is done **differently** in different contexts
- Safety-critical software should be tested differently from an e-commerce site

Seven Testing Principles (7)

"Absence of errors is a fallacy"

- Finding and fixing defects itself does not help in these cases:
  - The system built is **unusable**
  - Does not fulfill the users' **needs** and **expectations**

Test Scenarios and Test Cases

Outlining and Detailing the Testing Journey

Test Scenario

- **What is a "Test Scenario"?**
  - Any functionality, feature, or user story that can be tested
  - Often referred to as the "**story under test**" or "**feature under test**"
- **Why do we need Test Scenarios?**
  - Allow **complex systems** to be **broken down** into manageable, testable **parts**
  - Serve as a **quick tool** for **estimating** the testing work effort
  - Facilitate understanding of the **end-to-end functioning** of the software program

Test Scenarios and Test Cases

- One **test scenario** typically encompasses **multiple test cases**
- **Example**:
  - **User Story**: Users should be able to log in
  - **Test Scenario**: Login with username and password
  - **Test Cases**:
    - Login with valid username and password -> Expected Result: **Success**
    - Login with invalid username or password -> Expected Result: **Error message**

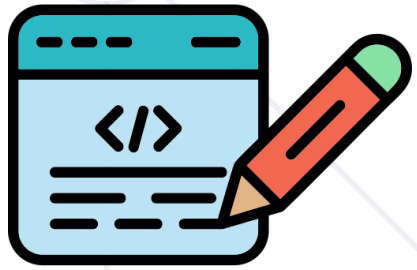
Test Cases

Software University

- **What is a "Test case"?**
  - A **sequence of actions** executed to verify a specific use of a feature or functionality, representing a **particular execution path**
  - It often includes **specific input** and **expected output conditions**
- **Why do we need Test Cases?**
  - Allow us to **compare expected to actual results** for a certain execution path, aiding in the identification of discrepancies
  - Help us **examine the functioning** of a software component with **specific input** and under **certain conditions**

Test Cases (2)

- **Sequence of steps** designed to verify correct behavior
- To test a certain scenario, at least two tests cases are required:
  - **Positive Test** - verifies the system behaves as expected in a normal situation
  - **Negative Test** - checks the system's response to unexpected or invalid inputs
- A **comprehensive Test Case** consists of:
  - **Title** (optional description)
  - **Steps** to follow
  - **Expected results**

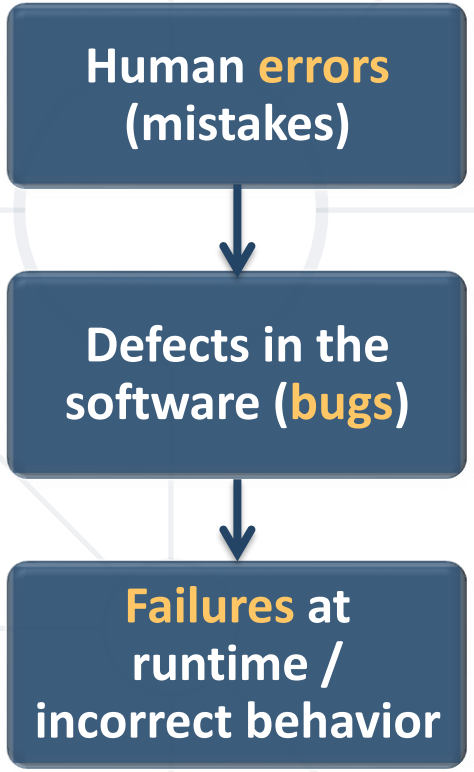


Bugs and Bug Tracking

Understanding and Managing Software Defects

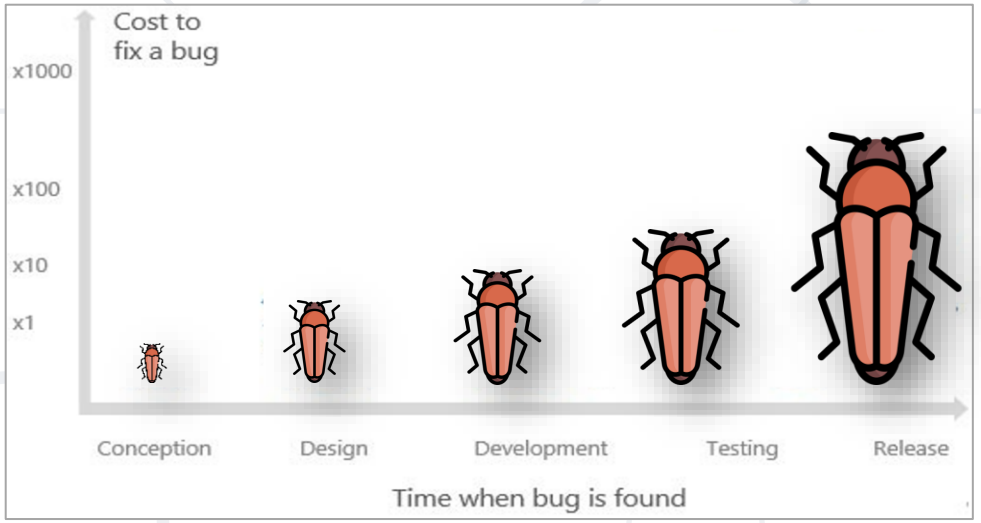
Software Defects

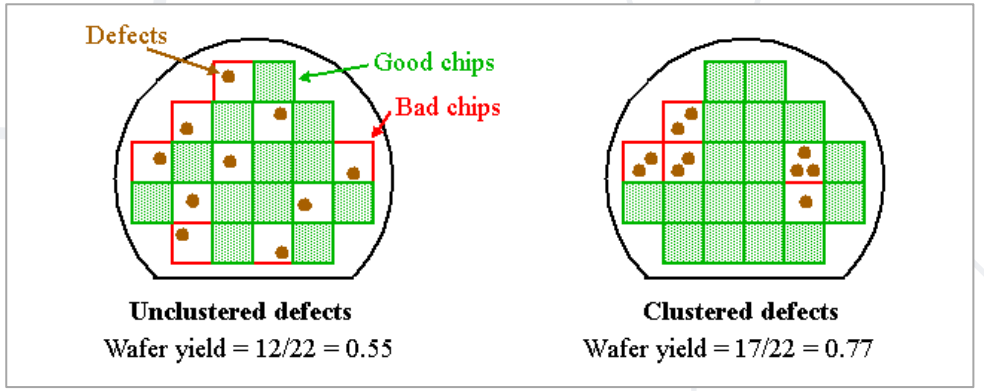
- Humans are prone to **making errors**, which can lead to defects in software
- **Defects**, or **bugs**, can exist in the **program code** or could be **mistakes** in the **requirements**, **design**, or **other** project components
- If a **defect** is **executed**, it might **cause a failure**, making the software do something it shouldn't or fail to do what it should
- The **primary goal** of QA and software testing is to **identify** these defects
- Implementing **Automated Testing** with **Continuous Integration** / **Continuous Deployment** (CI/CD) can significantly reduce the occurrence and impact of defects



Bug Fixing Importance

- Adhering to the **"Seven Testing Principles"** can underscore the significance of bug resolution:
  - "**Early testing saves time and money**"
    - Detecting bugs early in the development process reduces costs
  - "**Defects cluster together**"
    - Typically, 80% of problems are found in 20% of the modules, underlining the importance of focused testing





Bug Tracking in Software Testing

Software University

- What is **Bug Tracking**?
  - A **process** of capturing, reporting, and managing data about bugs in a software project
  - Enables teams to **keep track** of reported bugs, their **status**, and **resolution**
  - Facilitates **collaboration** between team members and enhances productivity
  - Helps in understanding common issues, enables **preventive measures** for future projects
- **Popular Tools** for Bug Tracking:
  - JIRA, Bugzilla, Mantis, etc.

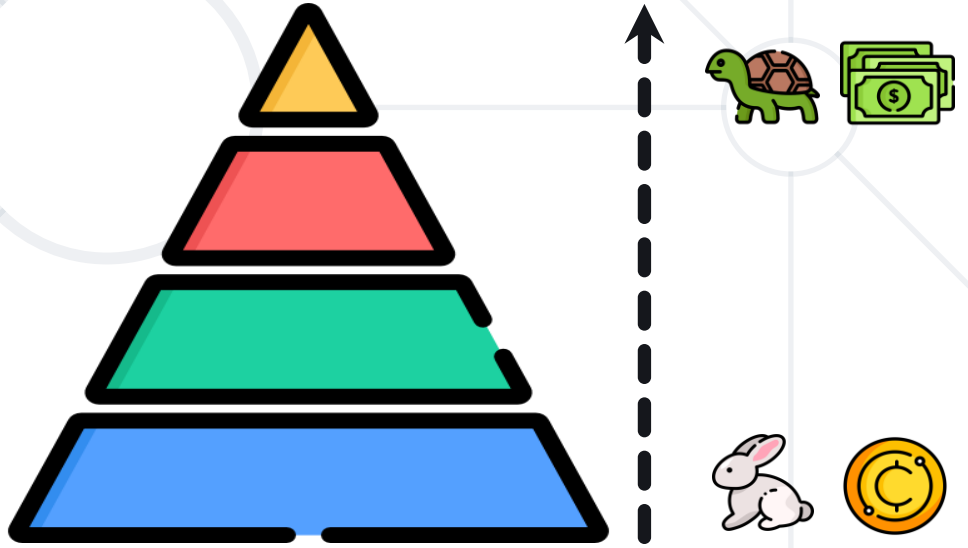
Test Levels

Unit / Integration / System / Acceptance Testing

Test Levels

- Groups of **test activities**
- Each **level** is an **instance** of a **test process**
- **Corresponding** to the software at a **different development level**

- **Test levels include:**
  - **Acceptance** testing
  - **System** testing
  - **Integration** testing
  - **Unit** testing



Unit Testing

- Tests **individual components** of the software such as functions, methods, procedures, modules, or objects
- **Done** during the **coding** phase, typically by the **developers**
- Done in **isolation**

- **Example:**
  - A function that checks user's age for certain conditions

```
function isAdult(age) {
  if (age >= 18) {
    return true;
  } else {
    return false;
  }
}
```

Integration Testing

- **Units** or **components** tested as a **group**
- Performed by developers, testers, or special integration teams
- Checks if **components collaborate correctly**
- Exposes faults in interfaces and interactions
- Two **sub-levels**:
  - **Internal Integration** Testing: "Integration test **in the small**"
  - **External Integration** Testing: "Integration test **in the large**"

Integration Testing: Example

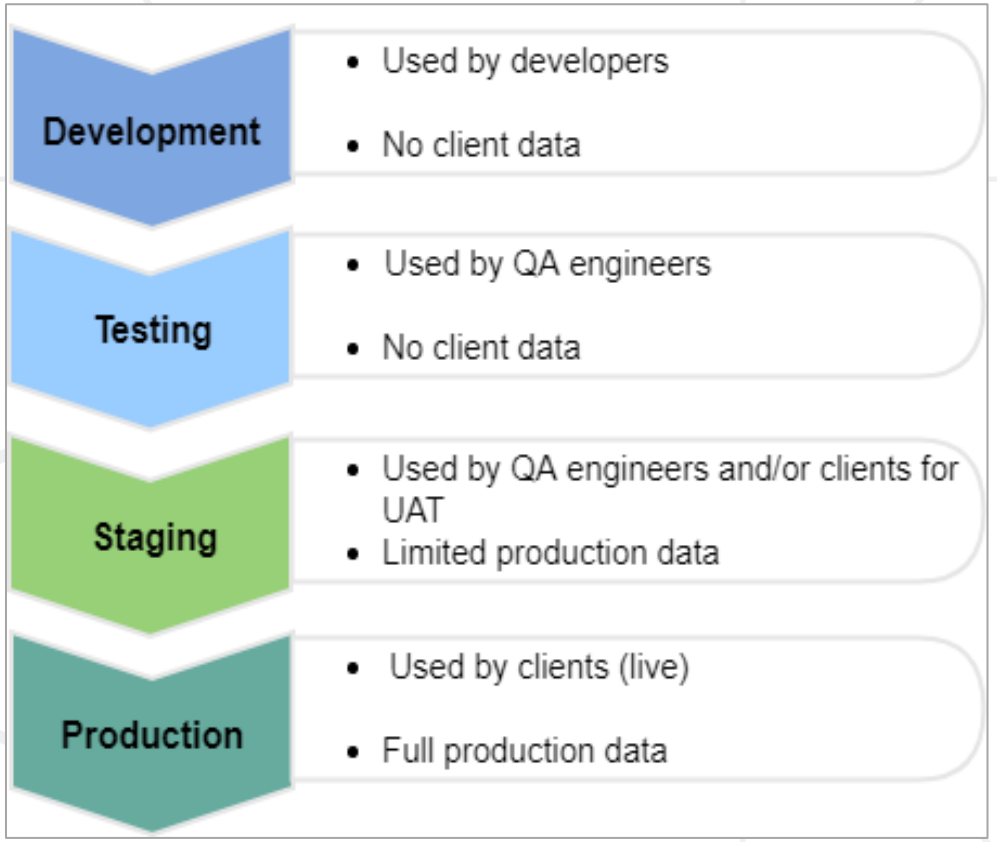
- GitHub **Example**
- Modules: **Home Page**, **Login Page**, **User Dashboard**
- Each module is unit tested
- **Integration** testing **checks** if **they work together**:
  - Test if the login link opens the login form
  - Test if a successful login shows the User Dashboard
  - Test if after logout, the User Dashboard is unavailable

System Testing

- It **focuses** on the **System as a whole**:
  - **System behavior**: What the system is doing (e.g., Is the system working as intended?)
  - **System capabilities**: How the system is doing it (e.g., Is the system reliable, secure, efficient?)
- Performed by executing end-to-end tasks
- Carried out exclusively by QA Engineers
- Looks at the system from the end-user perspective
- Covers **both functional and non-functional** aspects

System Testing Environment and Example

- Requires a **dedicated environment**:
  - **Mimics** the end-user environment
  - **Specifically designed** for system testing
- **Example:** An e-commerce application
  - **Test end-to-end user flow**: searching a product, adding it to cart, making a payment, and viewing order history



Acceptance Testing

- **Final** testing level, usually **pre-deployment**
- Validates **end-to-end business flow**
- **Conducted** by:
  - Business team members (**Alpha testing**)
  - End-users (**Beta testing**)
- **Follows** operational instructions
- Ensures **compliance** with **contractual** and **regulatory** guidelines

Acceptance Testing Importance and Example

Software University

- **Verifies** system functionality, pre-deployment
- Main goal: **Working business flow**
- Focus is not on cosmetic errors
- Aligns **actual system behavior** with **client expectations**
- **Example**: Microsoft Windows
  - **Alpha testing**: Internal testing in Redmond
  - **Beta testing**: Testing by selected end users globally

Test Types

Functional vs. Non-Functional Testing

Test Types

- Group of **test activities** that **test specific characteristics** of a software system
- Test types are divided into **two** main groups:
  - **Functional** testing
    - Answers to the question **"What?"**
    - Validates software **actions**
  - **Non-functional** testing
    - Answers to the question **"How?"**
    - Validates the **performance** of the software

Test Types: Example

- An **online banking software** example
- **Functional testing** includes:
  - Test login with valid and invalid credentials
  - Verify accurate fund transfer between accounts
  - Check timely processing of scheduled bill payments
- **Non-functional** testing focus on the **performance and security**:
  - Check system security against unauthorized access and threats
  - Measure system performance under normal and peak loads
  - Evaluate user interface for intuitiveness, readability, and ease of use

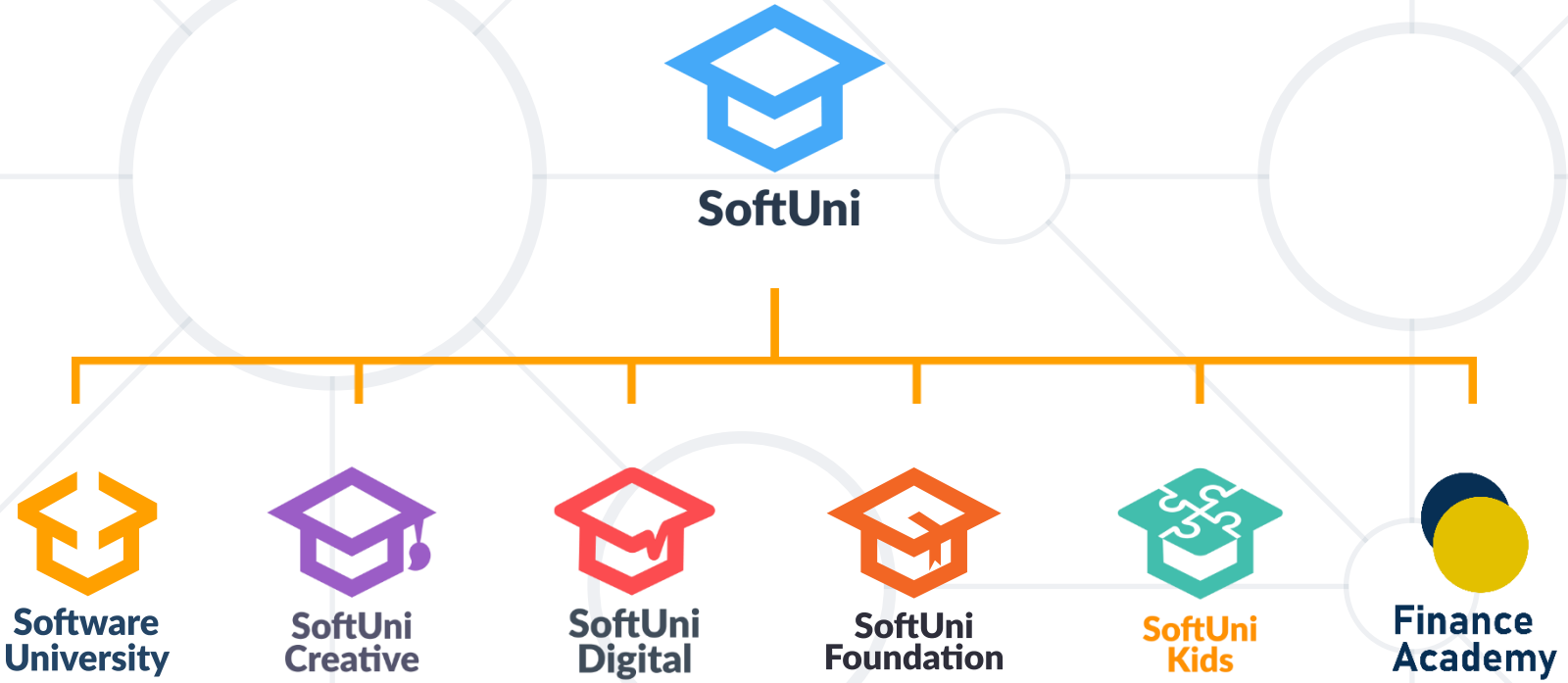
Test Types & Test Levels

- **Test types** can be **applied** at (**m**)**any test levels**
- Example: testing the "**register user**" scenario
  - **Functional** tests:
    - Valid user info, invalid user info, duplicated user info
  - **Non-functional** tests:
    - Performance (100k users), reliability (1 user per second for 24 hours), UX test (is it user friendly)

Summary

- Explored software testing: its **definition**, **objectives**, **importance** and **psychology**
- What are "**The 7 Testing Principles**"?
- Highlighted the role of **test scenarios** and **test cases**
- Emphasized the importance of **early detection** and resolution of **software defects**
- Discussed different testing levels: **unit**, **integration**, **system**, and **acceptance testing**
- Distinguished between **functional** and **non-functional testing**

Questions?



SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

Postbank
Решения за твоето утре

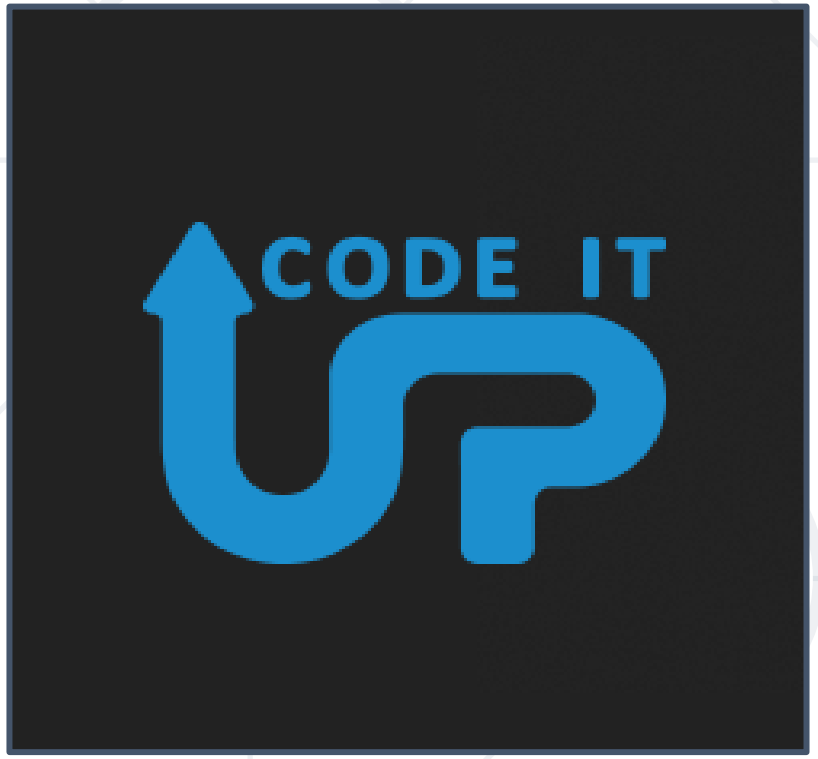
BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners

Software University



License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg