

softserve | quality
management office

softserve

INTRO TO SELENIUM

Oleksandr Synyava

softserve

AGENDA

- Installation
- Architecture
- Drivers
- API for interaction with a browser
- Capabilities/options
- Tabs

LET'S GO

softserve

INSTALLATION

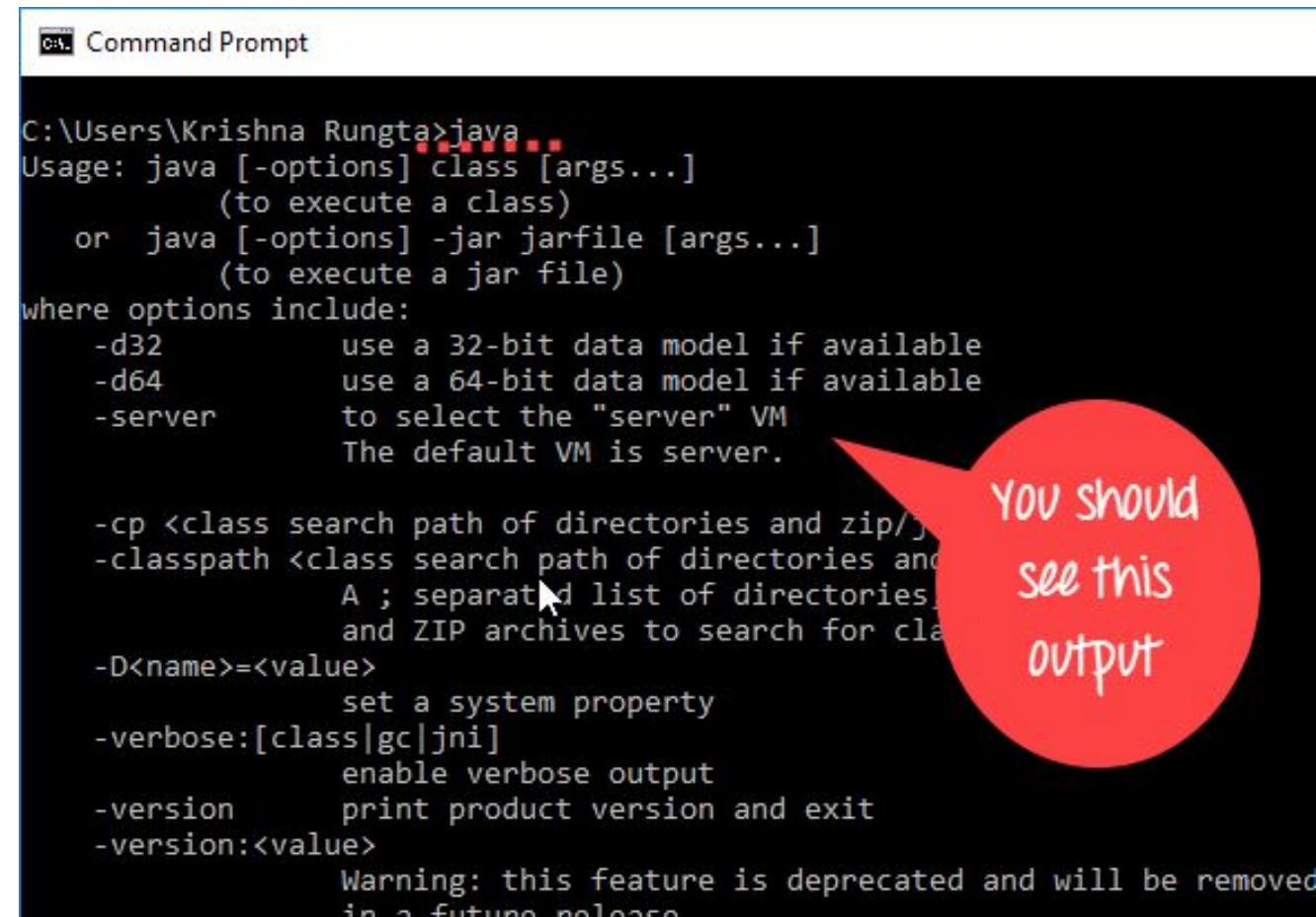
softserve

STEP 1 - JAVA

Download and install the **Java Software Development Kit (JDK)** [here](#).

This JDK version comes bundled with Java Runtime Environment (JRE), so you do not need to download and install the JRE separately.

Once installation is complete, open command prompt and type "java". If you see the following screen you are good to move to the next step

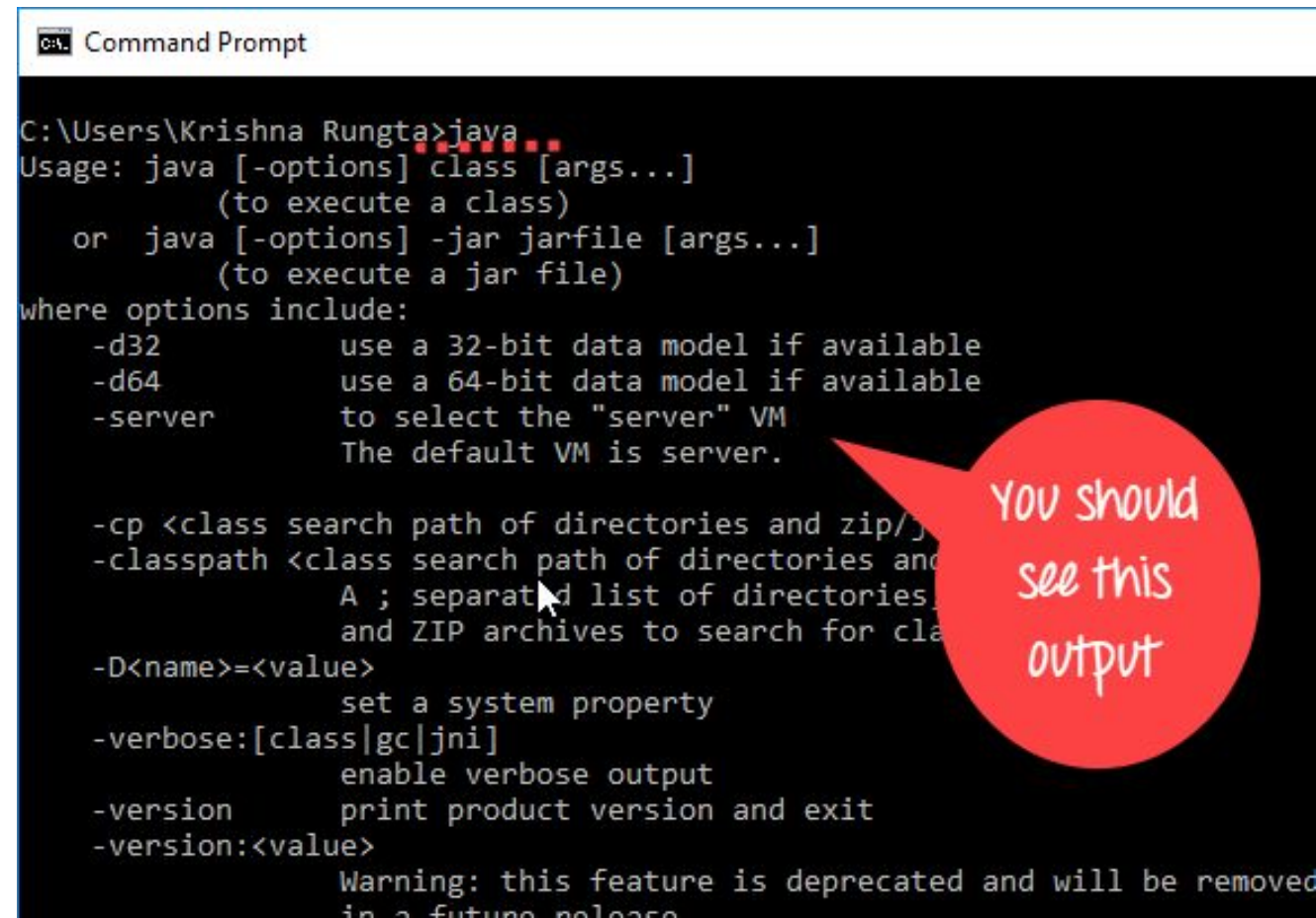


```
C:\Users\Krishna Rungta>java
Usage: java [-options] class [args...]
           (to execute a class)
 or  java [-options] -jar jarfile [args...]
           (to execute a jar file)
where options include:
    -d32          use a 32-bit data model if available
    -d64          use a 64-bit data model if available
    -server       to select the "server" VM
                  The default VM is server.

    -cp <class search path of directories and zip/jar files>
    -classpath <class search path of directories and zip/jar files>
                  A ; separated list of directories, ZIP archives, and
                  JAR archives to search for class files.
    -D<name>=<value>
                  set a system property
    -verbose:[class|gc|jni]
                  enable verbose output
    -version      print product version and exit
    -version:<value>
                  Warning: this feature is deprecated and will be removed
                  in a future release
```

STEP 2 - IDEA

Download and install the **Java Software Development Kit (JDK)** [here](#).



```
C:\Users\Krishna Rungta>java
Usage: java [-options] class [args...]
           (to execute a class)
   or  java [-options] -jar jarfile [args...]
           (to execute a jar file)
where options include:
    -d32          use a 32-bit data model if available
    -d64          use a 64-bit data model if available
    -server       to select the "server" VM
                  The default VM is server.

    -cp <class search path of directories and zip/jar files>
    -classpath <class search path of directories and zip/jar files>
                  A ; separated list of directories,
                  and ZIP archives to search for class files.
    -D<name>=<value>
                  set a system property
    -verbose:[class|gc|jni]
                  enable verbose output
    -version       print product version and exit
    -version:<value>
                  Warning: this feature is deprecated and will be removed
                  in a future release
```

TEST TYPE DEFINITION

Test Type it's a group of test activities aimed at testing a component or system focused on a specific test objective.

Test activities can be grouped by:

- Test Approaches
- Test Levels
- Test Objectives

PROACTIVE AND REACTIVE

REACTIVE

Reactive behavior is reacting to problems when they occur instead of doing something to prevent them

Testing is not started until design and coding are completed

PROACTIVE

Proactive behavior involves acting in advance of a future situation, rather than just reacting.

Test design process is initiated as early as possible in order to find and fix the defects before the build is created

VERIFICATION VS VALIDATION

Are we building
the product **right**?

To ensure that work products
meet their specified
requirements.

Are we building
the **right** product?

To ensure that the product
actually meets the user's needs,
and that the specifications were
correct in the first place.

POSITIVE AND NEGATIVE

In **positive** testing
our intention is

to prove that an application will
work on giving valid input data. i.e.
testing a system by giving its
corresponding valid inputs.

In **negative** testing
our intention is

to prove that an application will not
work on giving invalid inputs.

WHAT ABOUT BOXES?

Black-box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester.

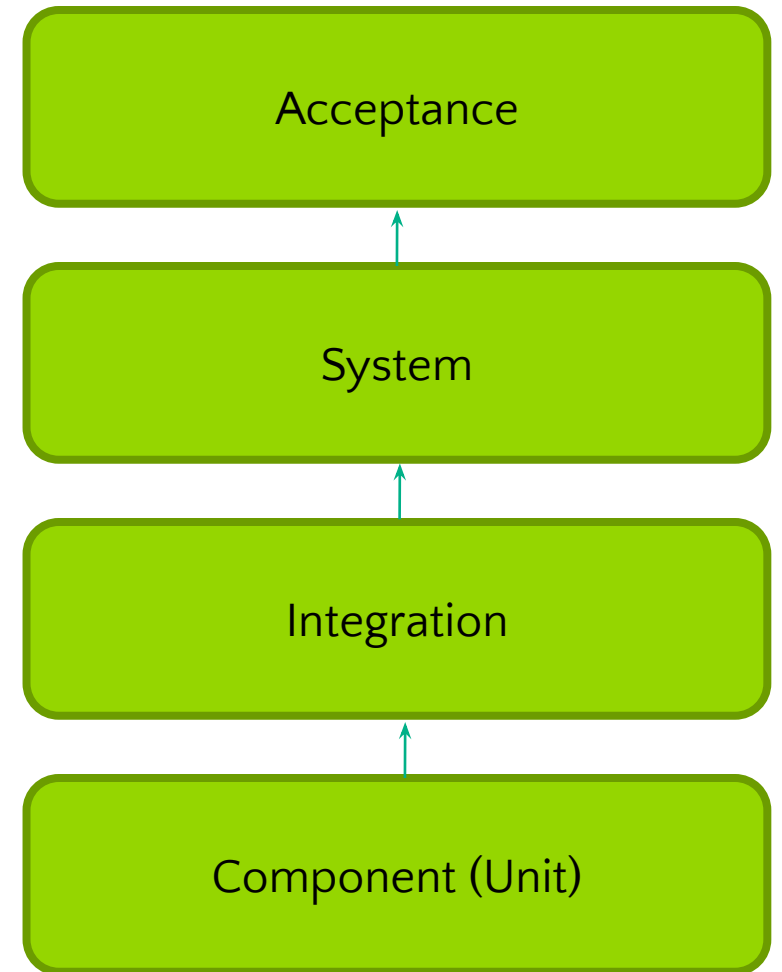
White-box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester.

Grey-box Testing is a software testing method which is a combination of Black-box and White-box Testing methods.

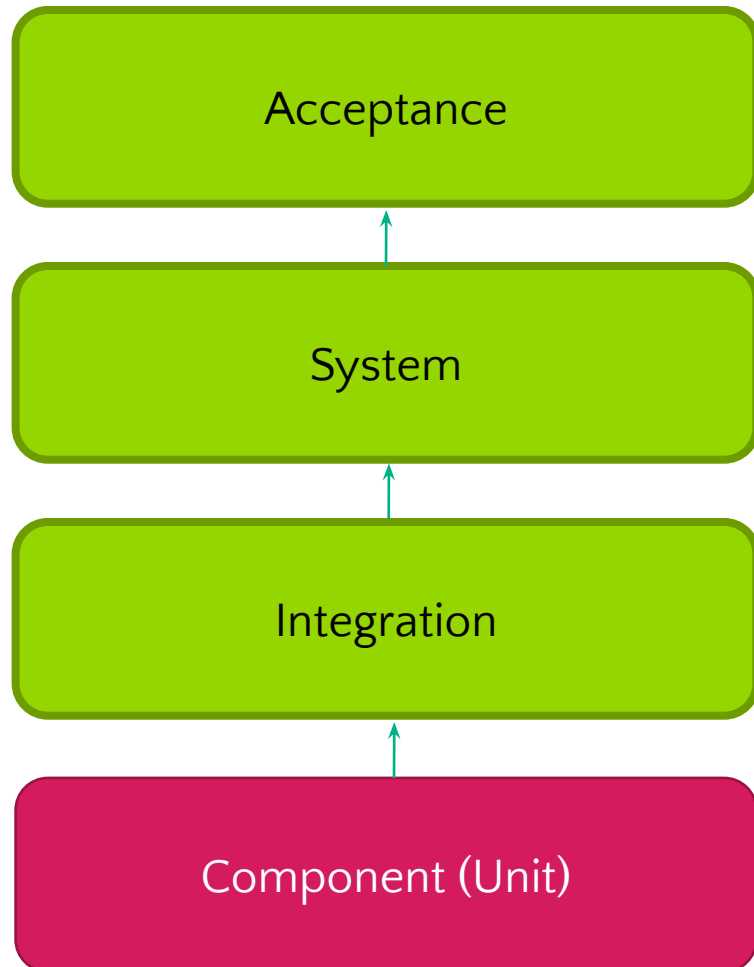
TEST LEVELS

Test levels are characterized by the following attributes:

- Specific objectives
- Test basis, referenced to derive test cases
- Test object (i.e., what is being tested)
- Typical defects and failures
- Specific approaches and responsibilities



COMPONENT LEVEL



Testing on the Component Test Level is called **Component (Unit, Module)** testing

Component (Unit) Test Level	
Who	DEV
When	Component is developed
Why	To validate that each unit of the software performs as designed
How	White-box testing

UNIT TESTING

Examples of a **test basis**:

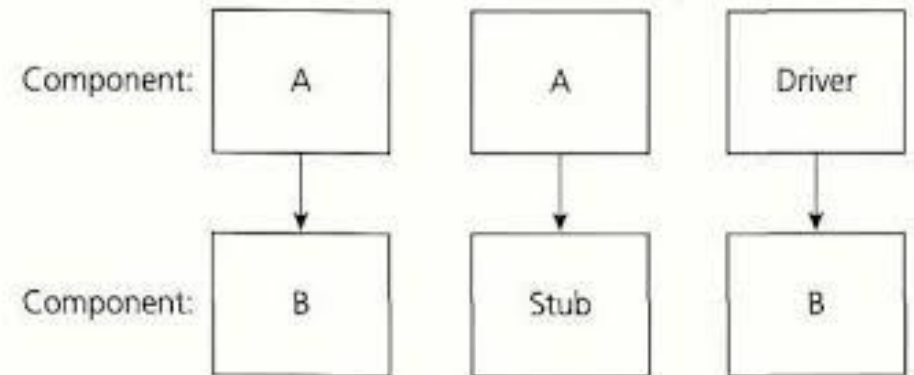
- Detailed design
- Code
- Data model
- Component specifications

Typical **test objects** for component testing include:

- Components, units or modules
- Code and data structures
- Classes
- Database modules

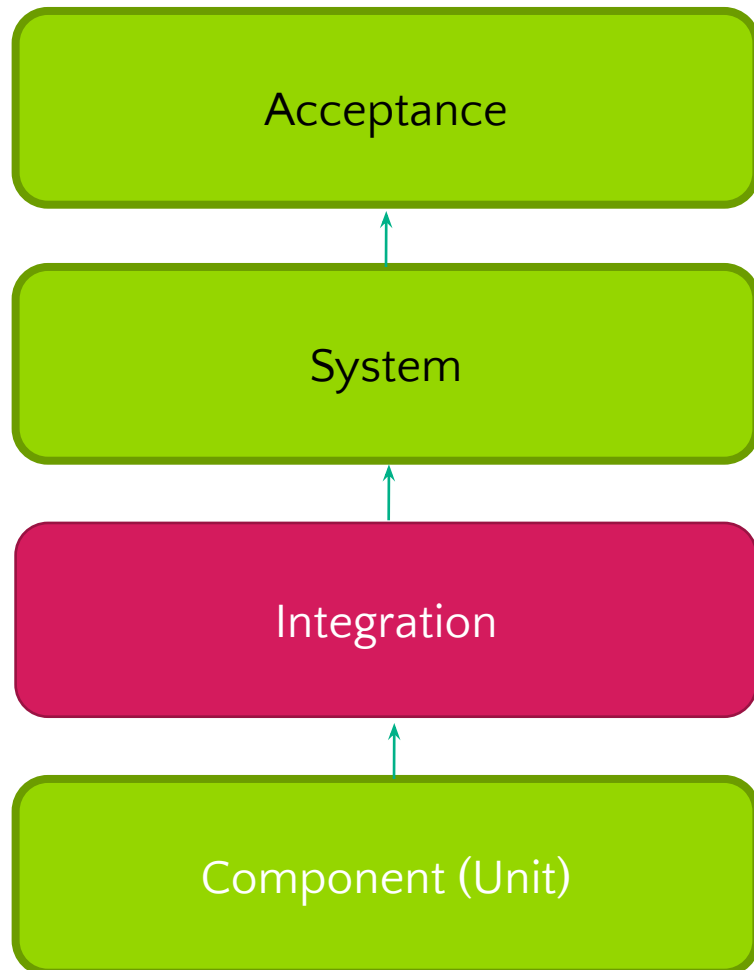
Typical defects and failures:

- Incorrect functionality (e.g., not as described in design specifications)
- Data flow problems
- Incorrect code and logic



softserve

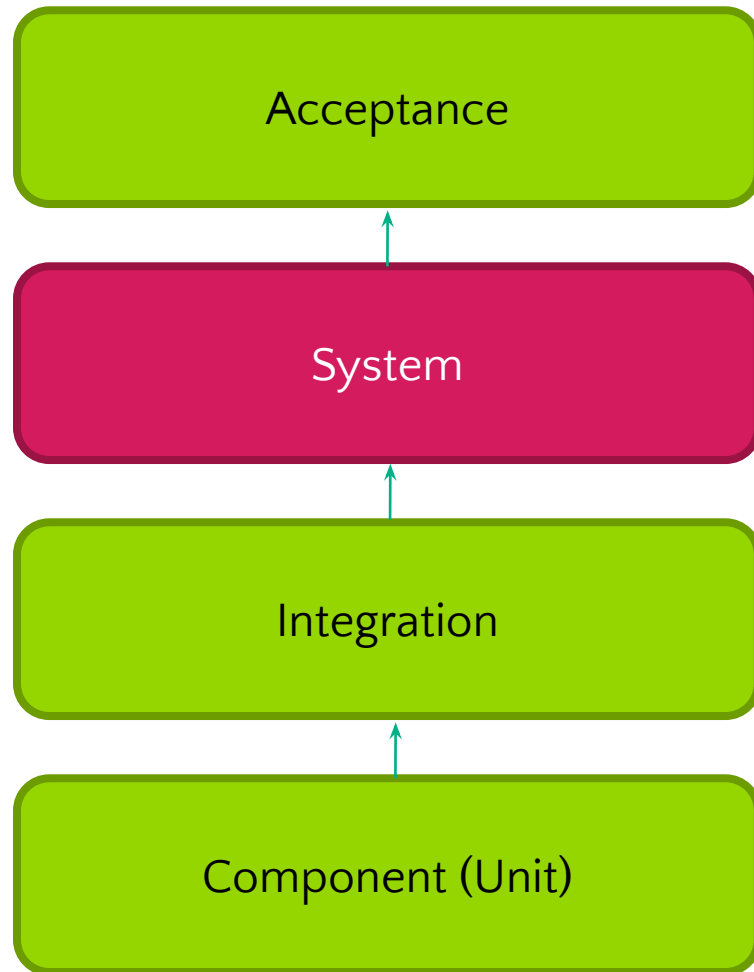
INTEGRATION LEVEL



Testing on the Integration Test Level is called **Integration** testing

Integration Test Level	
Who	DEV, QC
When	Units to be integrated are developed
Why	To expose faults in the interaction between integrated units
How	White-box/ Black-box/ Grey-box Depends on definite units

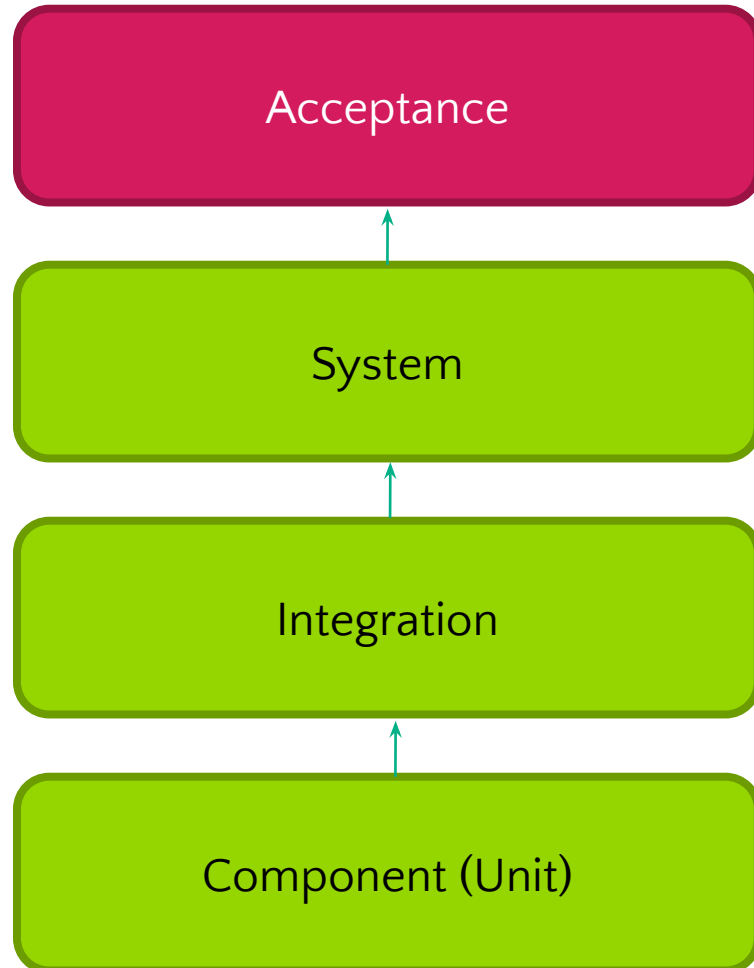
SYSTEM LEVEL



Testing on the System Test Level
is called **System** testing

System Test Level	
Who	QC
When	Separate units are integrated into System
Why	To evaluate the system's compliance with the specified requirements
How	Black-box testing

ACCEPTANCE LEVEL



Testing on the Acceptance Test Level is called
Acceptance testing

Acceptance Test Level	
Who	People who have not been involved into development
When	Component is developed
Why	To evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery
How	Black-box testing

TEST TYPES

Testing of function
(Functional testing)

Testing based on an analysis of the specification of the functionality of a component or system.

Testing of software product
characteristics
(Non-functional testing)

Testing of software
structure/architecture
(Structural testing)

Testing related to changes
(Confirmation and Regression
testing)

According to [ISO 25010](#) **Functional suitability** consists of:

- Functional completeness
- Functional correctness
- Functional appropriateness

softserve

TEST TYPES

Testing of function
(Functional testing)

Testing based on an analysis of the specification of the functionality of a component or system.

Testing of software product
characteristics
(Non-functional testing)

Testing of software
structure/architecture
(Structural testing)

Testing related to changes
(Confirmation and Regression
testing)

According to [ISO 25010](#) **Functional suitability** consists of:

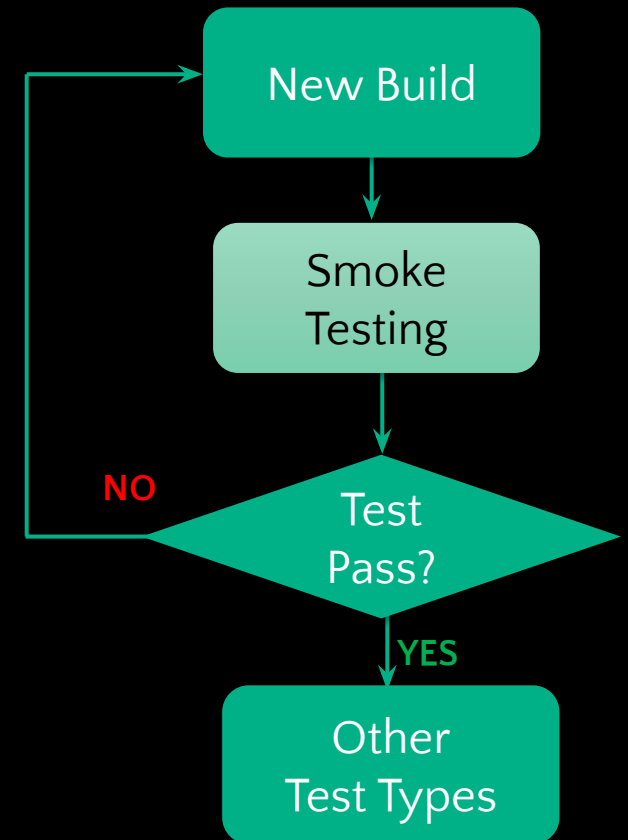
- Functional completeness
- Functional correctness
- Functional appropriateness

SMOKE TESTING

A subset of all defined/planned test cases that cover the main functionality of a component or system, to ascertaining that the most crucial functions of a program work, but not bothering with finer details.

Purposes:

- ✓ is done before accepting a build for further testing;
- ✓ is intended to reveal simple but critical failures to reject a software build\release;
- ✓ determines whether the application is so badly broken that further testing is unnecessary.



TEST TYPES: NON-FUNCTIONAL

Testing of function
(Functional testing)

Testing of software product
characteristics
(Non-functional testing)

Testing of software
structure/architecture
(Structural testing)

Testing related to changes
(Confirmation and Regression
testing)

Testing the attributes of a component or system that do not relate to functionality.

According to [ISO 25010](#) **Non-functional characteristics** are:

- Performance efficiency
- Compatibility
- Usability
- Reliability
- Security
- Maintainability
- Portability

softserve

TEST TYPES: NON-FUNCTIONAL

- **Performance efficiency:** Time behavior, Resource utilization, Capacity.
- **Compatibility:** Co-existence, Interoperability.
- **Usability:** Appropriateness recognizability, Learnability, Operability, User error protection, User interface aesthetics, Accessibility.
- **Reliability:** maturity (robustness), fault-tolerance, recoverability and availability.
- **Security:** Confidentiality, Integrity, Non-repudiation, Accountability, Authenticity.
- **Maintainability:** Modularity, Reusability, Analysability, Modifiability and Testability.
- **Portability:** Adaptability, Installability and Replaceability.

TEST TYPES: STRUCTURAL

Testing of function
(Functional testing)

Testing of software product
characteristics
(Non-functional testing)

Testing of software
structure/architecture
(Structural testing)

Testing related to changes
(Confirmation and Regression
testing)

Mostly applied at Component
and Integration Test Levels

TEST TYPES: REGRESSION

Testing of function
(Functional testing)

Testing of software product
characteristics
(Non-functional testing)

Testing of software
structure/architecture
(Structural testing)

Testing related to changes
(Confirmation and
Regression testing)

If we have made a change to the software, we will have changed the way it functions, the way it performs (or both) and its structure.

SUMMARY

Test activities can be grouped using different classification:

- By the degree of automation (Manual and Automated);
- By the level of awareness about the system and its internal structure (Black-, White-, Grey-box);
- By the basis of positive scenario (Positive and Negative);
- By the degree of preparedness to be tested (Scripted and Unscripted);
- By the degree of component isolation (by Test levels);
- By the Test Objectives.

TEST DESIGN

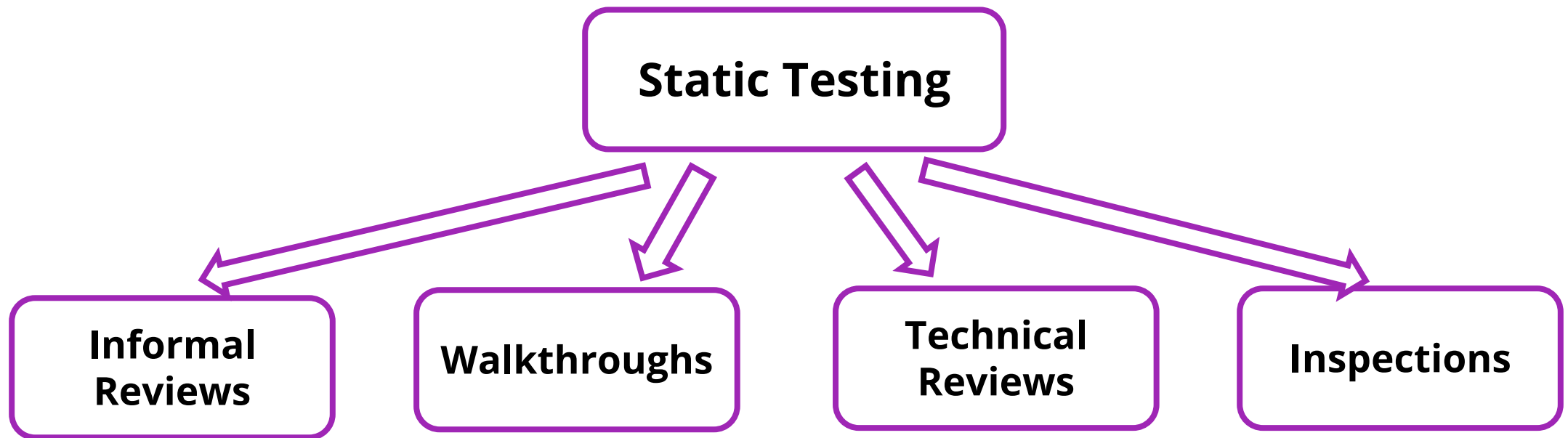
softserve

Categories

```
graph TD; A[Categories] --> B[Static: Static testing test software without executing it]; A --> C[Dynamic: Testing that involves the execution of the software of a component or system];
```

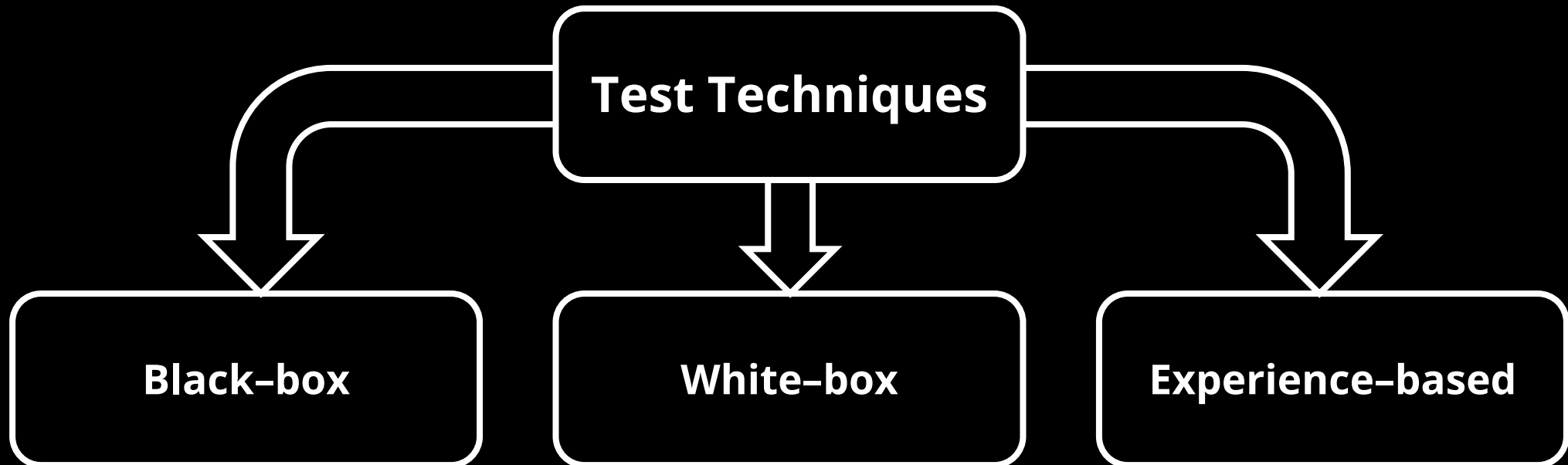
Static: Static testing test software without executing it

Dynamic: Testing that involves the execution of the software of a component or system



TEST TECHNIQUES

The purpose of a test technique, including those discussed in this section, is to help in identifying test conditions, test cases, and test data.



BLACK BOX

Equivalence
Partitioning

Boundary
Values
Analysis

State
Transition

Decision
Tables

Use Case
Testing

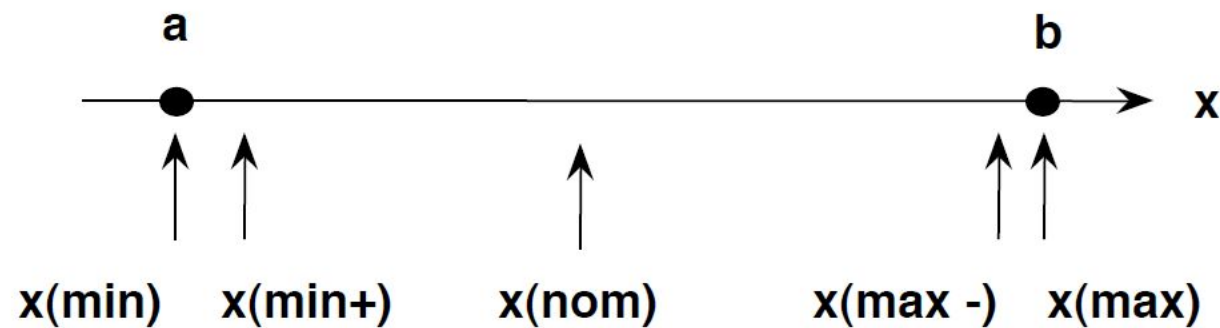
EQUIVALENCE PARTITIONING

- **Equivalence partitioning (EP)** – A black-box test design technique in which test cases are designed to execute representatives from equivalence partitions.
- **Idea**: Divide (i.e. to partition) a set of test conditions into groups or sets that can be considered the same (i.e. the system should handle them equivalently), hence equivalence partitioning. In principle test cases are designed to cover each partition at least once.
- **Example**: Bank represents new deposit program for corporate clients. According to the program client has ability to get different %, based on amount of deposited money. Minimum which can be deposited is \$1, maximum is – \$999. If client deposits less than \$500 it will have 5% of interests. In case the amount of deposited money is \$500 and higher, then client gets on 10% of interests more.

- **Boundary value analysis (BVA)**: A black box test design technique in which test cases are designed based on boundary values.

BVA is an extension of equivalence partitioning, but can only be used when the partition is ordered, consisting of numeric or sequential data. The minimum and maximum values (or first and last values) of a partition are its boundary values

- **Idea**: Divide test conditions into sets and test the boundaries between these sets. Tests should be written to cover each boundary value.



- **Decision table** – A table showing combinations of inputs and/or stimuli (causes) with their associated outputs and/or actions (effects), which can be used to design test cases.
- **Idea:** Divide test conditions into constraints, which could get positive or negative meanings, and rules which identify output based on values of conditions. While analyzing each possible variant of positive and negative meanings identify output or set of outputs for each variant based on the rules. Only combinations of these positive and negative meanings, which uniquely identify decisions that are made, should be covered by tests.

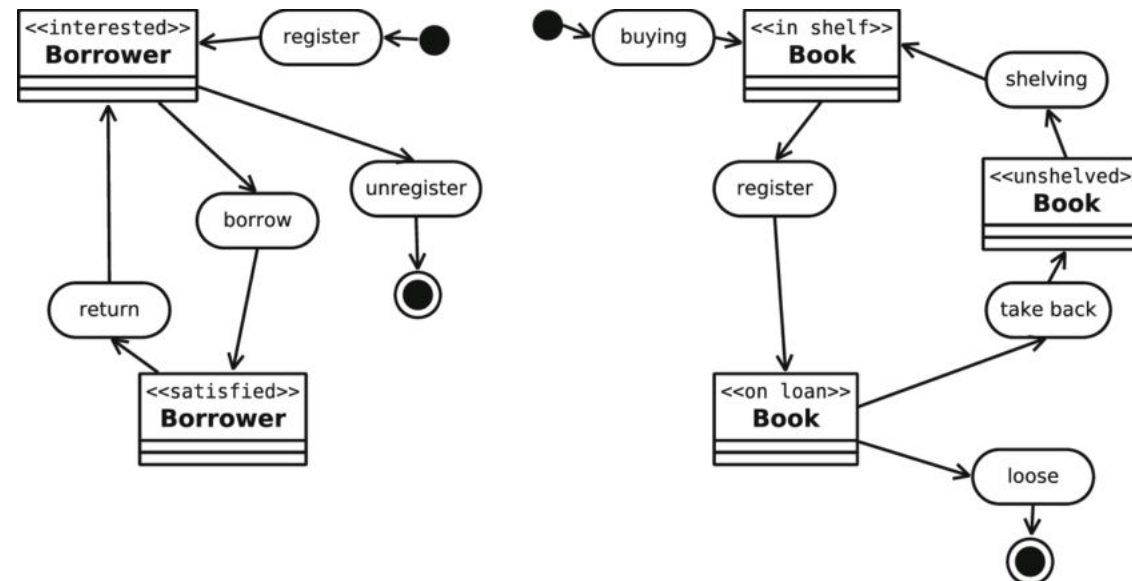
<u>Causes (inputs)</u>	R1	R2	R3	R4	R5	R6	R7	R8
Over 60s rail card?	Y	Y	Y	Y	N	N	N	N
Family rail card?	Y	Y	N	N	Y	Y	N	N
Child also traveling?	Y	N	Y	N	Y	N	Y	N
<u>Effects (Outputs)</u>								
Discount (%)	50	34	34	34	50	0	10	0
Message*	+	+						

- 'over 60s' rail card – 34%
- family rail card and traveling with a child – 50%
- traveling with a child, but do not have family rail card – 10%
- only one type of rail card can be used

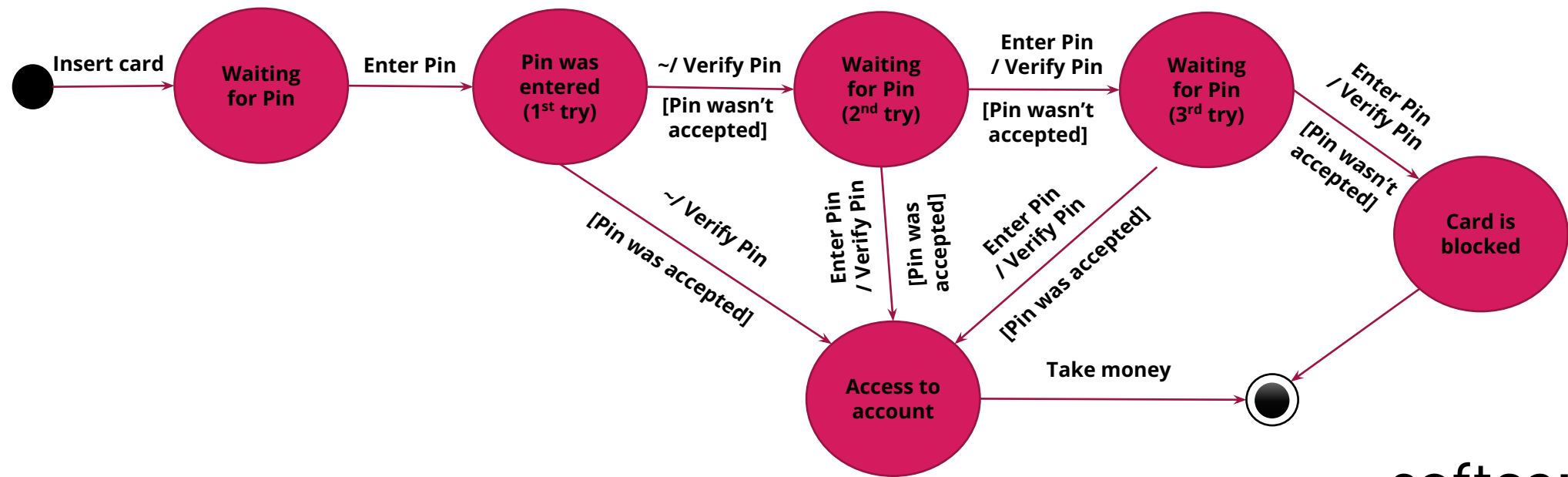
The rationalized table with a fewer columns and thus will result in fewer test cases:

softserve

- **State transition testing** – A black box test design technique in which test cases are designed to execute valid and invalid state transitions.
State transition – A transition between two states of a component or system.
- **Idea:** Design diagram that shows the events that cause a change from one state to another. Tests should cover each path starting from the longest state combination.



- **Example:** Client of the bank would like to take money from bank account using cash machine. To get money he should enter valid Personal Identity Number (PIN). In case of 3 invalid tries, cash machine eats the card.



WHITE BOX

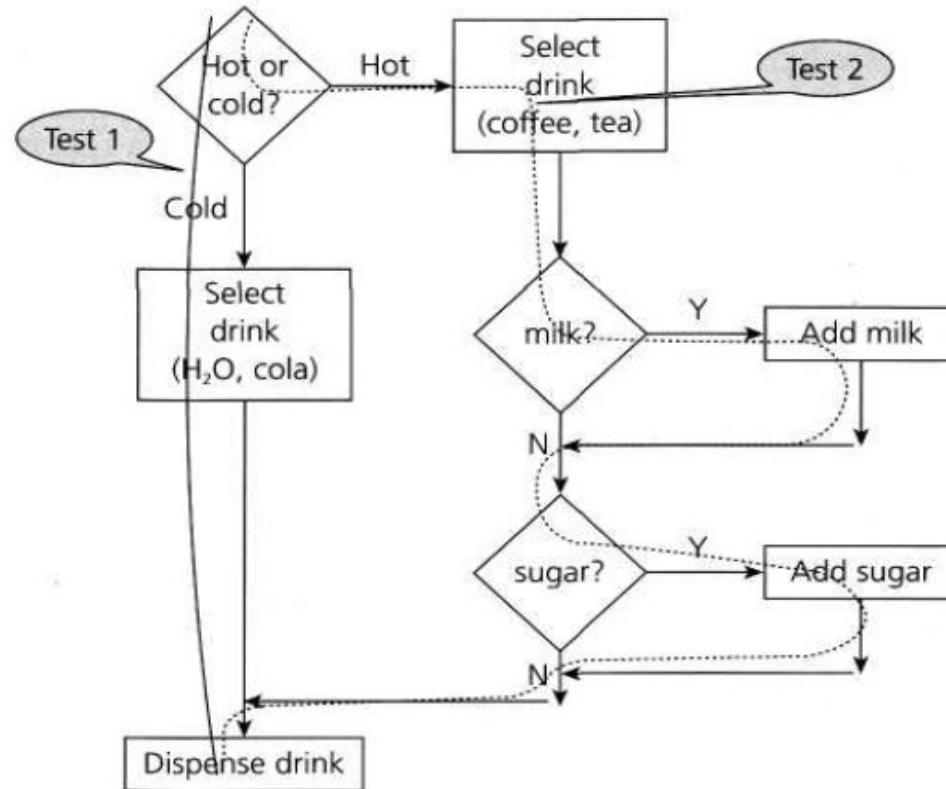
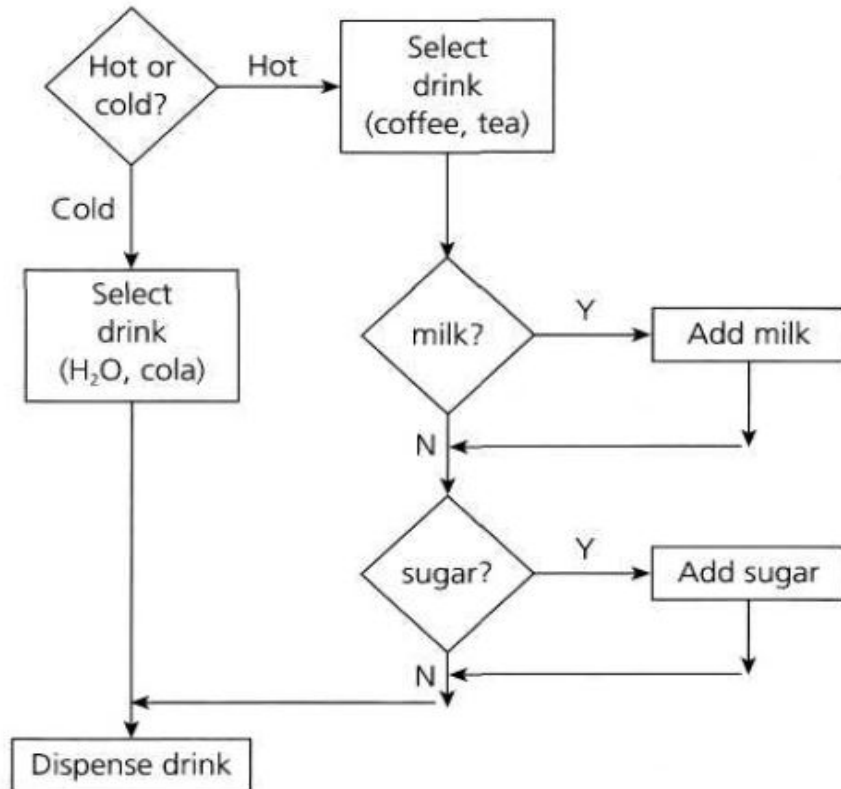
Statement

Decision

softserve

Statement Testing and Coverage*

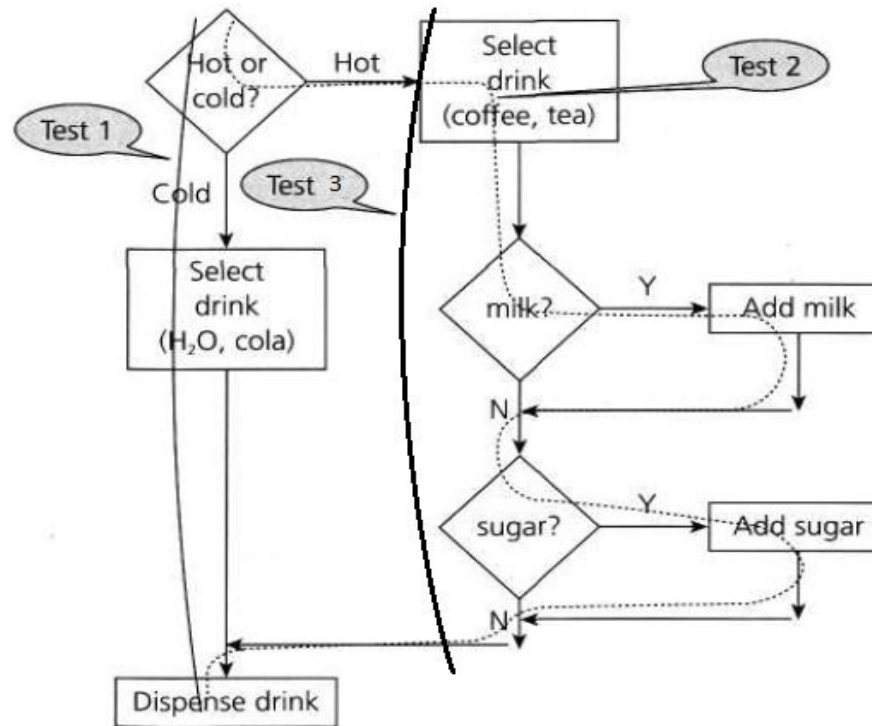
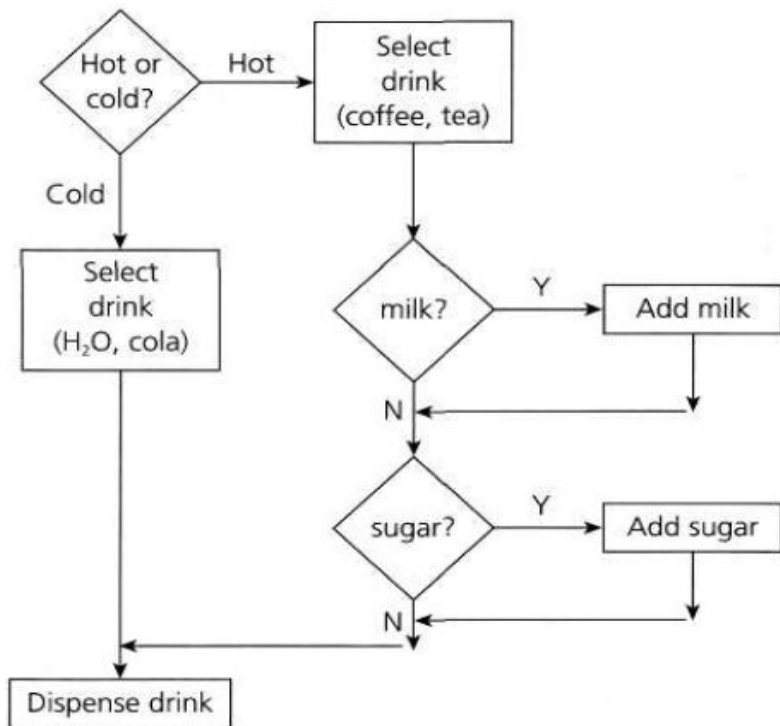
- **Statement** – an entity in a programming language, which is typically the smallest indivisible unit of execution.
- **Example:**



softserve

Decision Testing and Coverage*

- **Decision** is an IF statement, a loop control statement (e.g. DO-WHILE or REPEAT-UNTIL), or a CASE statement, where there are two or more possible exits or outcomes from the statement.
- **Example:**



EXPERIENCE BASED

Error Guessing

Exploratory
Testing

Checklist-base
d Testing

WHO'S NEXT?

softserve

YOU'RE NEXT

***#*STUDYHARD**

softserve

Q&A

softserve