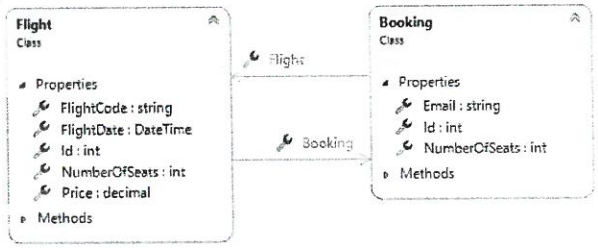
Teammate1:	Juan Picazo Martínez	Teammate2:	Javier García Terceño

Given the following code for the method *CreateRecommendedBooking* in the Booking controller related to the previous Use Case, apply the **Path Coverage technique** to create its unit tests.

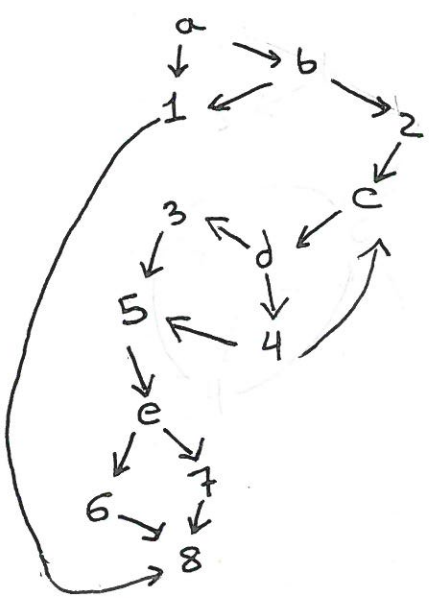
- The tables of the database **Flights** and **Bookings** are also considered as **input/output** of the test cases, depending on how they are used in the code.
- That indicated in the **return** instruction is considered as **outputs** for the test cases.

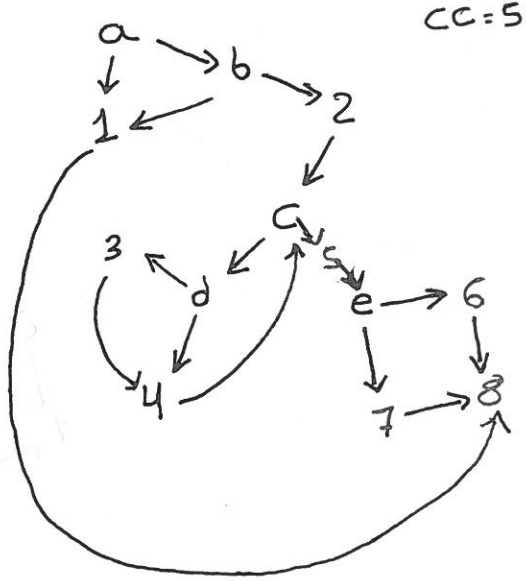
```
[HttpGet]
[ProducesResponseType(typeof(IList<Flight>), (int)HttpStatusCode.OK)]
[ProducesResponseType(typeof(string), (int)HttpStatusCode.BadRequest)]
public IActionResult GetFlightsAvailable(DateTime dateForBooking, int NumberOfSeats)
{
    if ((dateForBooking <= DateTime.Today) || (NumberOfSeats <= 0))
        return BadRequest("Flight date or the number of passengers are not correct");

    //it obtains all the flights that will fly on dateForBooking &&
    //have not booked yet (f.Booking == null)
    IList<Flight> flightsAvailable = _context.Flights
        .Where(f => f.FlightDate.Date.Equals(dateForBooking.Date) && f.Booking == null)
        .OrderBy(f => f.NumberOfSeats).ToList();

    int i = 0;
    IList<Flight> flights2return = new List<Flight>();

    while (i < flightsAvailable.Count)
    {
        if (flightsAvailable[i].NumberOfSeats >= NumberOfSeats)
            flights2return.Add(flightsAvailable[i]);
        i++;
    }

    if (flights2return.Count == 0)
        return BadRequest("There are no flights available for dateForBooking with such NumberOfSeats");

    else
        return Ok(flights2return);
}
```



Example of instances:
Flight1={XCT9675, 15/06/2026, 1, 20, 300.00€, null}
Flight2={YBG9076, 15/06/2026, 2, 50, 450.00€, **Booking1**}
Booking1={Marta.Martin@gmail.com, 1, 50, **Flight2**}

Example of tables:
Flights = {Flight1, Flight2}
Bookings = {Booking1}



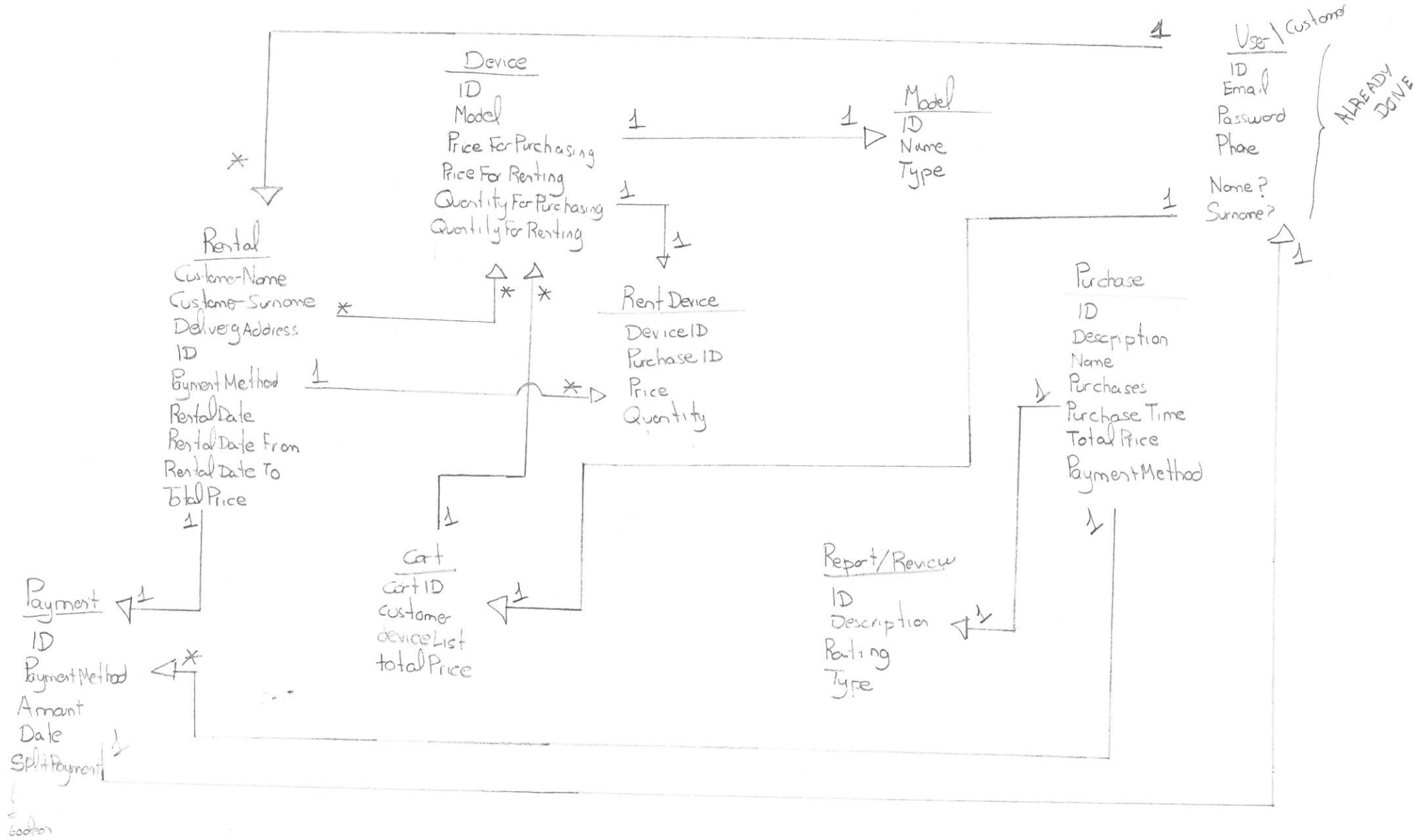


CC = 5+1 = 6

Nodes
a 1 7
a b 1 7
a b 2 c e 6 7
a b 2 c d 4 c e 6 7
a b 2 c d 3 4 c e 6 7

Path FlightCode flightdate NumberofSeats Return

Date time | Number of seats |


Device
ID
Model
Price For Purchasing
Price For Renting
Quantity For Purchasing
Quantity For Renting
Model
ID
Name
Type
User / Customer
ID
Email
Password
Phone
Name?
Surname?
ALREADY DONE
Rental
Customer Name
Customer Surname
Delivery Address
ID
Payment Method
Rental Date
Rental Date From
Rental Date To
Total Price
Rent Device
Device ID
Purchase ID
Price
Quantity
Purchase
ID
Description
Name
Purchases
Purchase Time
Total Price
Payment Method
Cart
Cart ID
Customer
deviceList
totalPrice
Report / Review
ID
Description
Rating
Type
Payment
ID
Payment Method
Amount
Date
Split Payment
Version Control
Change control
Audits
Status
Accounting

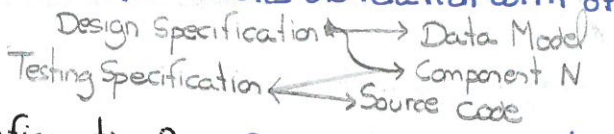
Unit 1. Software Configuration Management

In Configuration Management, the functional and physical characteristics of hardware and software as set of faith in technical documentation or archieved in a product

- Basis of SW products (multiple versions), SW projects (everything changes) and SW develop teams
- Necessary bcs high complexity and demand
- Lehman and Belady Law: "We realize what we need in software while using it"
- Help us to manage a system well (know and understood the system)

Basic concepts

→ Configuration Item (CI): Single entity (HW or SW) in the CM process.
Has name, attributes and relation with other CIs



→ Configuration Base: Basis of the configuration. Cannot be changed, only by formal change control procedures
The aim is to reduce uncontrolled changes by fixing and controlling. Established at the end of a stage life bcs refers to the act of placing an approved item.

→ Project Repository: Record all relevant information related to the configration (CIs, change requests, accounting and auditing)

Commit: Upload changes to repo
Check-Out/Pull:

→ Version: SW product that differs from the other in terms of capability, config... Any change to a CI produces a new version

→ Release: Delivered version to customers which includes part of an application

CM Process

a) Identification: 3 tasks
- Selection Not every CI must be under CM. Too many ⇒ Cost of time and development.
Too few ⇒ Difficulty in controlling changes
CI could be atomic or composed
- Labelling Scheme: To identify each CI uniquely
- Description: Document the characteristic of each CI

b) Version Control: Manage the versions created during the SW develop process. Possible to return to previous
Centralized or distributed
Staging area
Synchronization control: Resolve conflicts or locks
Access control: Able to identify who can do that. Security
Evolution graph: Each node is a version and each branch is a deviation from the main develop. file

c) Change Control: Identifying, documenting, approving or rejecting changes. A change Request (CR) is submitted by a developer and must be reported.
- Change Control Process: Actions taken to identify, document, review and authorize changes
- Change Control Board: Group of stakeholders for reviewing, evaluating, approving, delaying or rejecting changes
- Levels:
1. Informal: Stakeholder can do whatever change till the CI is baselined
2. Project: We have a baseline. To carry out a change it must be approved
3. Formal: We have a release. All the process must be carried out

d) Status Accounting: Report and record info to effectively manage a SW system. Dashboard

e) Auditing: Independent examination of a work product to assess compliance
- Functional audits: If the product complies with the goal requirement of the baseline
- Physical audits: If the baseline can be deployed.

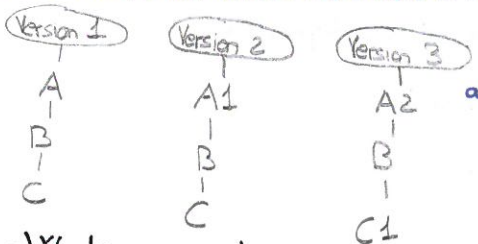
Version Control: GIT

(Seminar)

To save work and coordinate code with team

1 server with all files with versions

GIT thinks of data more like a set of snapshots of a miniature filesystem

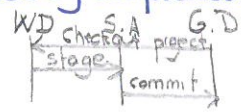


Every time you commit, it takes a picture of what your files look like and stores a reference to that snapshot

If a file is not saved, it is not restored, just links the previous file

Sections of a Git project:

- Git directory: Stores metadata and object DB of project.
- Working Directory: Single checkout of one version of the project. Took from Git directory and placed on disk for you to use or modify
- Staging area: File in Git directory with info of what is going into your next commit



Files in Working Directory can be in two states:

- Tracked file: Files that were in the last snapshot. Can be unmodified, modified or staged.

When first cloning a repo, all files are tracked and unmodified

- Untracked file: Everything else, files that were not in our last snapshot or in the staging area

- Modified: Changed file but not commited into DB.
- Staged: Marked a modified file to go into next commit snapshot
- Committed: Data is safely stored in local DB
- Pushed: Data is safely stored in remote DB.
- Fetch: Download changes from remote
- Merge: Apply changes from fetch into local repo
- Pull: Fetch + merge
- Revert:

Branchs

→ Merge commit

Unit 2. Software Testing: Techniques & Strategies

Software testing: Dynamic verification of the behavior of a program on a finite set of test cases, against the unexpected behaviour. Part of verification and validation, where checks it is complete and correct, fulfil the requirements and conditions. Planned by an independent team

Basic concepts

→ Fault (bug): Problem that could cause an application to fail or produce incorrect results
→ Failure: Event in which a system or system component doesn't perform a required function
→ Error: Human action that produces an incorrect result. Faults and failures are errors
→ Test case: Set of test inputs, exec. conditions, and expected results made for a particular objective With a path to be followed. Must have: specification, goal, documentation and execution
→ Test approach: Method to pick the particular test case values. From very general to very specific.
→ Test design: Documentation specifying details of the test approach
→ Test bed: Enviroment containing the HW and SW to support the TCs

Testing techniques

Main aim is to detect as many failures as possible trying to break the program, seeing which inputs will produce representative program behaviors. There are two types of test approaches:

a) White-box: With info about how SW has been coded. Tester needs the knowledge about the inner structure of the SUT. Impossible to test all the paths when too many. A program is adequately tested with respect of a criterion if all target structural elements have been exercised:
- Statement adequacy criterion: All statesments executed at least onee
- Decision adequacy criterion: All outcomes possible, at least once
- Condition adequacy criterion: All values in a compound predicate, at least once

Coverage analysis refers to the extent the TC satisfies the test adequacy criterion. A path is a sequence of control flow nodes, beginning from the entry node through to the exit node. Path coverage:
1. Create a flow graph (Sentence nodes, Edges and Condition Nodes)
2. Compute cyclomatic complexity: V(G) = P+1, being P the number of conditions nodes
3. As many independents paths as V(G). From the simplest
4. Prepare the test cases, one column per input parameter and one column per output parameter. As many rows as TCs. (Lo de los ejercicios de clase)

- Data Flow Testing: Variable is defined when its value is assigned or changed. Variable is used when the value is used without modifing it. Def-use path is a path variable from definition to use. P-use = used in predicated C-use = used in computation

- Loop testing: In a compound predicate, we need to see what would happen if we change order conditions. Simple, from 0 to N, nested, concatenated or unstructured (redesign)

b) Black-box Only rely on the input/output of the SW
- Equivalence Classes: Valid and invalid outputs (partitioning). One valid and one invalid
  1. Range of values: One valid, two invalid. If [1-5] → 2 and 0-6
  2. Finite number of values: " " " ". One per end. 4-7 chors → Javi and Ana-Cristobal
  3. Set of valid input values: One for the set and for any outside
  4. Must be: One valid that includes the condition and one invalid that doesn't
  5. Smaller classes: Divisions in other classes
- Boundary Classes: Same as before, but we change:
  1. Range of values: Two valid (one per end) and two invalids (one per end). If [1-4] → 1-4 and 0-5
  2. Finite number of values: " " (" " ") " " " (" " "). 4-7 chors → Javi-Manuel and Ana-Cristobal
  3. Input ordered as a set, we focus and the last and the first
- Error guessing: Zero, null and no values are prone to cause failures.

↑ ↑ ↑
Ejer. práctico hecho en clase (Path coverage and Equivalence class)

Unit 3. Software Testing: Process and activities

Software testing is no longer a post-coding phase, is a prevention phase that is dynamic. Related to quality process. Testing can be seen as a means for providing info about the functionality and quality attributes. Should be pervasive thrugh out the entire development and maintenance life cycle

Test levels

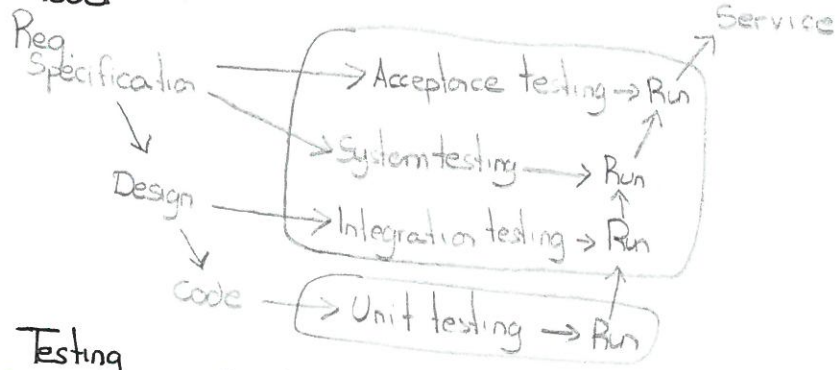
Main goal: Maximize use of time and resarces, need of test cases to detect defects. Test reuse. Deliver a higher-quality software product

Target: Object of testing. Single module, grap of modules or an entire system.

Three stages: Unit, integration and system

Objective: The purpose, the aim to perform in the target

V-Model



Unit Testing

Testing of individual routines and modules by the developer or independent tester

Unit: Smallest testable software component

Constraint: Unit with a high level of cohesion and low level of coupling. Easier to design, run and analyse

Automation: XUnit, NUnit, Simple Test...

In Object-Oriented (OO), we shald focus on:
- Getter-Setters
- Methods
- States

Advantages:
- OO promotes encapsulation ⇒ Reduce re-testing when class changes
- No necessay to retest inherited subclasses.

Unit Testing (Seminar)

Test a part of the app in isolation from its infraestructure. Only one action per logic

Generally, unit tests are simple to write and quick to run

They don't detect issues in the interaction between components.

In VS, each unit test is a method in a C# class

```
public GetDevicesForRental_test() → Name describes what test does
    "pattern (A/A/A)"
        Arrange = Setting up the conditions for test
        Act = Call the SUT
        Assert = Verify that the result was the expected. ALWAYS must be an assert in UT.
{
```

→ Test for double null
→ Test for string-null
→ Test for null-string
→ Test for double string

{ Equal();

Data-Driven test:

XUnit uses [Fact] attribute for parameterless unit test

[Theory] for parametrised test that is true for a subset of data, provided by [MemberData]

Unit 4. DevOps

Responsibility for adapting to market changes by deploying features and updates into production quickly and efficiently is made by the Development Teams

Operation teams ensure IT services are stable, reliable and secure, often creating strict safeguards to prevent changes that could risk stability

"The core, chronic conflict"

Development and IT operations have diametrically opposed goals and incentives

Speed and innovation ↵ ↳ Stability and risk management

DevOps = Set of software development tools, processes, and practices mixing software development (Dev) with information technology operations (Ops) to facilitate software development lifecycle

Needs an automated delivery cycle that includes planning, development, testing, release, deployment and monitoring

Continuous Integration (CI): Process of frequently integrating new code into the main branch with automating testing to validate.

Advantages: Early detection and resolution of issues // Faster SW delivery with fewer bugs

Requirements: 1. Centralized version Control 2. Automated build processes for reproducibility and debugging 3. Team commitment to integrate small and incremental changes regularly

Continuous Delivery (CD): Builds on CI by validating changes as release candidates through automated deployment and testing processes

Implement a deployment pipeline that makes build, test and release stages visible to everyone involved

Improve feedback

Automation of any SW version to any environment

Continuous Deployment: Extends CD by releasing every change that passes through the production pipeline to customers, without human touch unless test fails

Benefits: Accelerates feedback, reduces preassure to team bcs no "release date"

Unit 5. SW Testing: Process and Documentation

Testing is a key approach to risk mitigation in SW development. Risk-based testing, focus on the most important features and quality attributes. Test documentation is an output of the processes

Test processes: Provide information on SW quality comprising a number of activities grouped into sub-processes

Test management processes: At project level, for test management at different test phases and for managing various test types. Test Planning Process Develop → Test Plan (Project Document)
Depending where implemented could be Project Test Plan or Specific phase or Specific type of testing
Importance of Test Monitoring and Control process to check concordance to the Test Plan
Test Completion process when agreement has been obtained (Project or Specific case)

Dynamic Test Processes, to carry out dynamic testing in a poticular phase (unit, system...) or type of testing (performance, security...)
Implementation process requires testers to apply one or more test design techniques to derive test cases with a whole coverage

Change and configuration management processes exist, so test environment set-up and maintenance is required. Initial ideas would be described in the Test Plan, but will be clear when this process starts
The execution process is when all test are running, may be rumed several times due problems for single iteration
The incident Reporting test is run at the end, for unusual behaviours. New test may be needed (retest).

Integration testing, progressive linking and testing of programs or modules in order to secure the proper functioning of the whole system, checking SW and HW interaction
Goal: Multiple complete, integrated systems to evaluate their ability to communicate successfully with each other and fullfill system's specified requirements. TWO Approaches:

→ Incremental testing: Product must be hierarchically designed. By bottom-up or top-down + white/black box testing
Advantages: Faults easily located Disadvantages: Is neccesary driver and stub code
Automation: XUnit, JUnit...

→ Big-Bang testing: SW+HW elements combined all at once into an overall system, no stages
System not tested until all finished as a whole
Advantages: Driver and stub code not necessay Disadvantages: Faults not isolated

Top-Down vs Bottom-up testing

Bottom-up

- Advantages
  - Lower levels well-tested
  - Driver code is simpler
- Disadvantages
  - Requires driver code
  - Upper levels late tested, no so well
  - System as a whole at the end

Top-Down

- Advantages
  - Upper-levels early test (redesign)
- Disadvantages
  - Complex stub code

Which alternative? → Check risk factors: mission/safety/business

→ Object-Oriented systems: Hierarchy is not applicable. Cluster: classes that are related
Testing technique: First, select classes that request few services from other classes, then, classes that integrates

System testing. Performance Testing: To evaluate the compliance of a system or component with performance requirements. Signed by client and quantified → Response time, X per second...

System testing. Load and Stress Testing
Load: Evaluate system at the limits of its specified requirements
Stress: Determine upper limits, continues until sth breaks

Reveal defects
Checks per design
Build customers trust

System testing: Functional Testing. Evaluate compliance of a system with black box during requir. age
AUP- Guided by Use Cases
Test Cases from Use Cases?
1. Create Scenarios
   Sce-1 BF+AF1
2. Generate test cases (1 per sce)
3. Identify Data Values for TC

ID	Sce	Data	Expected result
TC3-1	SCE1...	"1", "house"	...
TC3-2	SCE1-...	"2", "house"	...

Tools: Eclipse, test projects for .NET, Selenium, Jmeter

Acceptance Testing: Determine if satisfies acceptance criteria for a specific customer after system testing
- Alpha testing: Users invited to developers site
- Beta testing: Sent to users and used under real world

Regression Testing: Change to a component affects functionality or has additional defects?

Unit 6. Software Quality

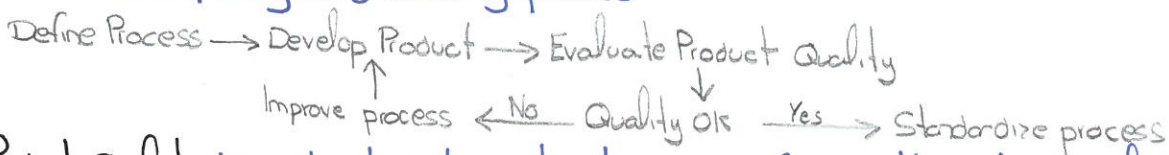
Two different views:

▷ Product Quality
- Fitness for use, count for it for what it was needed
- For customer, everything that increases its satisfaction
- Has the set of properties and characteristics requested

▷ Process Quality
- From "Factory" point of view
- Establish a quality management system in a company will produce a quality product

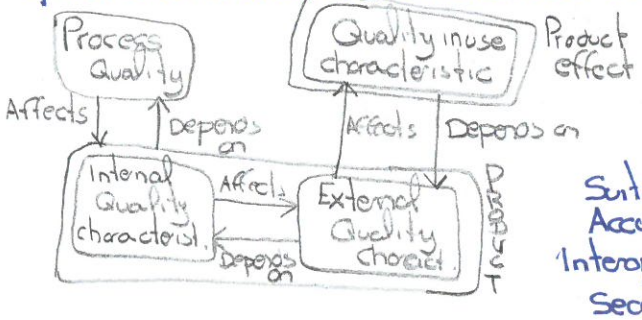
→ Product VS Process Quality

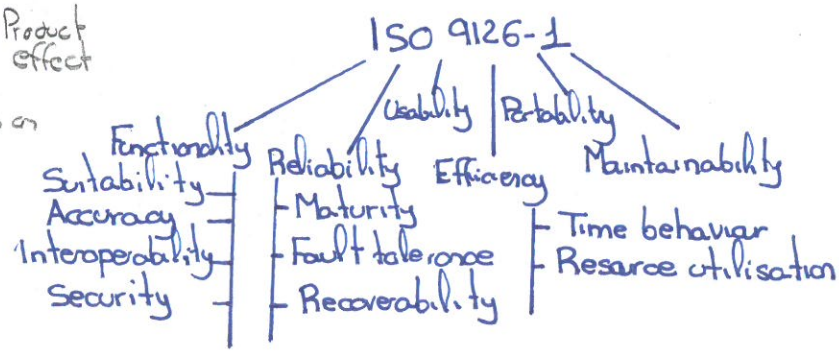
Product quality is influenced by process



Product Quality is a subjective characteristic, so is defined as the contractural requirements by SW process and product

Requirements in internal, external and Use that are verified, determine the quality





ISO 9126-2
External metrics

Measure by the system behavior
Only in testing stage
When the system environment must work

ISO 9126-3
Internal metrics

Can be applied to a non-runnable SW product
Initial stages
During the life cycle development

Unit 1, Configuration management

Configuration: Requirement, design and implementation of a particular version of a system. HW + SW

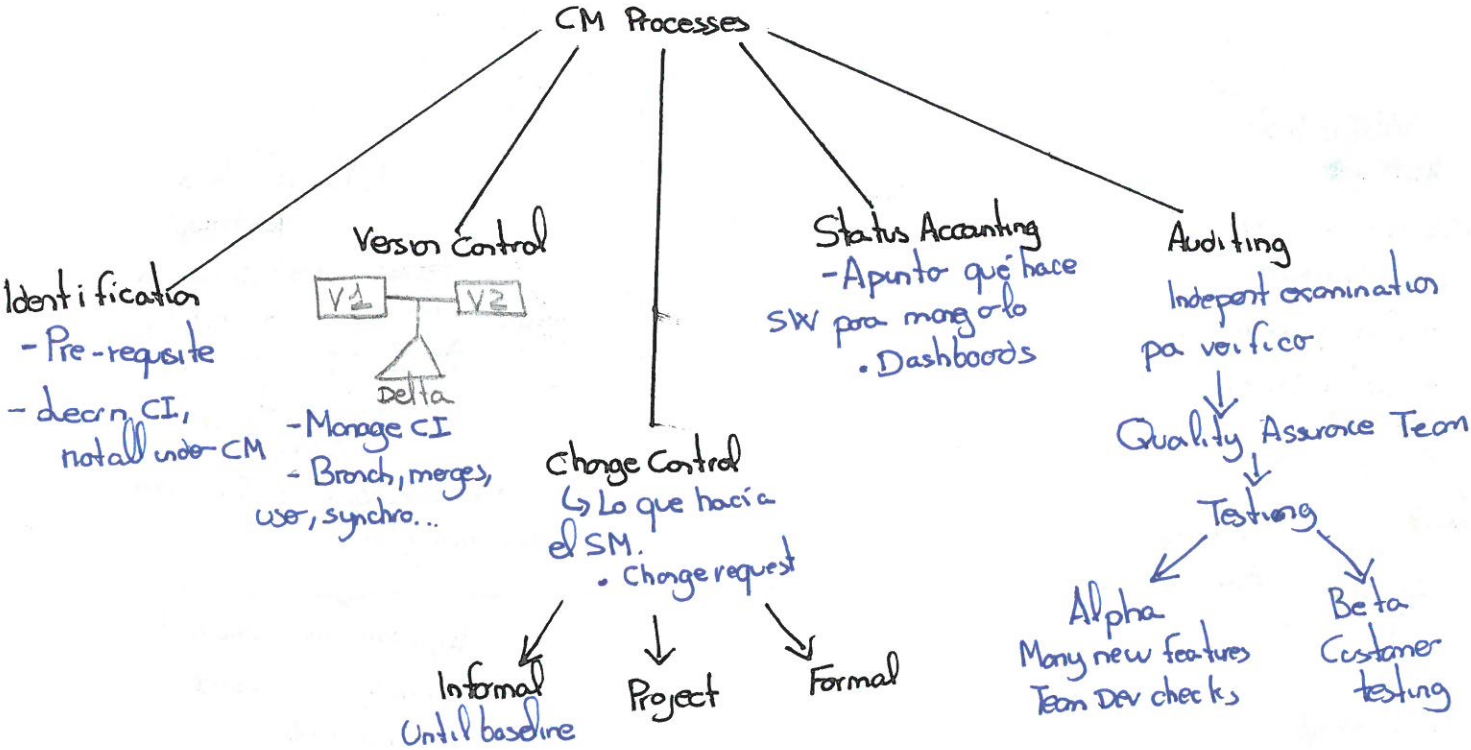
To manage easily

Configuration Item (CI): Part of SW / HW designated to CM as a single entity.

Example: source code, test documentation

Baseline: Approved version of a CI seen as a reference point for future development. Stablish a tend

Types: Requirement baseline, Design baseline...



1. Initiation
2. Classification
3. Analysis
4. Work
5. Implement
6. Verificate
7. Baseline

- Configuration: → Requirement, → Design, → Implementation } HW+SW ⇝ Specific version of system
- CI: HW/SW part as 1 entity → Source code, → Test documentation
- Baseline: Approved √√ version of CI for future develop
  - ↳ Requirement baseline
  - ↳ Design baseline

Unit 2. Techniques of SW Testing

Fault: Problem that could cause system fail

↳ **Failure:** The result of a fault

Test approach: Pa seleccona TC values

Test case → Input → exec. conditions → Expected output

Objective: Obtain failures // Break system

TWO ALTERNATIVES

White-box testing

Takes into account the inner structure

- Time consuming
- Use ful to see code defects

- Statement AC
- Decision AC
- Condition AC

Path Coverage

All paths at least once

Edges → E, Nodes → N

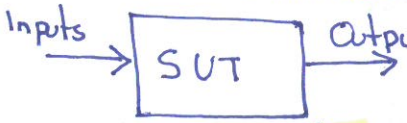
V(G): E - N + 2

V(G): P + 1 (P: conditions)

- Loop testing
  - → Nested, from inner
  - → Concatenated
  - → Unstructured, redesign
  - → Simple

Black-box testing

Tester doesn't know inner



correct?

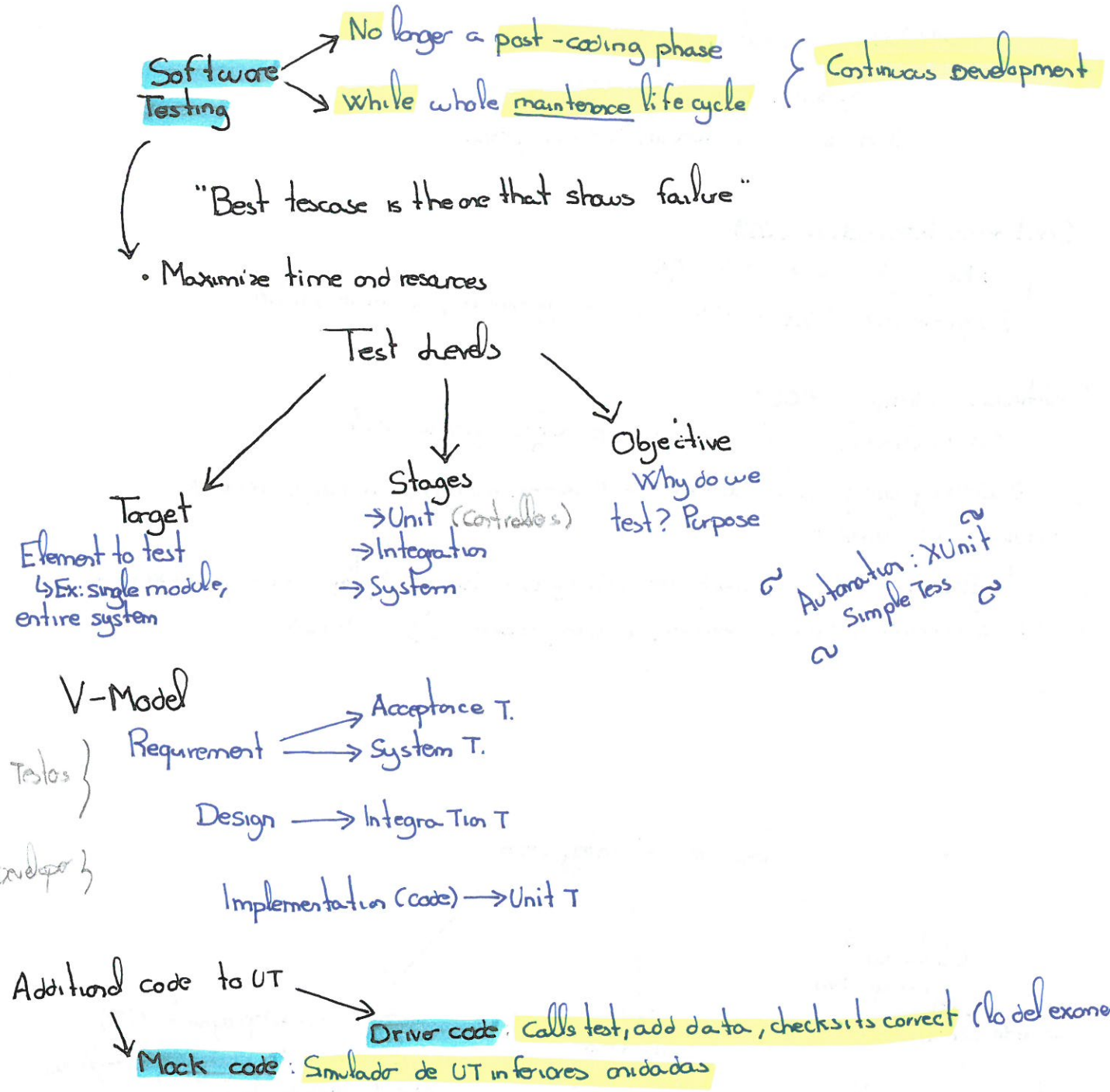
- Design set of TC with maximum yeld.

Equivalence classes

Boundary classes

Random (200...)

Unit 3. Unit testing



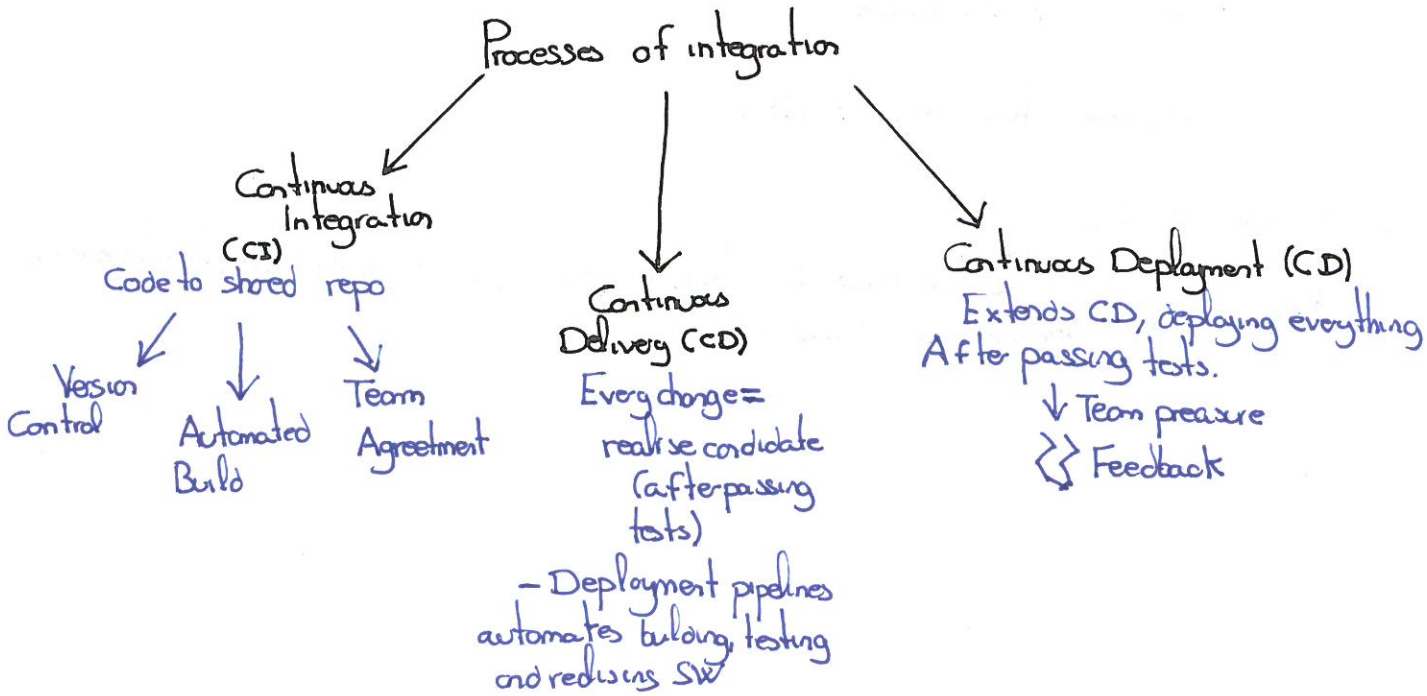
Unit 4. DevOps

DevOps: SW Develop + IT operations
- Speed vs operational Stability
- Automation through whole process

Continuous Integration (CI)
- New code into a shared repo
- Key element: Version Control, team agreements, automated build

Continuous Delivery (CD)
- Every change is a release candidate after passing test
- Deployment pipelines automates building, testing and realising software

Continuous Deployment
- Extends CD, doing that every delivery is automated deployed after passing tests
- Eliminates manual intervention, ↓ team preassure ⟆ Feedback



Unit 5 SWT. Processes and documentation

Test Management Processes (TMP): At project level

Dynamic Test Processes: Dynamic phase in a particular phase of testing

→ Design and Implementation

→ Environment Set-up and Maintenance (changes)

→ Execution (n times)

→ Incident report

Integration testing + + + + Until all whole system

↳ hierarchy designed

✓ Easy to find faults

✗ Driver and stub code

VS

Big-bang testing: All combined at once

Bottom-up

Down well tested

✓ Easy driver code

✗ Up bad tested, driver code.

System at the end

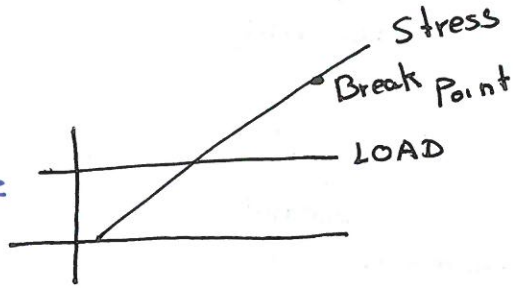
Top-Bottom

✓ Time to redesign

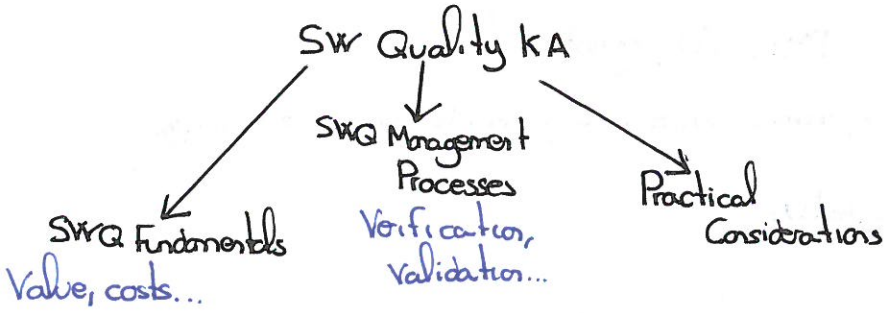
✗ stub code

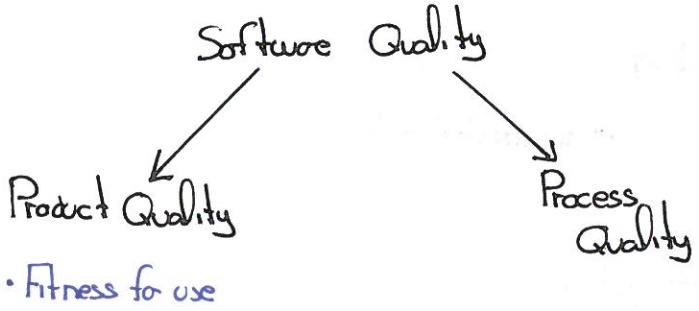
Load Testing ⇒ Se comprueba HASTA el límite

Stress Testing ⇒ Se pasa el límite hasta romperse

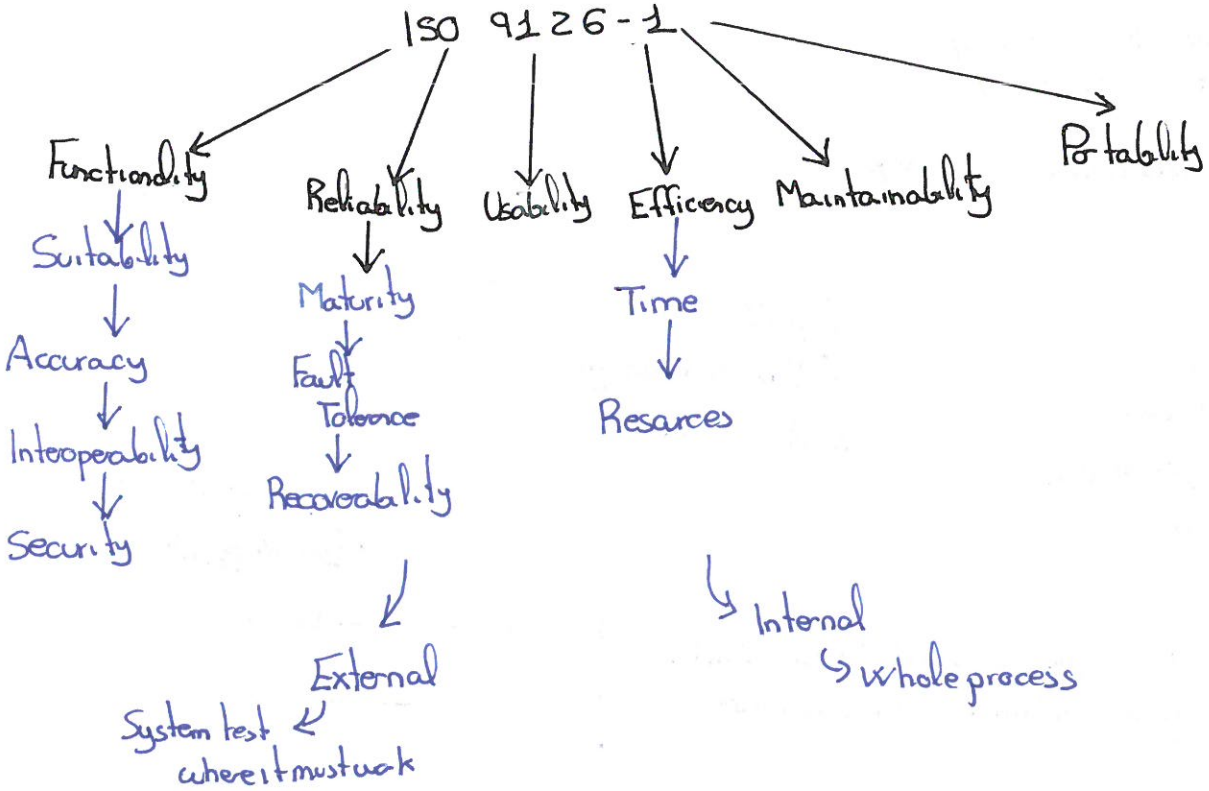


Unit 6. Software Quality





Quality: Subjetivo.
Aquello que está o no en el producto es característica de su calidad



Ejercicios Prácticas

1. Generation of test from Use cases

1.1.

Scenario	Flows
Sce 1	BF
Sce 2	BF + AF1 → Un AF por cada entrada del UC
...	...

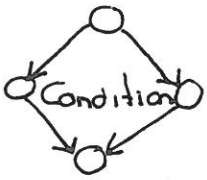
1.2

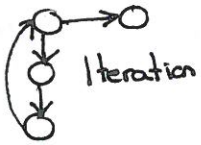
ID-TC	Sce/condition	Mandatory1	Mandatory2	...	Optional	Result

1.3 Special requirements / Pre-conditions

"All scripts must be runned..."

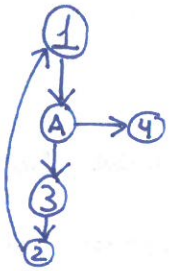
2. Path coverage

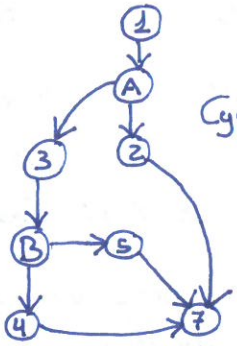




Example: FOR

```
1      A      2
for(int i=0; i<100; i++){
3  x=y
4  {
```





→ Smalls + Bigger one

CC = Closed Areas = 3

Cyclomatic Complexity = Conditions + 1 = 3

Path	Condition 1	Condition 2	...	Table 1	Table 2	...	Return
1, 2, A...							
...							

3. Equivalence class

Properties	Valid Inputs	Invalid inputs	Heuristics
Property 1			(*)
Property 2			
Property 3			
...			

(*) • Range of values: (Números)

Valid	Invalid	Heuristic
[1000 - 9999]	<1000	Range of values
	>9999	

• Finite number of elements: (String)

Valid	Invalid	Heuristic
[5-255] char	<5	Finite number of elements
	>25	

• Set of valid input:

Valid	Invalid	Heuristic
TO, AB, CR	MA	Set of valid input

• Boolean: Algo que deba ser X o Y. Ejemplos → Uppercase, No number

• Smaller classes: Cuando una propiedad puede simplificarse en mini clases

Ejemplo → Flightcode TX21233

Property	Valid input	Invalid Input	Heuristic
Flight Code			Smaller Classes
	3 char uppercase	<3	Finite number of elements
		>3	Boolean
		No uppercase	
	[1000-9999]	<1000	Range of values
		>9999	
		No number	Boolean

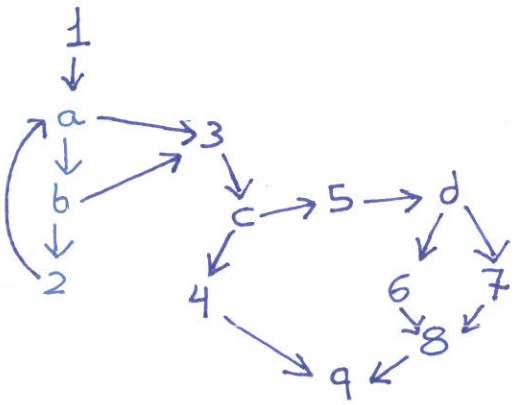
6. Given the following code for Crea[redacted] the Path Coverage technique to create its unit

12

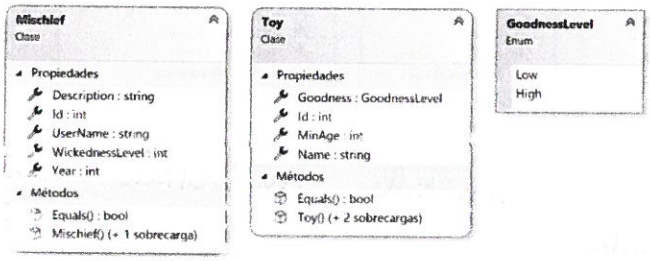
1. Given the following code for **GetToys** method in the **ToysController**, apply the **Path Coverage technique** to create its unit tests. This method automates **the step 2 of the Basic Flow as well as the AF1 of the previous Use Case**.

- The tables of the database **Toys, and Mischiefs** are also **input/output** for the test cases, depending on how they are used in the code. + moda-toy/optional
- What is indicated in the **return** instruction is considered as an output for the test cases.
- It is **not necessary** to define test cases for the **Where clauses**.

```
// GET: api/Toys
[HttpGet]
[ProducesResponseType(typeof(IList<Toy>), (int)HttpStatusCode.OK)]
[ProducesResponseType((int)HttpStatusCode.BadRequest)]
public async Task<ActionResult> GetToys(string UserName, int age, int year)
{
    //it obtains the mischiefs (travesuras) done by the child with UserName during that year
    IList<Mischief> mischiefs = _context.Mischiefs
        .Where(m => m.UserName.Equals(UserName) && m.Year == year).ToList();
    int wickedness = 0;
    int i = 0;

    while (wickedness < 100 && i < mischiefs.Count) {
        wickedness += mischiefs[i].WickednessLevel;
        i++;
    }

    if (wickedness >= 100)
        return BadRequest("Only sugar charcoal can be obtained");

    IList<Toy> toys;
    //it obtains the toys suitable for a child older than MinAge
    //and depending on his/her wickedness during the year

    if (wickedness >= 50)
        toys =await _context.Toys
            .Where(t=>age >= t.MinAge && t.Goodness==GoodnessLevel.Low)
            .ToListAsync();
    else
        toys =await _context.Toys.Where(t => age >= t.MinAge).ToListAsync();
    return Ok(toys);
}
```



Examples of instances and tables:
M1=("badness", 1,"bart@uclm.es", 150, 2023)
M2=("bad", 2,"homer@uclm.es", 70, 2023)
Ms={M1, M2}
T1=(GoodnessLevel.Low, 1, 5, "Buzz Lightyear")
T2=(GoodnessLevel.High, 2, 5, "Star Wars")
Ts={T1, T2}



Nº	Path	Username	Age	Year	Toys	Mischief	Return	Comment
	1,a,3,c,4,9							Imposible, si inicia a 0, el a aumenta
	1,a,b,3,c,4,9							Imposible, lo mismo
	1,a,b,2,a,3,c,4,9	jaleoab	14	2025	Ts	Ms	BadReq("Only sugar...")	covers a
	1,a,b,2,a,b,3,c,5,d,6,8,9	jaleoab	21	2025	Ts	Ms	OK(Toys)	covers b
	1,a,b,3,c,5,d,6,8,9	jaleoab	107	2025	Ts	Ms	Ok(Toys1,Toys2)	

1. Specify the test cases needed for testing the Use Case "Book holiday package" applying the technique "**generation of test cases from Use Cases**".

Precondition. User must be logged in.

Basic Flow of events

The use case starts when a user decides to book a holiday package and click on this option in the menu.

The system requests the traveling dates (**from**, **to**), whether such dates are strict, or they are the earliest and the latest possible dates. It also requires selecting which countries and types of travel (clubbing holidays, luxury holidays, beach holidays, winter holidays) she is interested in as well as the number of places. All these data are mandatory.

The user provides the required data and select next.

The system shows the available packages detailing for each one: traveling dates (**from**, **to**), country, type of travel, brief description, price and number of places available.

The user selects which package is of interest to her and select next.

The system shows the detail of the package (traveling dates (**from**, **to**), country, type of travel, brief description and total price) and requests the user to confirm her data (name, surname, id), the number of places she is interested in and travelling dates.

The user provides the required data and press next.

The system redirects the user to the Use Case <Process Invoice>

Extension Point. Use Case <Process Invoice> at step 8.

Alternative Flows

Alternative Flow to Step 4 – No mandatory data

If the system detects that user has not selected travelling dates, any country, any type of travel or number of places, then, it will inform user these are mandatory data and will come back to step 2.

Alternative Flow to Step 4 – No available packages

If the system detects that there are no available packages that satisfy user's selection, it will inform user and will come back to step 2 to ask for new data.

Alternative Flow to Step 6 – No selection

If the system detects that the user has not selected any package, it will inform her and will come back to step 4.

Alternative Flow to Step 8 – No mandatory data

If the system detects that the user has not provided all the data (name, surname, id), the number of places and travelling dates, or there are not enough available places, it will inform her and will come back to step 6.

TEST CASE SPECIFICATION

k) **Test items.** Holiday System v1.0. Use Case "Book holiday package"

l) **Input AND Output specification.**

Scenarios	Flows
Sce1	BF
Sce2	BF + AF0
Sce3	BF + AF1
Sce4	BF + AF2
Sce5	BF + AF3

ID	Scenario	From, to, country, type, Numplaces	Name, surname ID	Return
TC1	Sce 1	12/12/25 – 13/12/25, Spain, Holiday, 3	Javier, Garcia, 3	Process done
TC2	Sce 2	– 13/12/25, Spain, Holiday, 3	—	Mandatory m
TC3	Sce 2	12/12/25 – 13/12/25, , Holiday, 3	—	Mandatory m
...	...	...	...	...
TC10	Sce3	12/12/25 – 13/12/25, Spain, Holiday, 3	—	No Satisfy packages
TC11	Sce4	12/12/25 – 13/12/25, Spain, Holiday, 3	—	No package selected
TC12	Sce5	12/12/25 – 13/12/25, Spain, Holiday, 3	, Garcia, 3	Mandatory m
TC13	Sce5	12/12/25 – 13/12/25, Spain, Holiday, 3	Javier, , 3	Mandatory m
...	...	...	...	...

a) Enviromental needs In test plan

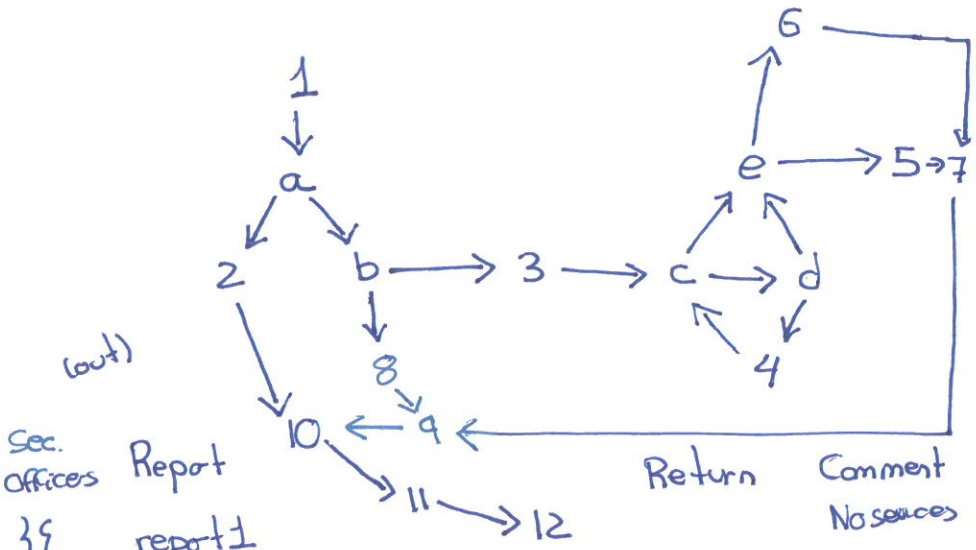
b) Special procedures

Run all scripts

2. Given the following code for a Create method in a controller, apply the **Path Coverage technique** to create its unit tests. This meth automates the step 6 of Basic Flow as well as AF3 and AF4 of the previous Use Case.

- The tables of the database **SecurityIncidentReports, and SecurityOfficers** are also **input/output** for the test cases, dependi on how they are used in the code.
- What is indicated in the **return** instruction is considered as an output for the test cases.

```
public class SecurityIncidentReportController : Controller{
public IActionResult Create(SecurityIncidentReport report)
{
1 {var officers = _context.SecurityOfficers.Include(so=>so.SecurityIncidentReports)
    .OrderBy(so=>so.SecurityIncidentReports.Count(sir=>sir.Status.Equals("Active"))).ToList();
  a
  if (officers.Count() == 0){
  2 report.Status = "Pending";
    ModelState.AddModelError("", "There are no available Security Officers");
  }
  else {  b
    if (report.TypeofIncident.Equals("Severe disruption to critical services (SDCS)")){
  3 {int i = 0;                                                    c                d
      while (i < officers.Count() &&  officers.ElementAt(i).SecurityIncidentReports
  4 {   .Any(SIR => SIR.TypeofIncident.Equals("Severe disruption to critical services (SDCS)")))
        i++;
                                  e
      if (i != officers.Count()) {
  5 {   report.SecurityOfficer = officers.ElementAt(i);
        report.Status = "Active";
      }
      else
      {
        report.Status = "Pending";
      6 ModelState.AddModelError("", "There are no available Security Officers");
      }7
    }
    else{
  8 {   report.Status = "Active";
    9 report.SecurityOfficer=officers.First();
    }
  10};
  report.CreatedDate = DateTime.Now;
11{ _context.SecurityIncidentReports.Add(report);
  _context.SaveChanges();
  return View(report);
12}
```



(in) (out)

Sec. Incident Report	Sec. officers	Report
{}	{}	report1
		report1

Path

1, a, 2, 10, 11, 12

1, a, b, 8, 9, 10, 11, 12

1, a, b, 3, c, e, 5, 7, 9, 10...

1, a, b, 3, c, d, e, 5, 7, 9...

1, a, b, 3, c, d, 4, c, e, 5...

1, a, b, 3, c, d, e, 6, 7...

covers c

covers d

Software Exercises

1- "Generation of test cases from Use Cases"

2- "Path coverage"

3- "Driver code"

4- "Equivalence class table"

2023-2024

Niño escribe a 3RM. Precond: Niño logeado en 3RMAPP

BF: 1- Niño select opt. escribir carta

2- System shows toys (name, goodness level, min age). At least selects 1. Depending age and

3- Child selects and next

4- System shows selection. Asks for deliv Address (compulsory), select high/low goodness level for next year (max 1 and letter (optional)

5- Child provides data

6- System shows everything

AF:

AF steps 1 and 2
↳ System detects niño malo, sale carta

AF step 2 and 2
↳ No toys avaible

AF step 3 to 6
↳ Mandatory data missing, returns to step 4 and error

a) Scenarios

Sce 1 — BF
Sce 2 — BF + AF1
Sce 3 — BF + AF2
Sce 4 — BF + AF3

b)

ID	SCE/condition	Toys	Address	Goodness	Letter	Output
TC1	SCE1/Goodness low	(Terrarcito, low, 12)	Aquí, mi casa	low	"La siento"	la siento, low, Aquí, mi casa (Terrarcito, low, 12)
TC2	SCE1/Goodness high	(Minecraft, high, 3)	Tmb aquí	high	"Tama ya!"	Tama ya, Tmb aquí, (Minecraft, high, 3)
TC3	SCE2/Worst child	"Only coal 4 you"				"Only coal for you"
TC4	SCE3 / NO TOYS					NO TOYS AVAIBLE
TC5	SEE4	(Minecraft, high, 3)	Pues aquí			MISSING DATA
TC6	SCE 4	(Minecraft, high, 3)		high		Missing Data

Equivalence class ID → Int, higher than 0 Name, string between 5 and 255

Goodness. Only Allows "High" "Low"

Min Age: Int between 0 and 80

Property	Valid Inputs	Invalid Inputs	Heuristic
Id	>0 (1)	<=0 (2) No number (3)	Range of values Boolean
Name	[5-255] char (4)	<5 (5) >255 (6)	Finite number of elements
GoodnessLevel	"High" (7) "Low" (8)	Anything else (9)	Set of valid values
MinAge	[0-80] (10)	<0 (11) >80 (12) No number (13)	Range of values Boolean

Status, valid values: New, pending, partiallyOngoing, Ongoing, closed

Name: String, no nulls and max lenght 10

Phone: String lenght 9, first num 6 or 9

Property	Valid Inputs	Invalid Inputs	Heuristic
Status	"New" (1) "Pending" (2) "Part. Ongoing" (3) "Ongoing" (4) "Closed" (5)	Anything else (6)	Set of valid inputs
Name	[1-10] char (7)	<1 (8) >10 (9)	Finite number of elements
Phone	[6,9] (10) [ccccccc-9999999] (11)	<9 char (12) >9 char (13) Else than 6 or 9 (14) No 8 int digits (15)	Finite number of elements Smaller classes Boolean Boolean

Types Of Travel → valid: ClubbingHolidays, LuxuryHolidays, BeachHo, WinterHo

Country → String, no null, max length 15, just alphabetic

Number of Places → Greater than 0, less than 20. Integer

Property	Valid Inputs	Invalid Inputs	Heuristic
Types of Travel	"ClubbingHo" (1) "LuxuryHo" (2) "BeachHo" (3) "WinterHo" (4)	Anything Else (5)	Set of valid inputs
Country	[1-15] char (6)	<1 (7) >15 (8) No alphabetic (9)	Finite number of values Boolean
Number of places	[1-19] (10)	<1 (11) >19 (12) No number (13)	Range of values Boolean