UNIVERSIDAD DE LAS FUERZAS ARMADAS

“ESPE”

INGENIERÍA EN TELECOMUNICACIONES

ESPE- OOP-TC

NOTE BOOK

EDISON LASCANO, PhD

NAME:

KATHERIN BRAVO

NRC

7490

DÍA MES AÑO NOTA /

UNIT 1

GIT PUSH AND RUN

Managing Software Development Artifacts

Some common artifacts:

- Source code
- Build scripts
- Implementation notes
- Test cases
- Deployment scripts
- Data files
- Design documents
- Requirement definitions.
- And more

Version control Systems.

- Workspace: holds what the developer is currently working only.

Repository

Keep track of all work artifacts, with a complete change history.

- GIT

- Git is an open-source distributed VCS, and perhaps currently the most popular.
- Git can track changes for any kind of file.

Workspace

Working Files | Staging Area | Repository (clone) Git <—> Git Remote Repository

NOTA /

DÍA MES AÑO

Repository

done

- cd _(ruta)
- git clone _link

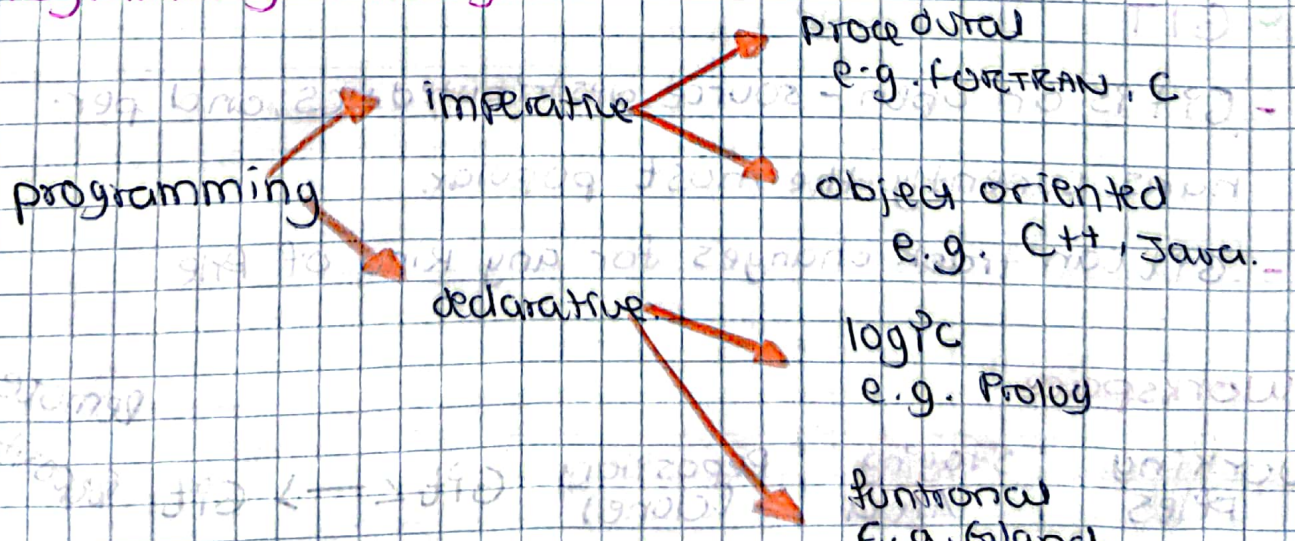
- git pull
- git add -A
- git commit -m " "
- git push.

↳ DONE

- git pull
  → conflict

- We will not cover
  - Branch
  - Tags
  - Checkout
  - Fetches
  - Reset
- And other advanced operations that may be needed to resolve conflicts.

Programming Paradigm



DÍA MES AÑO NOTA /

Declarative versus imperative

Declarative	Imperative
you say what you want	you say what you want
But NOT how you want it	And how you want it

- OBJECT IDENTITY

- Many of those properties may not be observable or interesting to a software system.

- In a software system, even if two objects have the same known properties, they are not the "same" object.

Freedom for the programmer

NOTA / DÍA MES AÑO

ENCAPSULATION

- See David Parnas' paper from 1972.
- Encapsulation is the act of "hiding" or placing decisions in software component like a method or a class.
- In software systems, good encapsulation involves both the
  - Hiding of design decisions
  - Localization of desing decisions

ABSTRACTION

- Abstraction is the act of summarizing or generalizing something to focus on the ideas most relevant to a conversation or certain kind of communication.
- An abstraction (the noun) is a description that leaves out unnecessary details.
- "interfaces", as found in C# and Java, are abstractions, but so are abstract classes, abstract methods.

DÍA MES AÑO NOTA 7

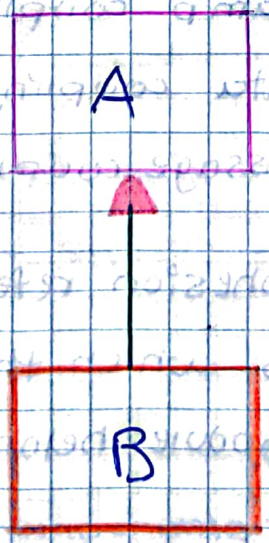
Classification

- Classification (the verb) is the process of grouping objects together into sets based on common properties.
- A classification (the noun) or "class" is a set of objects that share certain common properties.

Sub-classification

Specialization

- Sub-classification, also called specialization involves identify subsets of another class
- In software, class inheritance is a reuse mechanism, where a class B's definition is based on (reuses) class A's definition



NOTA 7 DÍA MES AÑO

Coupling

- Coupling is the manner and degree of interdependence between software components.
- When there is high coupling then there is a change to a software components causes a ripple changes in other components.

Types of Coupling

- Content coupling
- Common coupling
- Control coupling
- Stamp coupling
- Data coupling
- Message coupling

Cohesion

Cohesion refers to the degree to which the elements of a module belong together.

DÍA MES AÑO NOTA /

Types of cohesion

Causes of bad cohesion, in an approximately severe-to moderate order.

- Coincidental cohesion
- Logical cohesion
- Temporal cohesion
- Procedural cohesion
- Communicational / informational cohesion
- Sequential cohesion
- Functional cohesion

Modularization

- Modularization (verb) is the process by which developers break up a system into cohesive and loosely coupled components.

- A modularization (noun) is the specific decomposition characterized by the components' abstractions.

NOTA /

DÍA MES AÑO

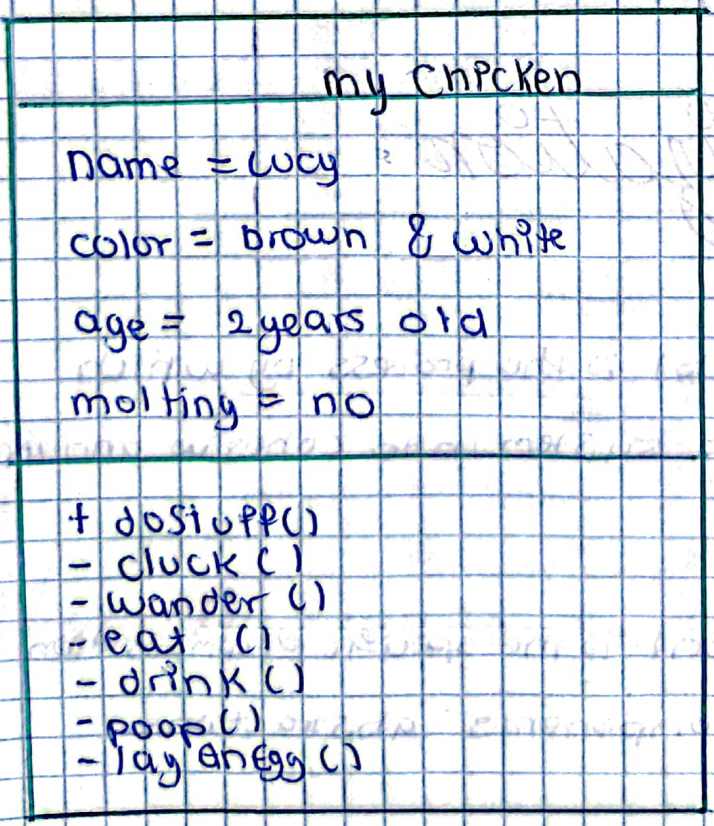
Real - World - Objects

- An object is anything that has a boundary in space or time.
- Objects have state and behavior

Objects.

- Every object is a unique entity in the universe, distinguished by its properties.
- Object may include other objects as part.

Software objects.



DÍA MES AÑO NOTA /

Conceptual Modeling

▲ What?

Capture ideas and concepts in a form that allows other activities, reasoning, analysis, prioritization, design, etc.

▲ Why?

To be better understand a real-world system.

To visualize a system as it is.

To envision a system as we want it to become.

▲ When

Any time during the software development process starting at very beginning and from:

- Analysis
- Design
- specification
- implementation
- Deployment

Conceptual Modeling with UML

UML (Unified Modeling Language)

- Use case diagrams - for describing those who need a software system and their goals.

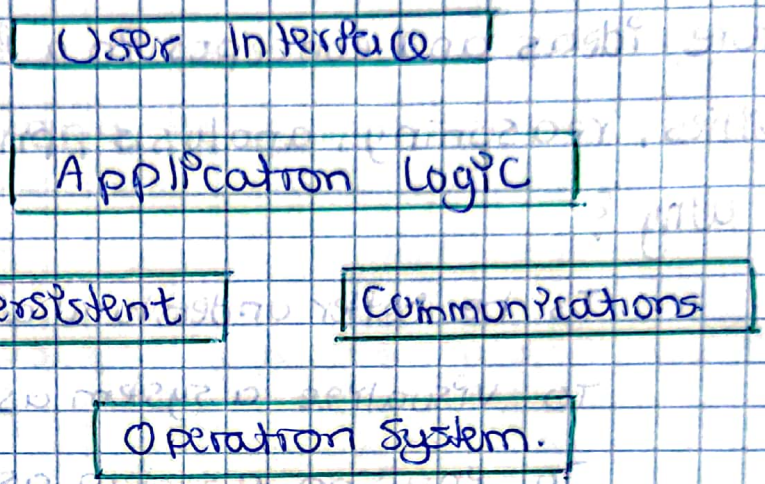
NOTA /

DIA MES AÑO

1 Organize Software into Loosely Coupled Layers !

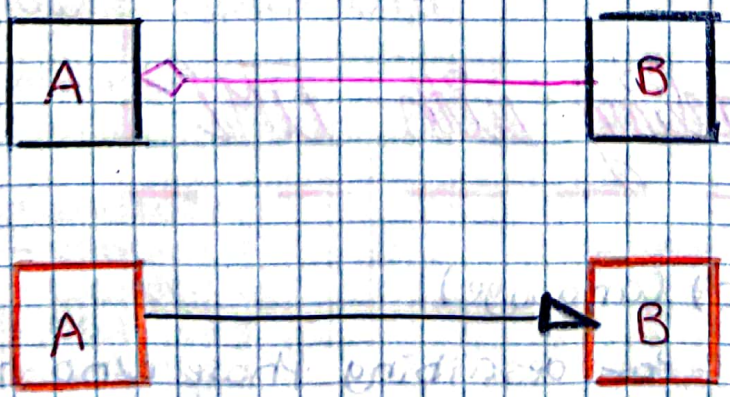
Design your software sistem in layers.

Example:



2 Prefer Aggregation Over Inheritance !

- Agregation. e.g. object A contains object B and therefore can "reuse" object B).
- Inheritance, e.g. class A inherits from class B.



DÍA MES AÑO NOTA /

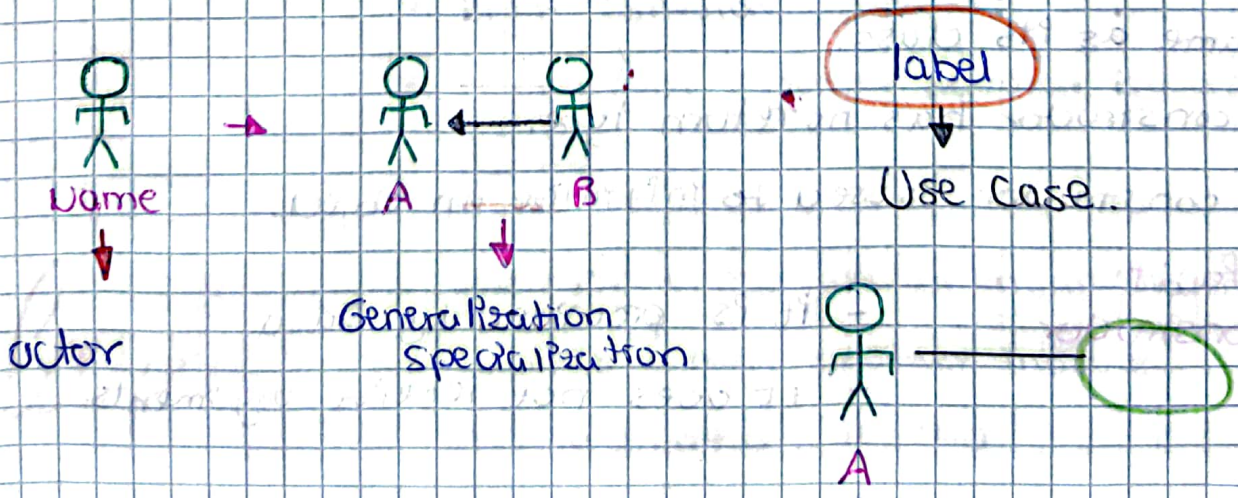
UNIT 2

Test Cases

- Testing individual units of code with executable unit test cases.
  - Helps find design errors.
  - Helps document the usage of a method or class.
  - Enables regression testing.

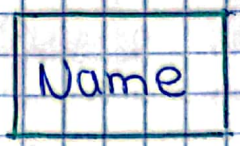
UML - Quick Reference

Keep an "Analysis" perspective.
- Model the "real-world" problem or at least the world as the stakeholders with like it to be.
- Don't try to model software components at this point.
- Keep diagram readable
- Avoid line crossings.
- Use color effectively

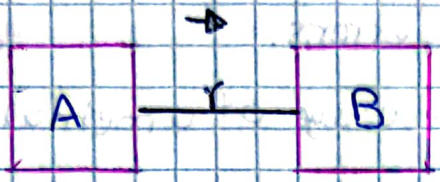


NOTA / DÍA MES AÑO

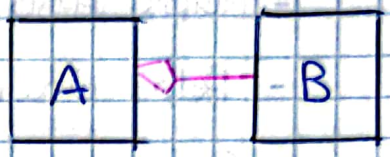
TITLE



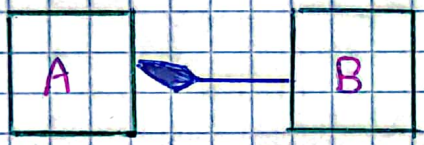
Class



Named Binary Association



Aggregation



Composition

Constructors

A constructor is a method used to create an instance of a class.

A constructor (in java) is a method with the same name as its class.

A constructor has no return type.

A constructor is used to initialize an object.

Default Constructor

- It is provided by Java.
- It does not receive arguments.

DÍA MES AÑO NOTA /

Modularity

Observations relative to modularity

- Earlier and seminal work by David Parnas (1972, plus later works).
- Design decisions need to be identified and implemented in one place. Here, we call this Localization of Design Decisions.

Definition: Modularity exist in a software system when it is comprised of loosely coupled and cohesive components that isolate each significant or changeable design decision in one component and ensure that related pieces are as close as possible.

Adherence Criteria

- Localization of design decisions
- Low coupling
- High cohesion
- Modular or Extented Modular Reasoning
- Congruency

Coupling

One module depends on encapsulated concepts within another module instead of an abstract

NOTA /

DÍA MES AÑO

Cohesion

- Components grouped together arbitrarily.
- Components grouped together because they process close together in time

Modular Reasoning

Definitions:

- Modular reasoning
- Extended modular reasoning.
- Global reasoning

Congruency

In OO software development a class is congruency if its defined properties match the common properties of all its objects.

Tradeoffs

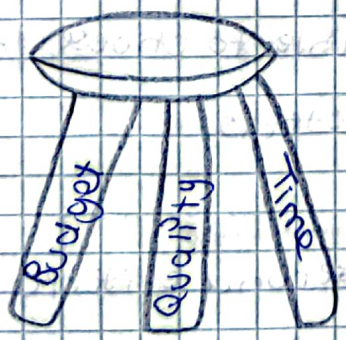
Localization of design decisions and high cohesion can lead to many fine grain components which is good for testability, extensibility and resue.

Software Engineering Goals

Software engineers aim to build quality products on time and within budget.

Qualities:

- understandability
- testability
- maintainability
- efficiency
- reliability
- security
- extensibility
- openness
- interoperability
- reusability



NOTA / DÍA MES AÑO

Abstraction

Background:

From a process perspective, abstraction is the act of bringing certain details to the fore front while suppressing all others.

BACKGROUND

- Creating good software abstractions is hard
- Abbott et al. described an abstraction as the "replication and conceptualization of distinction"

Wrap Up

conditions for non-redundacy and complimentory of the AME principles

1.- No principle can be a special case.

2.- Developers should be able to choose to follow one principle but not the others.

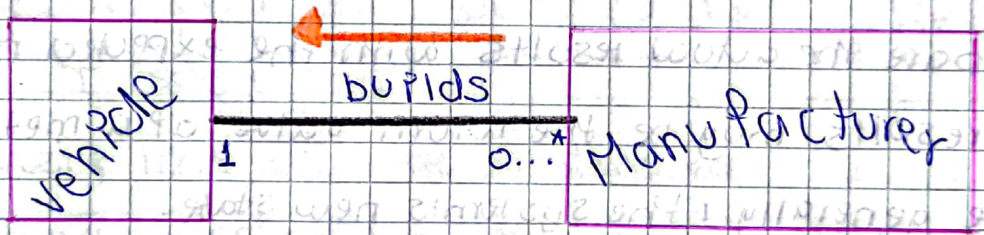
Criterion #2.

We show satisfaction of the second criteria, namely that developers and choose to follow each principle independent, with an example consisting of four functionally - identical code snippets.

DIA MES AÑO NOTA 7

Introduction to Aggregation

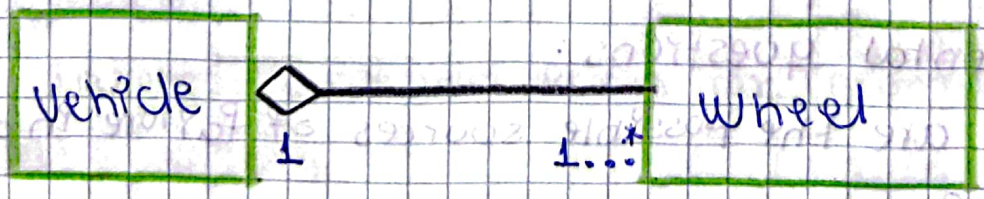
- Associations model set of links between classes of objects.
- Specifically, a binary association connects two classes, e.g. Vehicle and Manufacturer and describes a set of link such each link connects an object from one of the classes with an object from the other.



This aggregation can be read as

- Vehicle contains 1 or more wheels

Other synonyms are possible too



NOTA / DÍA MES AÑO

Conducting a Test

Three general steps

setup:

Ensure the system is in a know state, or at least the component you are testing is in a known state e.g. create a triangle object with specif sides.

simulate. Execute the thing you are testing e.g. the computeArea() method

Observe

compare the actual results with the expected results The results may be the return value of a method, or more generally, the system's new state.

Test Case Selection

Selection of Appropiate Test Data

Fundamental questions:

- what are the possible sources of failure in a program?
- what test data should be selected to demonstrate that failures do not arise from these sources?

DÍA MES AÑO NOTA /

Convenient Notations

Shorthand for success / failure of a test case when F is understood:

$$OK(d) = OUT(d, F(d))$$

Shorthand for success/failure of a whole test suite, T:

$$SUCCESSFUL(T) = \forall t \in T(OK(t))$$

Sample Criteria for Modified Path Testing

Choose T such that

- Every line of code is executed.
- Every possible branch of a condition tried.
- Boundary conditions for loops are tried.

Nature and Cause of Software Errors.

Terminology:

failures - observed errors or inconsistencies.

What's a "bug"

This is an overloaded terms that people informally use for both failures and faults

NOTA /

DÍA MES AÑO

Sources of Software Errors.

- Memory management errors
  - confusing references / pointers
  - out-of-bounds references
  - Initialization
- Understanding semantics of other components.
- configuration and environment
- Timing errors.

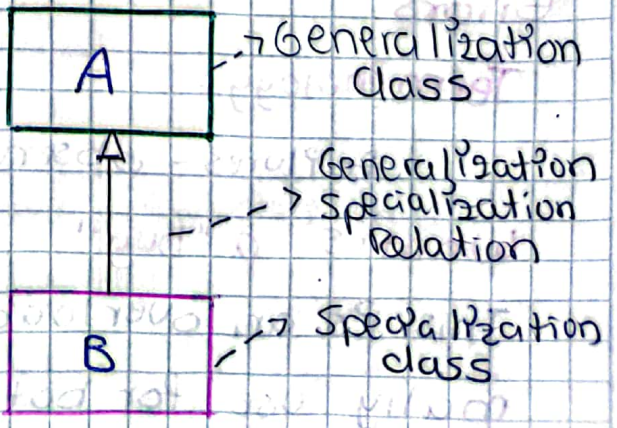
Inheritance (Generalization / Specialization)

When we classify (group objects into set according to common properties, it is natural to also identify.

Example:

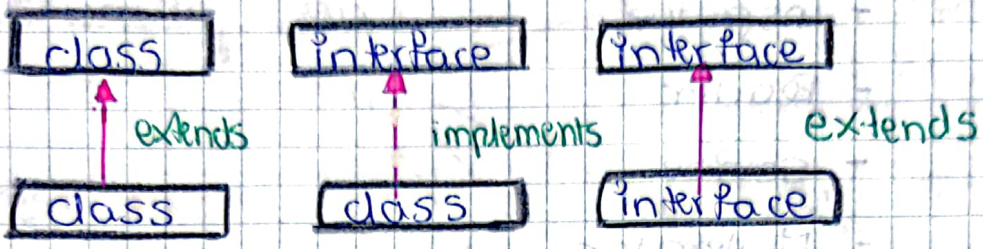
After grouping small, furry, four-legged pets that meow into the class Cat, someone might.

UML Notation for Generalization / Specialization



DÍA MES AÑO NOTA /

Inheritance and Implementation



Introduction to Reviews

Overview

- **Inspections:** Formal validation/verification of artifacts where reviewers direct the process.
- **Walkthroughs:** Informal review where author directs or guides the review and reviewers ask probing questions.

PROS	CONS
- Review validate/verify non-executable work products - Reviews can help validate/verify executable work products without execute	- It is expensive to conduct reviews after every change. - Reviews are unlikely to find certain types of errors that are better.

NOTA /

DÍA MES AÑO

Inspections

Roles:

- Author
- Inspector
- Reader
- Scribe
- Moderator

Walkthroughs

- An individual may play multiple roles

For example, the author may be scribe and moderator.

- When done formally, inspections require:
  - Planning meeting
  - Inspection event

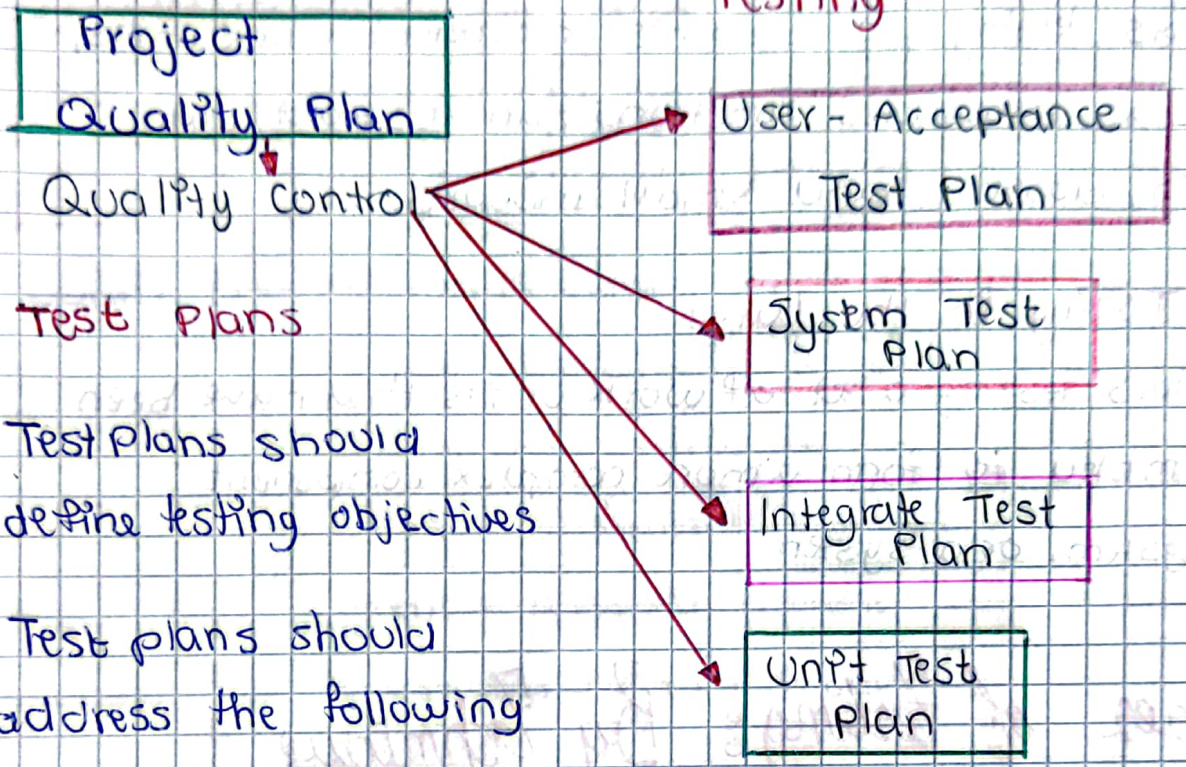
which can be done all together or individually

- follow up
- Inspections are usually initiated by a team.

DÍA MES AÑO NOTA /

Unit Test

Planning For Successful Testing



Test Plans

Test Plans should define testing objectives

Test plans should address the following

Test plans contents (continued)

- Who will do the work?
- How will the testing work be managed?
- How will testing be coordinated with other development activities?

By scope

The entire system a subsystem, a module a single function, etc

By technique

Path test, input-validation testing, logic-based testing, stress testing load testing, etc.

NOTA /

DÍA MES AÑO

Types of Testing : By Scope.

- Testing in the small

involves testing code elements or coherent work units, such as procedures, functions, an abstract, data types, classes, and small modules.

- Testing in the large

involves testing a set of work units that have been assembled to form a more complex component subsystem, or a system.

Types of Testing: By Technique

- Reduce the number of test.
- Each testing technique focuses on a different asped on the software.
- Different techniques uncover different faults.
- Regardless of technique, test cases consist of.

setup (or stimulus)
predicated results
comparison of observed results to predicated results.

UNIT 3

Introduction to the Solid Principles

SOLID is a mnemonic acronym for five principles.

- Single Responsibility Principle
- Open/closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency inversion principle.

OPEN / CLOSED PRINCIPLE.

Core Ideas

Software entities (e.g. classes generics)

Original definitions.

A class is open if it is still available for extension

A class is closed if its is available for use by other class, and therefore should not be modified.

Revised definitions.

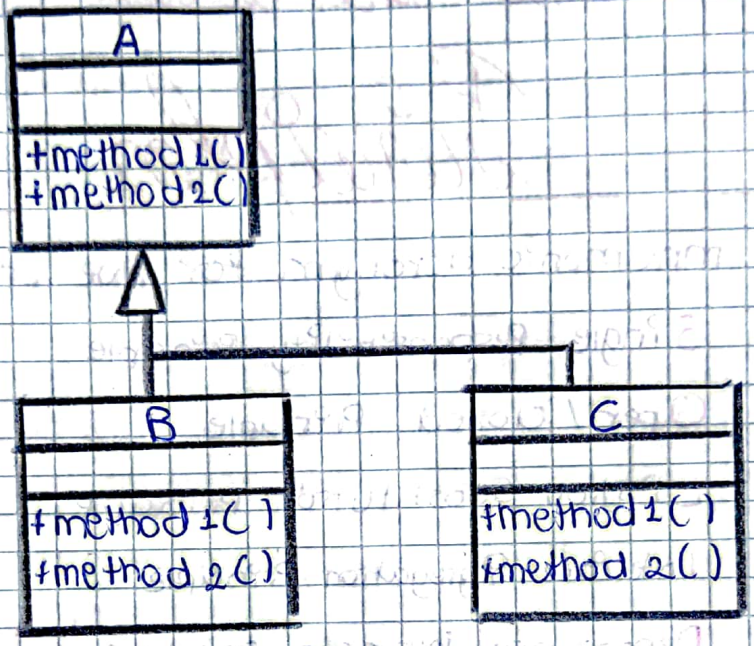
A system of classes is open for extension and closed for modification, if.

NOTA /

DÍA MES AÑO

ESTUDIO

Interfaces.



Abstraction, Modularity, and Encapsulation

- The Abstraction, Modularity, and Encapsulation (AME) principles have value in all popular software development paradigms, albeit to different degress.
- Present consistent, actionable definitions for the AME principles.

DÍA MES AÑO NOTA /

Purpose of Principle

Principle.

1) a truth or proposition that supports.
2) a rule or code of conduct
3) an ingredient that imparts a characteristic

Localization of Design Decision and Encapsulation

What are the classes of objects in the system? ✓

UML Class Diagram: Classes.

How do the definition of classes depend on each other ✓

UML Class Diagram: Dependencies.

What are the structural relationship among the objects of those classes? ✓

UML Class Diagram: Associations and class variables.

Localization of Design Decision involves.

- Identifying individual design decision and then
- Implementing that decision one place.

Duplicate code is a consequence of not localizing design decisions.

NOTA /

DÍA MES AÑO

Common Paradigms

Object orientation (OO)
Aspect orientation (AO)
Functional programming (FP)
Logic Programming (LP)
Genetic Programming (GP)
Structured program (SP)

Today, many languages and development environments support multi-paradigm software development (MPSD)

Core Problem

There are no general, unifying definitions especially for multi paradigm software development

Purpose of principles.

1) a truth or proposition that supports reasoning.
2) a rule or code of conduct.
3) an ingredient that imparts a characteristic quality.