PARCIAL 1

Encapsulation keeps states private

Inheritance is the implementation of DRY

Association 1:1, 1:N, N:N

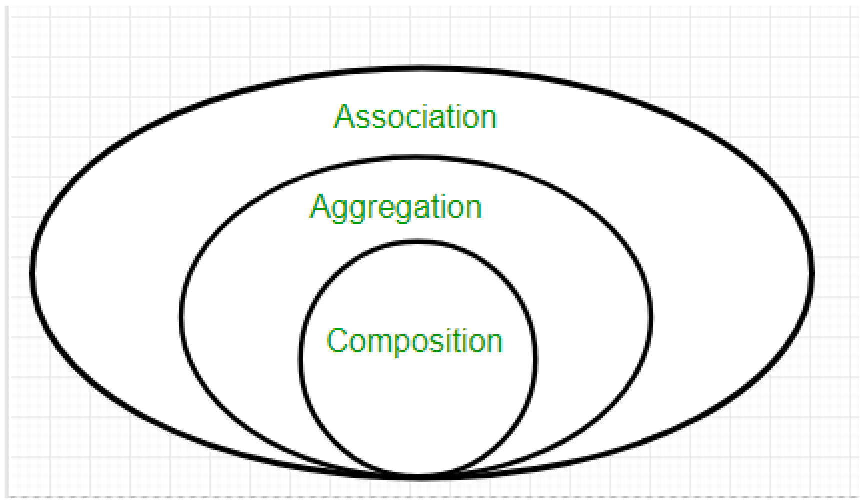
Aggregation one to many, one way , the child may exist

Composition the child must exist, has.a association

Is one of the fundamental concepts in object-oriented programming. It describes a class that references one or more objects of other classes in instance variables. This allows you to model a *has-a* association between objects.

Composition

Composition is a restricted form of aggregation



Select all the attributes that would be part of the Driver class in Uber:
Name, User ID, Address, Car

Select all the aspects of the architecture profession
Technical Knowledge, Leadership & Communication; Methodology & Strategy

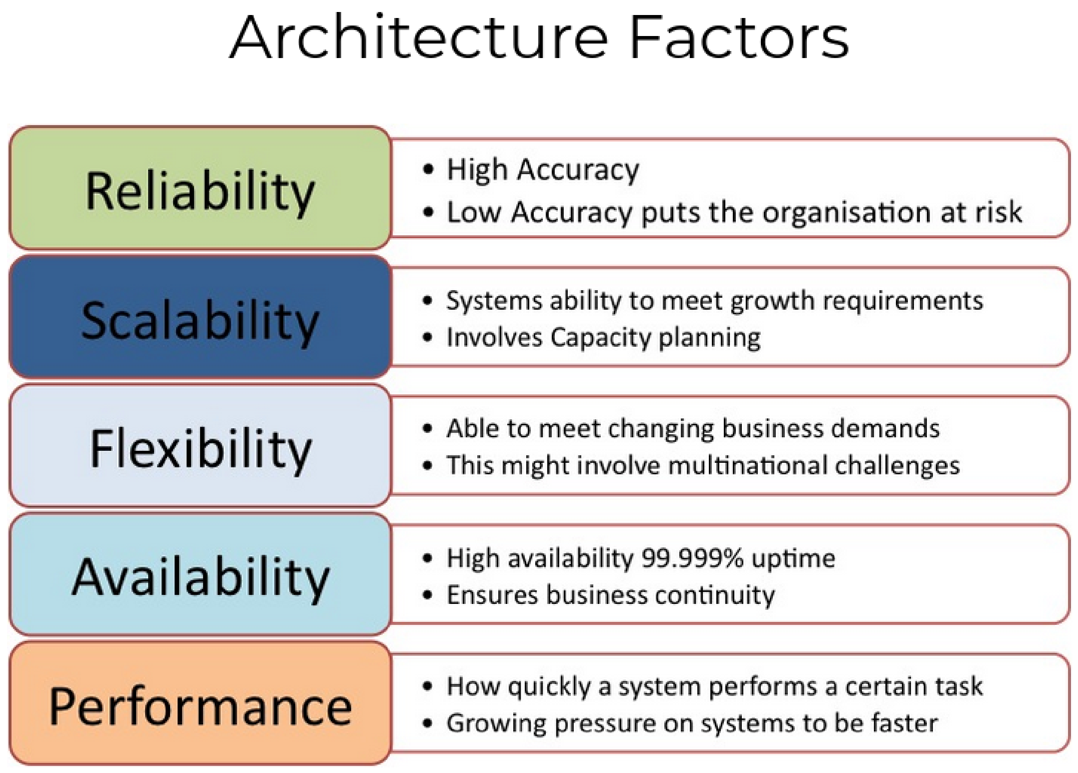
Single Responsibility Principle
Open/close principle
Liskov substitution principle
Interface segregation principle:
Si una clase tiene muchos métodos se debería de dividir en varias clases que se pueden composicional dentro de otra

Dependency Inversion principle:
Las clases más abstractas deberían de tener componentes de las clases menos abstractas no al revés.

Todo los que usan una interface(clase padre) deberían de poder usar todos los metodos que este tiene
Dependency inversion principle:
High level components should not depend on low level components; webapp should not depend on stripe or paypal

DRY (Don't Repeat Yourself)
KISS(Keep It Simple Stupid)
YAGNI(You Aren't Gonna Need It) Something should not be built if youre not sure youre going to need it
Demeter Principle: Each unit should only talk to its friends; don't talk to strangers.Each unit should have only limited knowledge about other units:
Hollywood, well call you API

System faults
-software faults
-hardware faults
-human faults



Architecture Factors

Reliability	• High Accuracy • Low Accuracy puts the organisation at risk
Scalability	• Systems ability to meet growth requirements • Involves Capacity planning
Flexibility	• Able to meet changing business demands • This might involve multinational challenges
Availability	• High availability 99.999% uptime • Ensures business continuity
Performance	• How quickly a system performs a certain task • Growing pressure on systems to be faster

Reliability que regrese los resultados esperados y válidos
Scalability poder manejar mayor carga (vertical y horizontal)

Flexibility es la facilidad para realizar cambios
Availability que siempre está operando bien
Performance

Software Architect Responsibilities

- Analyze technology, market trends and try to keep them
- Analyze current technology and recommend solutions for improvement
- Ensure compliance with the architecture. Let the team know what is the architecture
- Make the technology decisions
- Understand the company policies and rules and make decisions that dont interfere

PARCIAL 2

Métodos tradicionales
-waterfall
-iterative
-Agile
Agile Manifesto

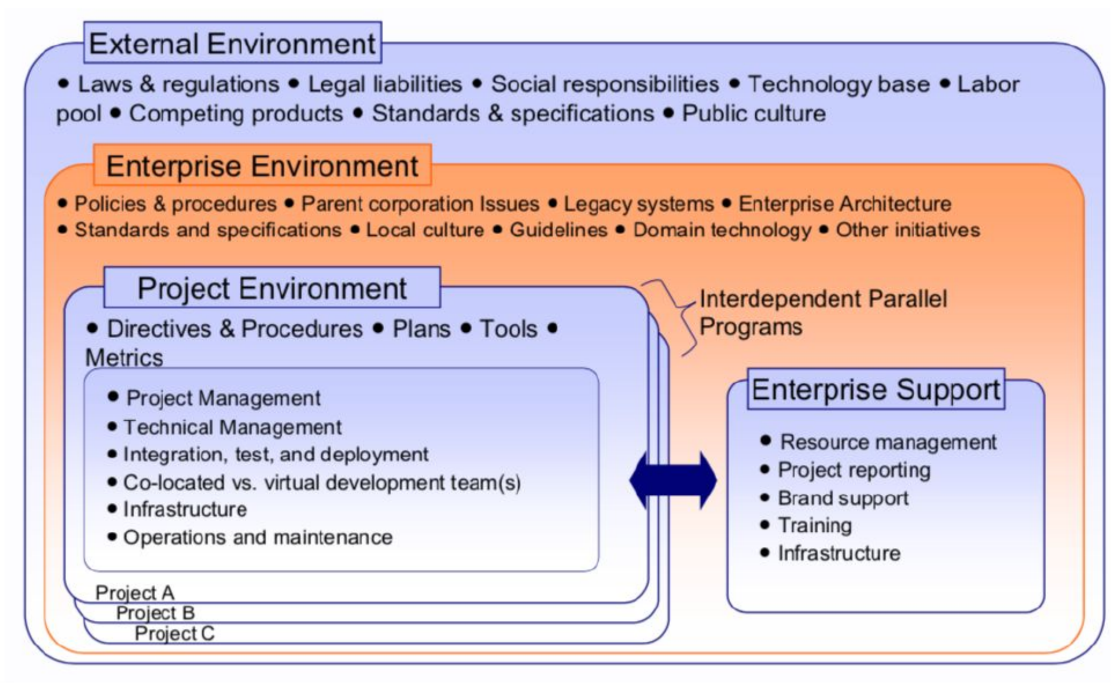
●Individuals and interactions over processes and tools.
 ○ Working software over comprehensive documentation.
 ○ Customer collaboration over contract negotiation.
 ○ Responding to change over following a plan.

●Scrum is a management and control process that cuts through complexity to focus on building software to meet business needs. Scrum is superimposed on top of and wraps existing engineering practices, development methodologies and standards. [Schwaber]

How To Start?
The architecture usually starts from a set of requirements or use cases, where the customer specifies the desired functionality and system attributes.
This process has to be followed no matter the development methodology adopted by the software development team.

What are requirements?
Functional Requirements:
– Are capabilities needed by users to fulfill their job– Answers the question of "what" the customer needs (but often not "how" it is achieved)
Qualities:
– Define the expectations and characteristics that the system should support
– Might be runtime (for example, performance or availability) or non-runtime (for example, scalability or maintainability)
Constraints:
– Given those things that cannot be changed within the scope and lifetime of the project
– Other factors, such as mandated technologies, available skills, and budget
Qualities and constraints are sometimes referred to as “non- functional requirements”

The wider project environment


External Environment
• Laws & regulations • Legal liabilities • Social responsibilities • Technology base • Labor pool • Competing products • Standards & specifications • Public culture
Enterprise Environment
• Policies & procedures • Parent corporation Issues • Legacy systems • Enterprise Architecture
• Standards and specifications • Local culture • Guidelines • Domain technology • Other initiatives
Project Environment
• Directives & Procedures • Plans • Tools • Metrics
• Project Management
• Technical Management
• Integration, test, and deployment
• Co-located vs. virtual development team(s)
• Infrastructure
• Operations and maintenance
Project A
Project B
Project C
Interdependent Parallel Programs
Enterprise Support
• Resource management
• Project reporting
• Brand support
• Training
• Infrastructure

Use cases for Functional Requirements

Sample Use Case

Main scenario

1. The system requests the person's details.
2. The user supplies family name and town.
3. The system supplies the address and phone number.
4. The use case ends successfully.

Alternatives

3a. Non Unique name or town

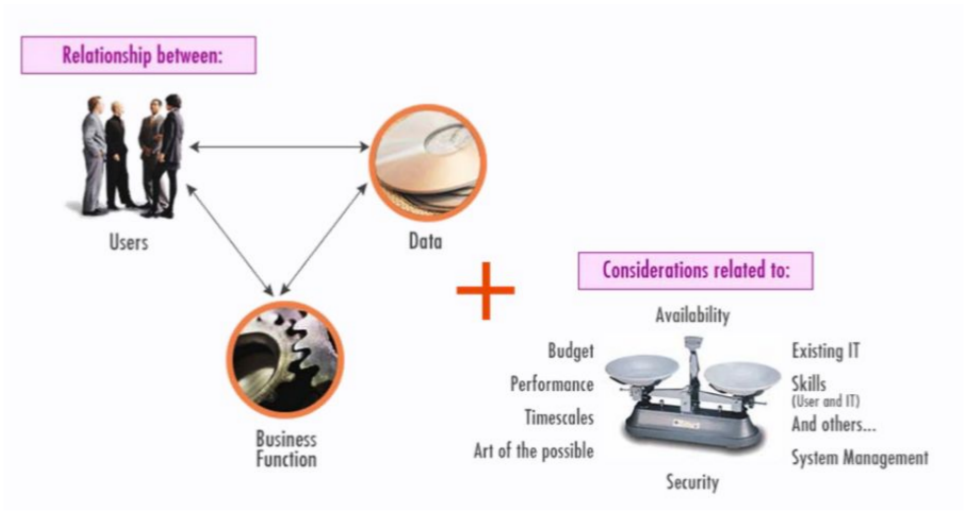
3a1. The system supplies a list of full names and post codes.

3a2. The user selects an entry from the list.

3a3. The system supplies the address and phone number.

3a4. The use case ends successfully.

What key requirements drive the shape of the architecture?



Significant Requirements
Architecturally significant requirements are ones that are often related to:

- High business payback functions
- Critical business operations
- High volume, high usage functions
- Essential functionality of the system
- Influential users
- Mobile users
- Have broad coverage and exercise many architectural elements
- Security requirements
- Complexity (such as complex business rules)
- Stringent demands on the architecture (such as performance and availability requirements)
- High potential for change (such as rules relating to taxation)
- Communication and synchronization with external systems
- Those which highlight identified issues/risks

Apply 80/20 rules – which 20% of the requirements drive 80% of the user activity?

Requirements are Dynamic
Reasons for change
– Users' needs and technology change as time passes
– As the system evolves, requirements become clearer
– External changes to the environment make requirements obsolete or create new requirements
– Change might be needed to manage requirement conflicts between different stakeholders

Change management

- Iterative documentation, review, and sign-off of requirements
- Sufficient emphasis on developing change cases
- Well-defined and executed change management procedures
- Ongoing evaluation of the application and the architecture

Requirements should be SMART
<u>Specific</u>: Must be unambiguous, consistent, and be at the appropriate level of detail
<u>Measurable</u>: Must be possible to verify that a requirement has been met, so include success criteria
<u>Attainable</u>: Must be technically feasible and be within the art of the possible
<u>Realizable</u>: Must be realistic given all the constraints (e.g., resources, skills, time, infrastructure)
<u>Traceable</u>: Must be linked from conception through specification, design, implementation, and test

Pitfalls

- The "This Is Too Technical for Me" Attitude
- The "All Requirements Are Equal" Fallacy
- The "Park It in the Lot" Problem
- The "Requirements That Can't Be Measured" Syndrome
- The "Not Enough Time" Complaint
- The "Lack of Ownership" Problem
- The "All Stakeholders Are Alike" Misconception
- The "Too General" Tendency

Requirement Rules of Thumb

1. A requirement must form a complete sentence.
2. Avoid using abbreviations unless they are defined.
3. Make consistent use of one of the following verbs:
   – Shall, Will, or Must to show mandatory requirements
   – Should or Might to show optional requirements
   – Could to show desirable requirements
4. They must contain the success criteria and be measurable and testable.
5. A unique reference ID must be provided; for example, NFR-PERF-001.
6. Reference supporting material, do not duplicate information
7. Avoid:
   – Ambiguity (instead, write as clearly and explicitly as possible)
   – Rambling, long sentences with arcane language
   – Wishful thinking (such as “100% reliable”, “totally safe” and so on)

Use Cases Rules of Thumb

1.A use case is what one person does in one place at one time.
2.A use case step is:
– A sentence in the present tense
– With an active verb
– Describing an actor achieving a goal that moves the process forward
3. Use cases are primarily textual, not “matchstick men and ellipses.”
4. Do not include “system internals” in a use case.
5. Do not have branch statements in a use case (these should be “alternative paths”).
6. Keep the GUI out.
7. Define use cases by working “breadth first.”

Issues in System Design

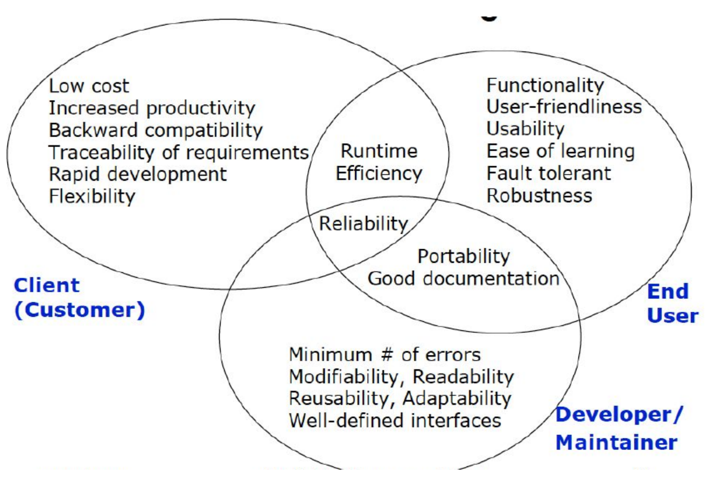
1. Identify Design Goals
2. Subsystem Decomposition
3. Identify Concurrency
4. Hardware/Software Mapping
5. Persistent Data Management
6. Global Resource Handling
7. Software Control

Design Goals (Expanded)

Rapid development	Ease of remembering	Flexibility	Understandability
Minimum number of errors	Ease of use	Reliability	Adaptability
Readability	Increased productivity	Modifiability	Reusability
Ease of learning	Low-cost	Maintainability	Efficiency
Portability	Traceability of requirements	Fault tolerance	Backward-compatibility
Cost-effectiveness	Robustness	High-performance	

Typical Design Trade-offs
- Functionality v. Usability
- Cost v. Robustness
- Efficiency v. Portability
- Rapid development v. Functionality
- Cost v. Reusability
- Backward Compatibility v. Readability

Stakeholders views



Types of Coupling
Tight coupling means classes and objects are dependent on one another. In general, tight coupling is usually not good because it reduces the flexibility and re-usability of the code while Loose coupling means reducing the dependencies of a class that uses the different class directly.

Creational Patterns
Singleton Pattern
Problem:
We want to access to an unique object from any part of the system.
When to use it
You want to guarantee that only one instance exists in the system.
You want to access it from anywhere on the application
You need/want lazy initialization of your object

Factory Pattern

Abstract Factory

Builder
Problem:
We need to develop a program that creates hamburger combos
○ Drink
○ Meat
○ Topping
○ Fries

When to use it

- When we want to encapsulate the object creation
    - Not expose the details to the clients
- Create object by constructing one step at a time–

Object Pool

Structural Patterns

Façade Pattern

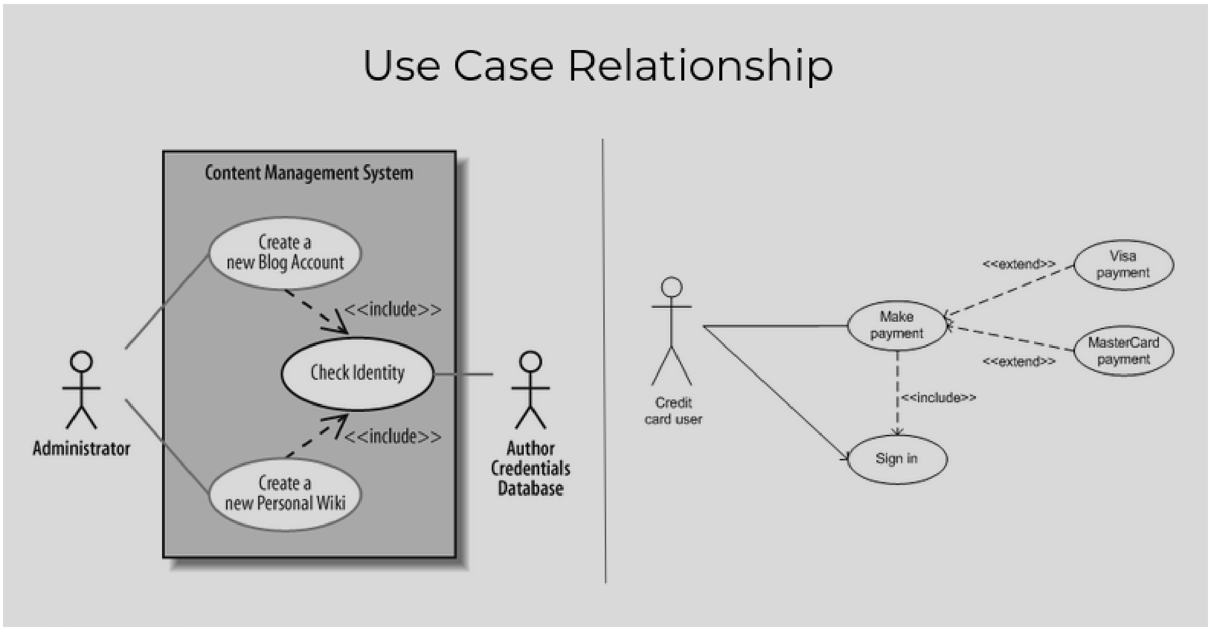
Adapter Pattern

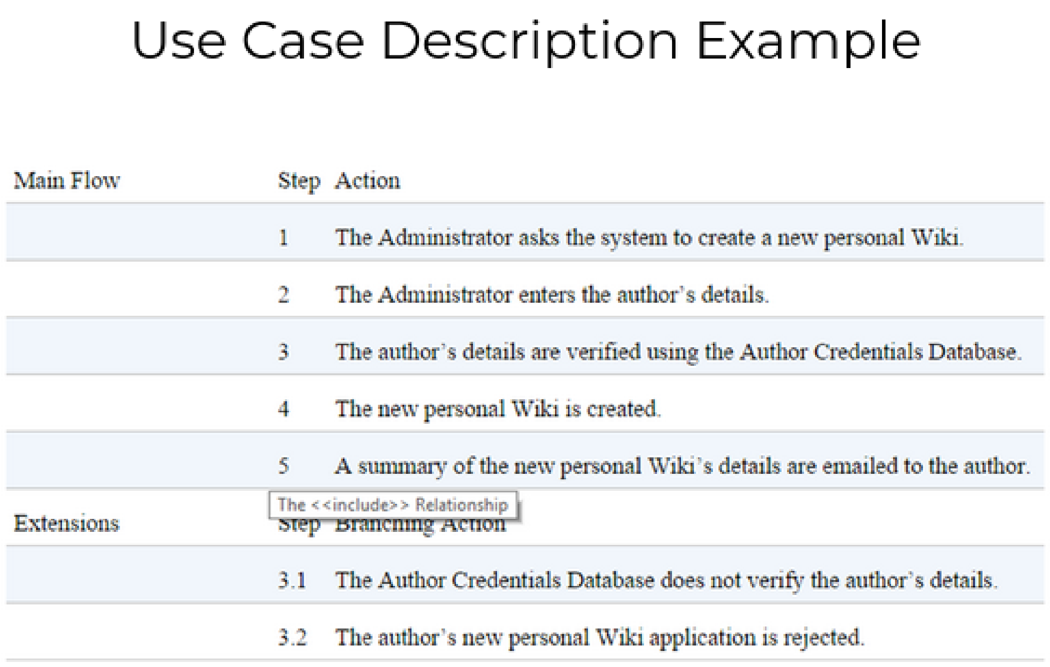
Behavioral Patterns

Strategy Pattern

Antipatterns

USE CASE DIAGRAM

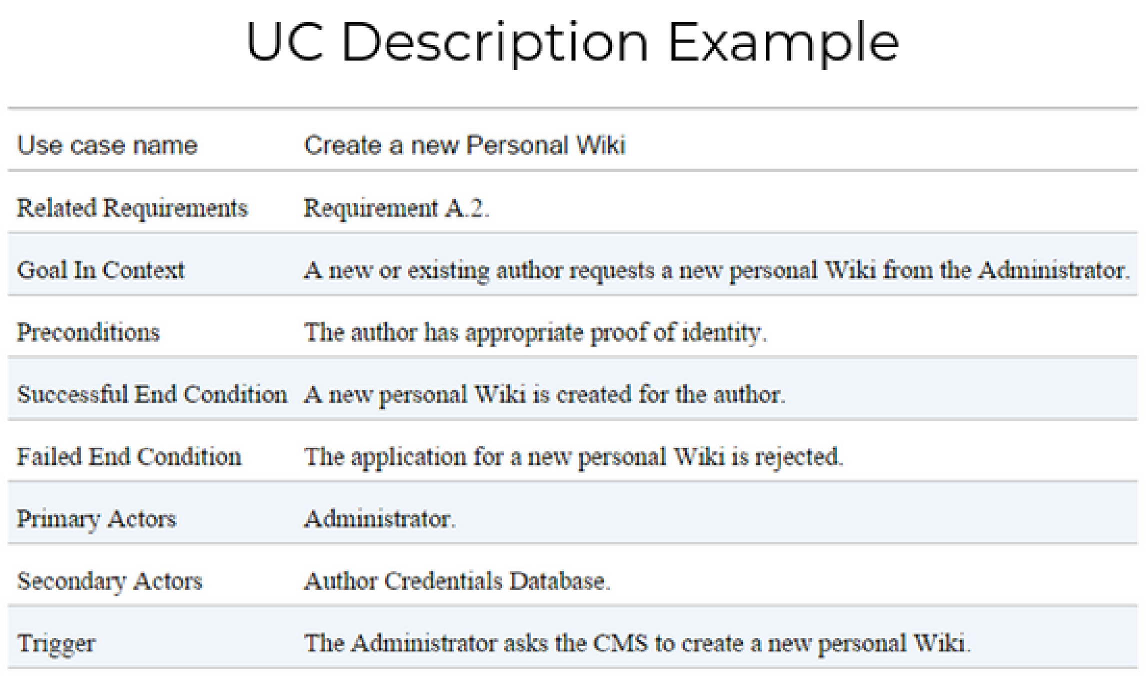




Use Case Description Example

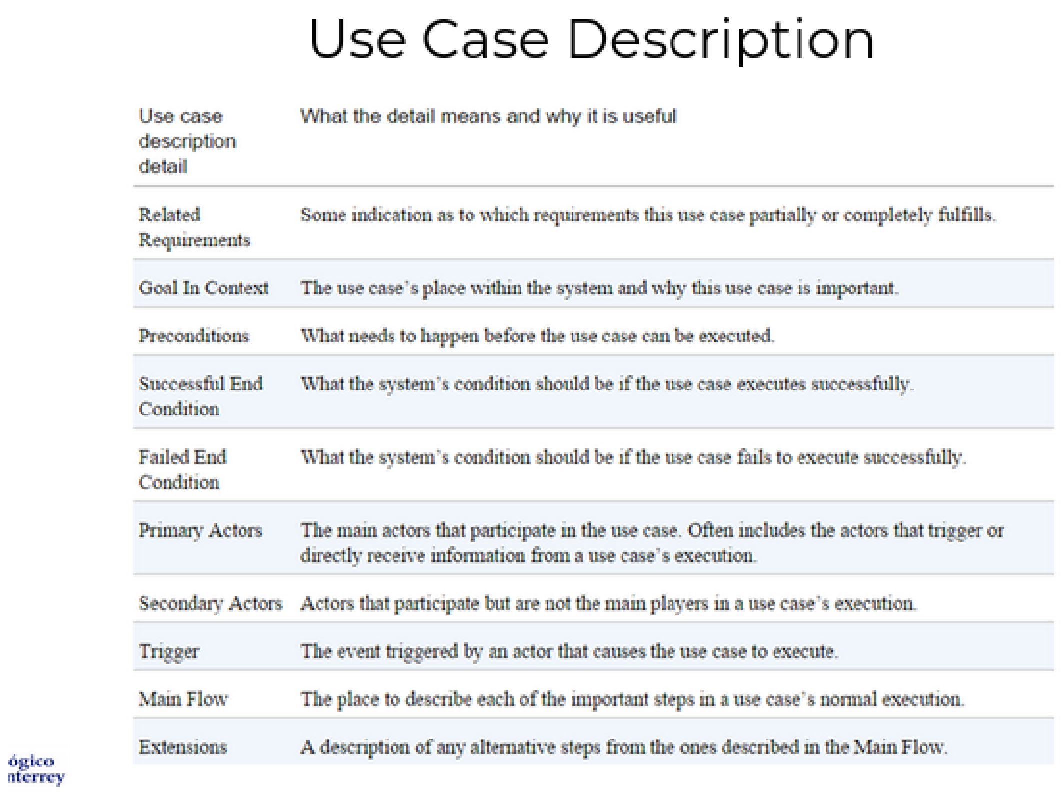
Main Flow	Step	Action
	1	The Administrator asks the system to create a new personal Wiki.
	2	The Administrator enters the author's details.
	3	The author's details are verified using the Author Credentials Database.
	4	The new personal Wiki is created.
	5	A summary of the new personal Wiki's details are emailed to the author.
Extensions	Step	Branching Action
	3.1	The Author Credentials Database does not verify the author's details.
	3.2	The author's new personal Wiki application is rejected.

The <<include>> Relationship



UC Description Example

Use case name	Create a new Personal Wiki
Related Requirements	Requirement A.2.
Goal In Context	A new or existing author requests a new personal Wiki from the Administrator.
Preconditions	The author has appropriate proof of identity.
Successful End Condition	A new personal Wiki is created for the author.
Failed End Condition	The application for a new personal Wiki is rejected.
Primary Actors	Administrator.
Secondary Actors	Author Credentials Database.
Trigger	The Administrator asks the CMS to create a new personal Wiki.



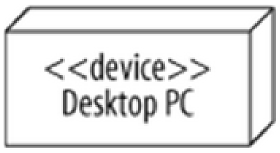
Use Case Description

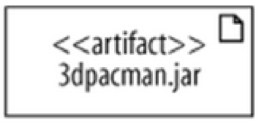
Use case description detail	What the detail means and why it is useful
Related Requirements	Some indication as to which requirements this use case partially or completely fulfills.
Goal In Context	The use case's place within the system and why this use case is important.
Preconditions	What needs to happen before the use case can be executed.
Successful End Condition	What the system's condition should be if the use case executes successfully.
Failed End Condition	What the system's condition should be if the use case fails to execute successfully.
Primary Actors	The main actors that participate in the use case. Often includes the actors that trigger or directly receive information from a use case's execution.
Secondary Actors	Actors that participate but are not the main players in a use case's execution.
Trigger	The event triggered by an actor that causes the use case to execute.
Main Flow	The place to describe each of the important steps in a use case's normal execution.
Extensions	A description of any alternative steps from the ones described in the Main Flow.

DEPLOYMENT DIAGRAM

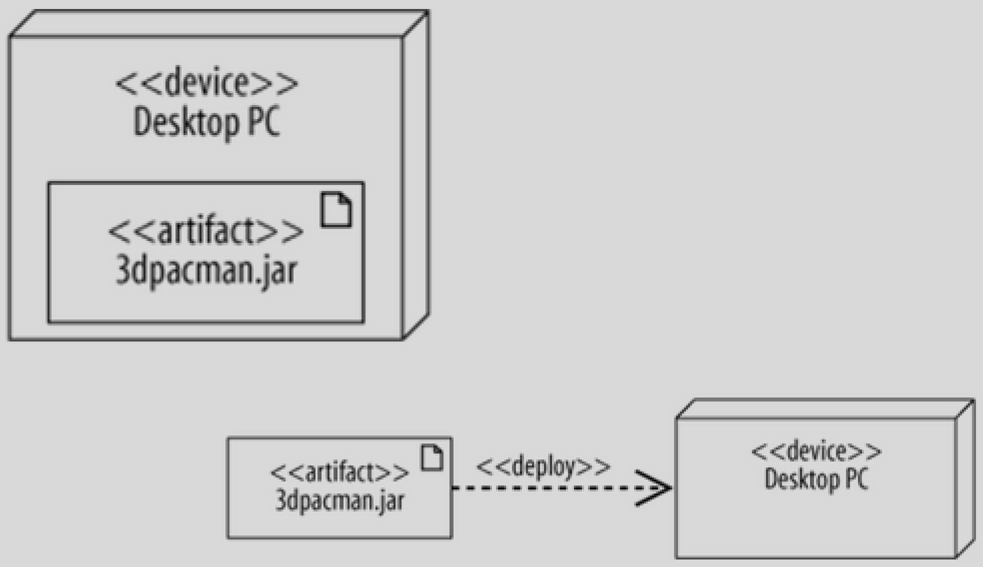
Deployment Diagram

- This diagram tell us how our component will be deployed
- Devices
  - Where the software will run
- Artifact
  - Our software

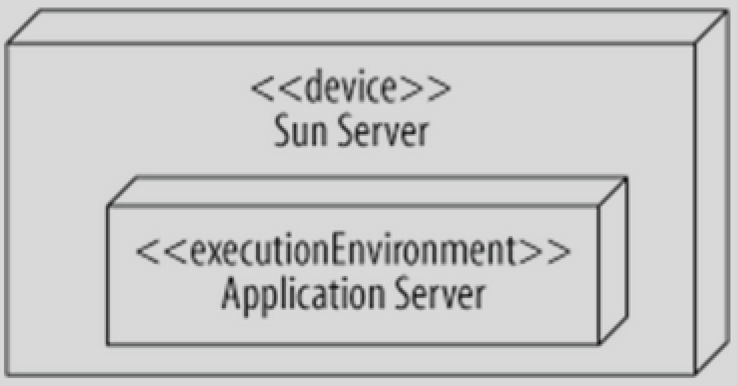


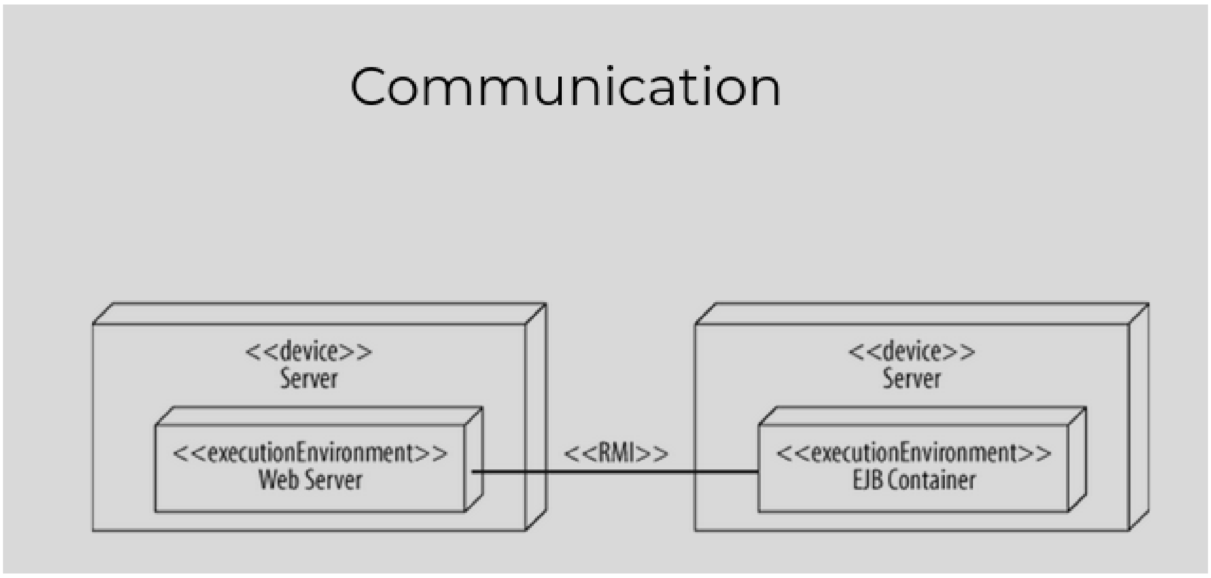


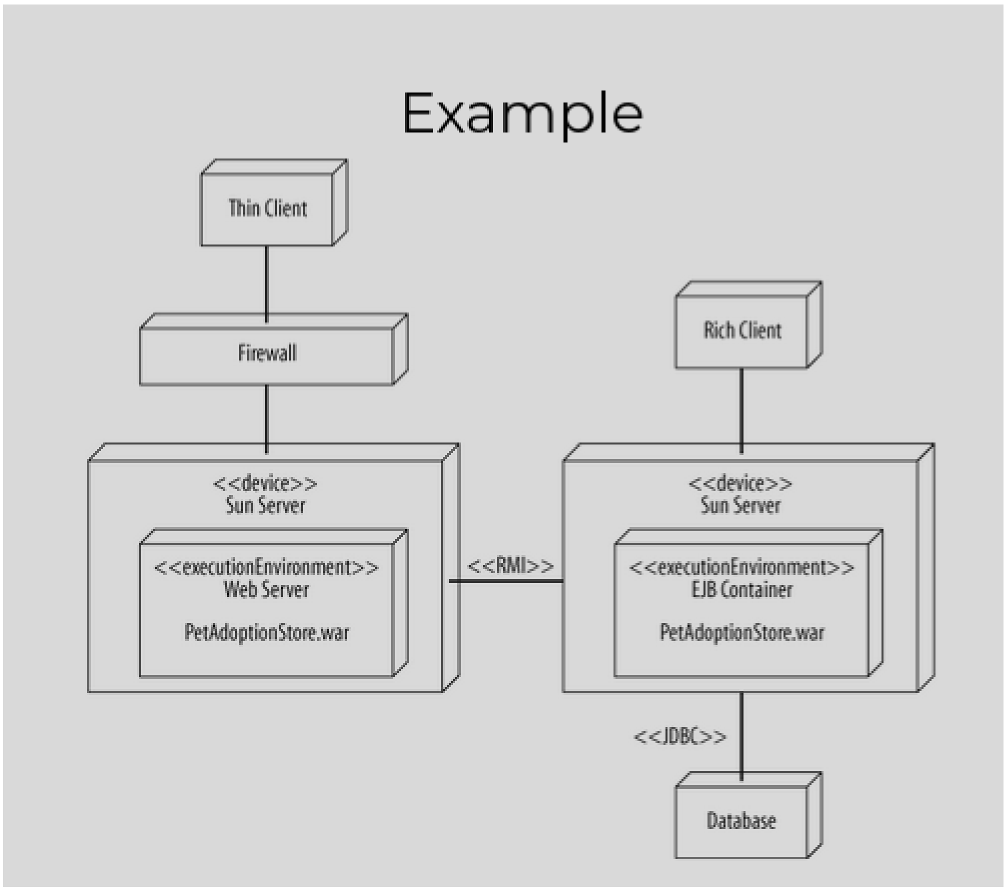
Simple Diagram



Nodes







Rúbrica

Deberán entregar lo siguiente:

1. Diseño de alto nivel (como el que vimos la clase pasada). [35 pts]
2. Diagrama de clases [20 pts]
3. Diagrama de despliegue [10 pts]
4. Decisiones de diseño que tomaron, justificadas con los patrones de diseño, principios (SOLID,YAGNI, etc) [15 pts]

5. Documentación de cómo su diseño implementa los Requerimientos No Funcionales [20 pts]
6. Documentación de casos de uso/requerimientos futuros (aquellos requerimientos que creen que se pueden implementar pero que no son parte de los ya requeridos, es decir, su visión para el proyecto)[10 pts (Extra)]
7. Entregar todo en un pdf en canvas.