Requirements, Architecture, Design

[DT-0540] Metodi di sviluppo agile

Daniele Di Pompeo

daniele.dipompeo@univaq.it

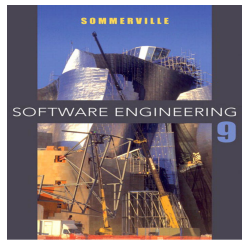
UNIVERSITÀ DEGLI STUDI DELL'AQUILA A.A. 2024-2025

Requirements

The requirements for a system are the descriptions of what the system should **do**, the services that it **provides**, and the **constraints** on its operation.

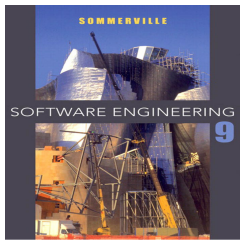
These requirements reflect the needs of customers for a system that serves a certain purpose such as controlling a device, placing an order, or finding information.

The process of finding out, analyzing, documenting and checking these services and constraints is called requirements engineering (RE).



Requirements

- **User requirements** are statements, in a natural language plus diagrams, of what services **the system is expected to provide** to system users and the constraints under which it must operate.
- **System requirements** are more detailed descriptions of the software system's functions, services, and operational constraints. The system requirements document (sometimes called a functional specification) **should define exactly what is to be implemented**.



Requirements: Functional

These are statements of services the system should provide, how the system should react to particular inputs, and how the system should behave in particular situations. In some cases, the functional requirements may also explicitly state what the system should not do.

[...] requirements for the MHC-PMS system, used to maintain information about patients receiving treatment for mental health problems: A user shall be able to search the appointments lists for all clinics. The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day. Each staff member using the system shall be uniquely identified by his or her eight-digit employee number.

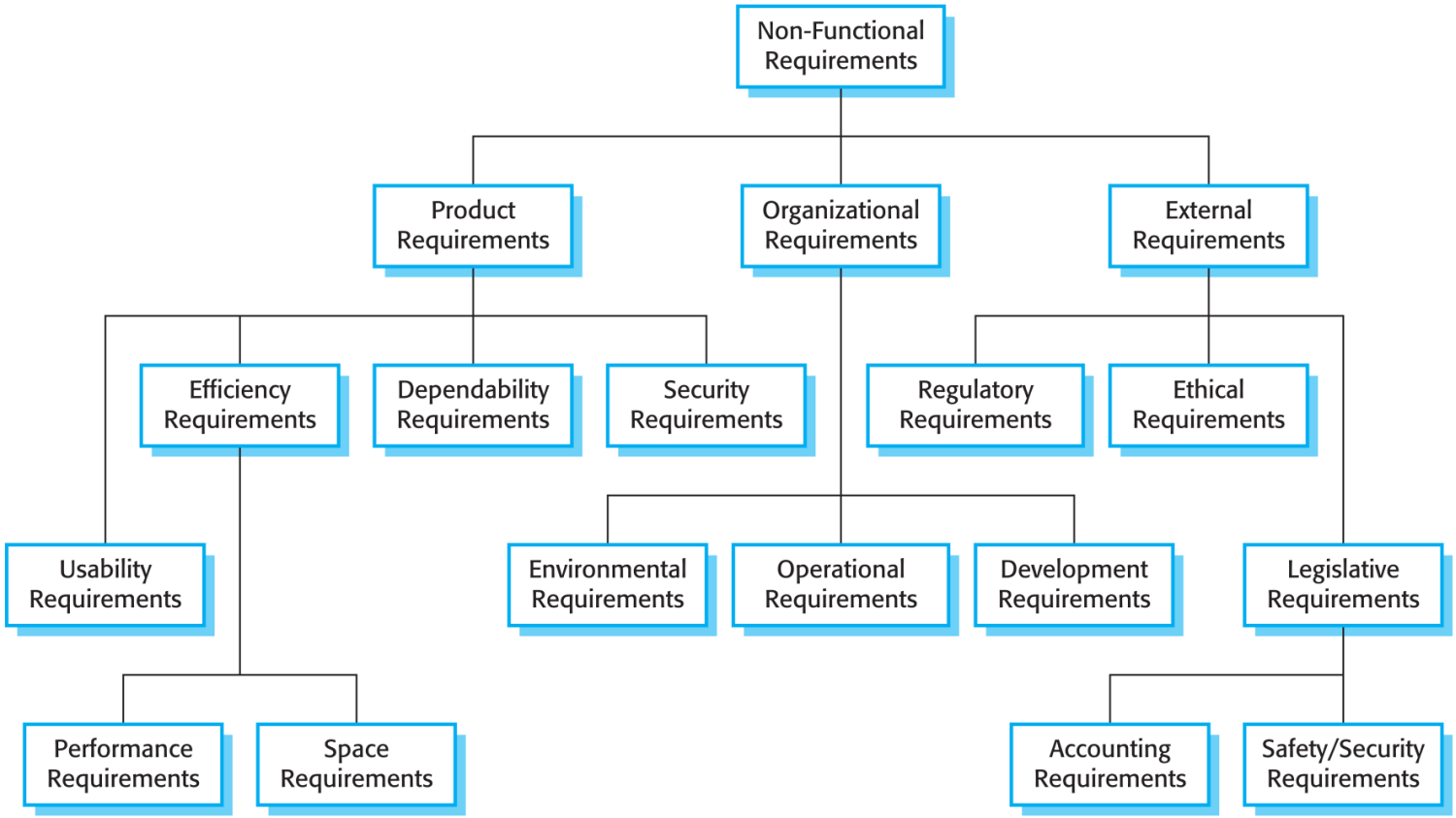
Requirements: Non-functional

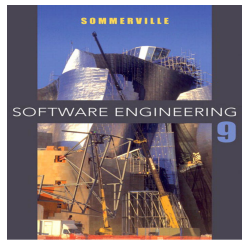
These are constraints on the services or functions offered by the system.

They include timing constraints, constraints on the development process, and constraints imposed by standards.

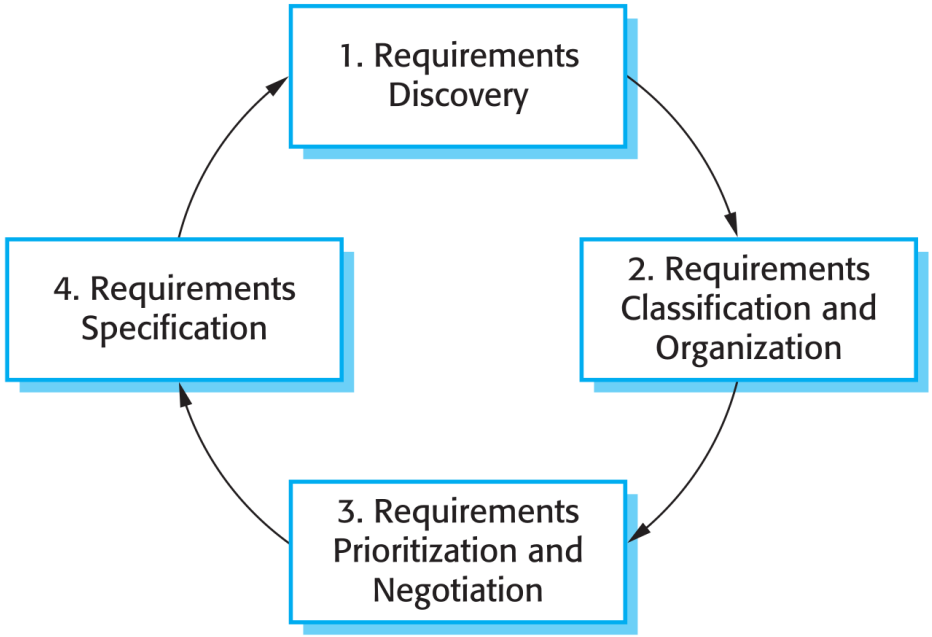
Non-functional requirements often apply to the system as a whole, rather than individual system features or services

Requirements: Types of non-functional requirements



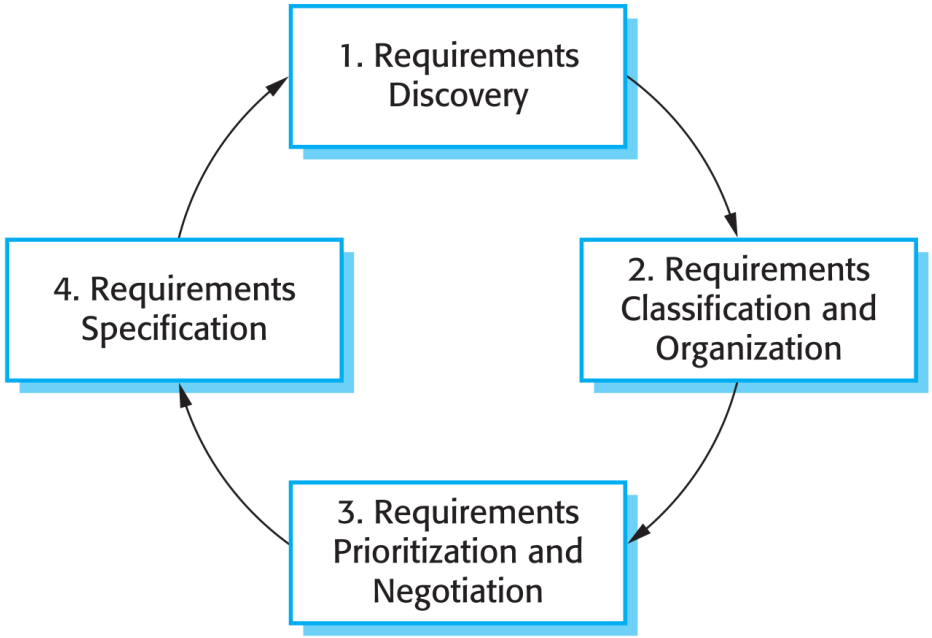


Requirements: Elicitation



Requirements Discovery This is the process of interacting with stakeholders of the system to discover their requirements. Domain requirements from stakeholders and documentation are also discovered during this activity.

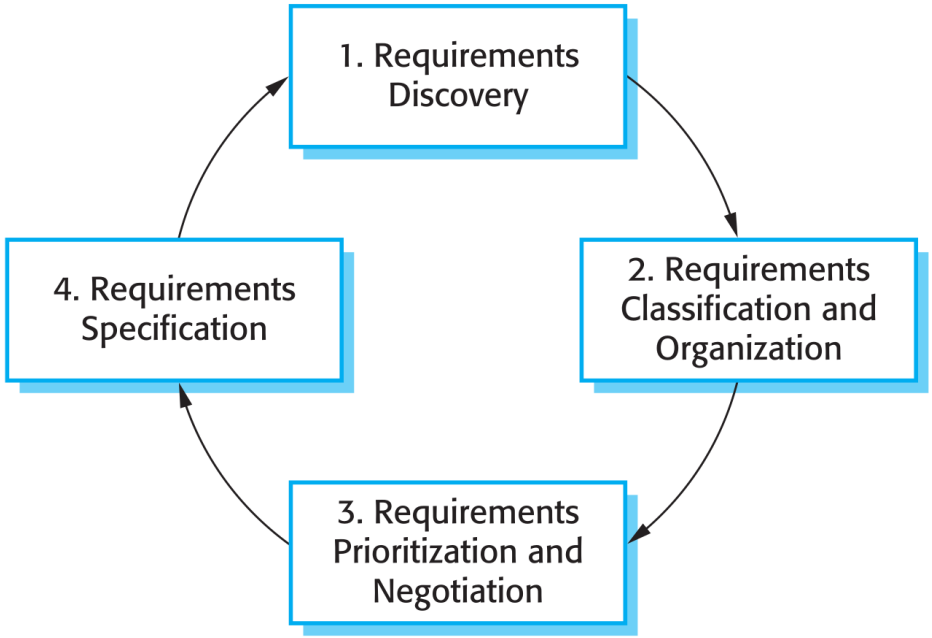
Requirements: Elicitation



Requirements Classification This activity takes the unstructured collection of requirements. The most common way of grouping requirements is to use a model of the system architecture to identify sub-systems and to associate requirements with each sub-system.

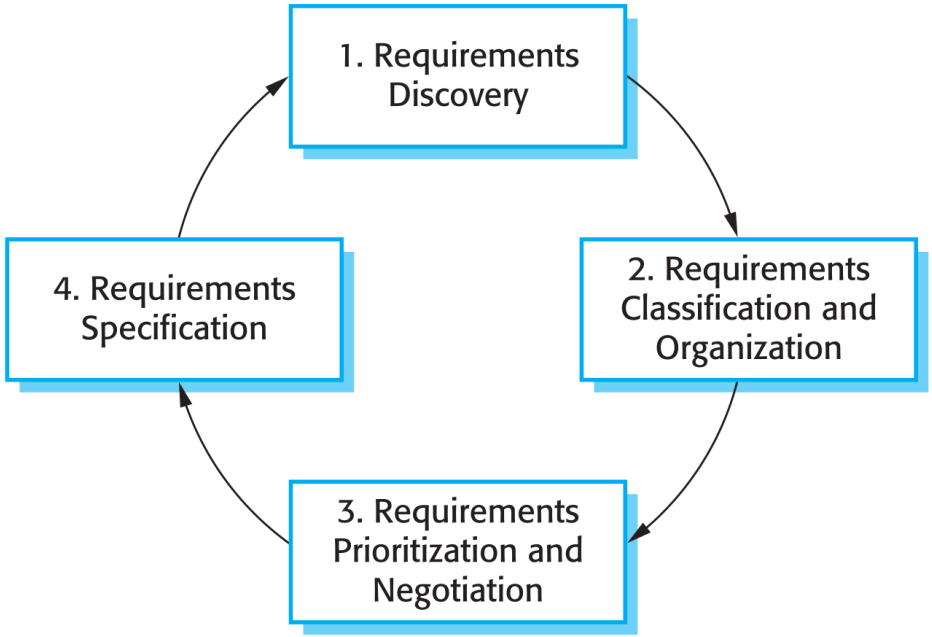
In practice, requirements engineering and architectural design cannot be completely separate activities.

Requirements: Elicitation



Requirements Prioritization
Inevitably, when multiple stakeholders are involved, requirements will conflict. This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiation. Usually, stakeholders have to meet to resolve differences and agree on compromise requirements.

Requirements: Elicitation



Requirements Specification The requirements are documented and input into the next round of the process.

What about domain requirements?

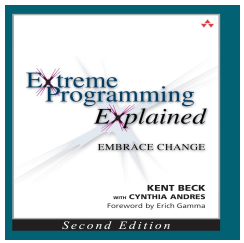
Some requirements elements describe **properties of the domain** in which the system will operate. Rules on accounts, deposits, etc. in a banking system Signal speed, frequency range, call pricing policy, etc. in phone software

"The speed of light is not an implementation decision"

Requirement Engineering in Agile

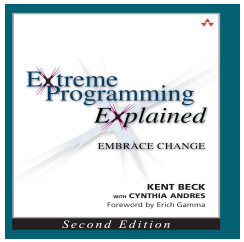
Agile criticism on upfront requirements

"Requirements gathering **isn't** a phase that produces a **static document**, but an activity producing detail, just before it is needed, throughout development."



Change criticism

"Software development is full of the waste of overproduction, [such as] requirements documents that rapidly grow obsolete."



Agile criticism on upfront requirements

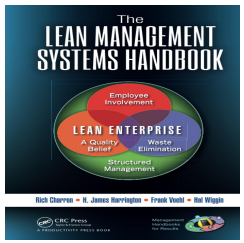
https://www.mountaingoatsoftware.com/blog/agile-design-intentional-yet-emergent

"Scrum projects do not have an upfront analysis or design phase; all work occurs within the repeated cycle of sprints."

Waste criticism

If your company writes reams of requirements documents (equivalent to inventory), you are operating with mass-production paradigms.

Think "lean" and you will find a better way.



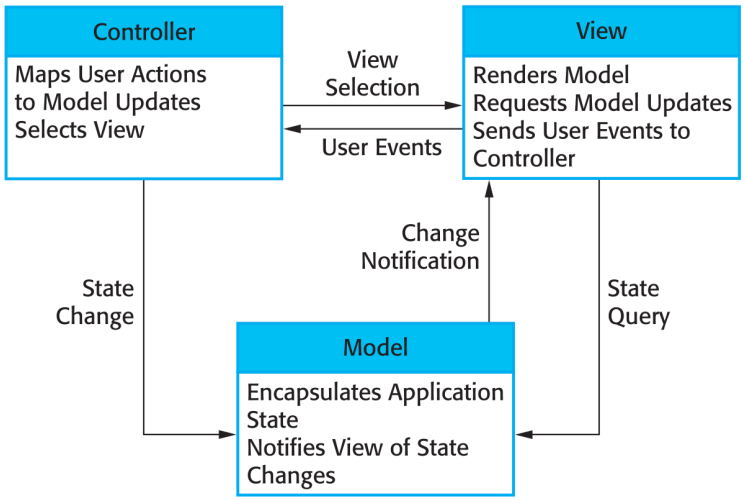
Software Architecture

Software Architecture

https://en.wikipedia.org/wiki/Software_architecture

"Software architecture refers to the **fundamental structures** of a software system and the discipline of creating such structures and systems. Each structure comprises software elements, relations among them, and properties of both elements and relations. The architecture of a software system is a **metaphor**, analogous to the architecture of a building. It functions as a **blueprint** for the system and the developing project, laying out the tasks necessary to be executed by the design teams"

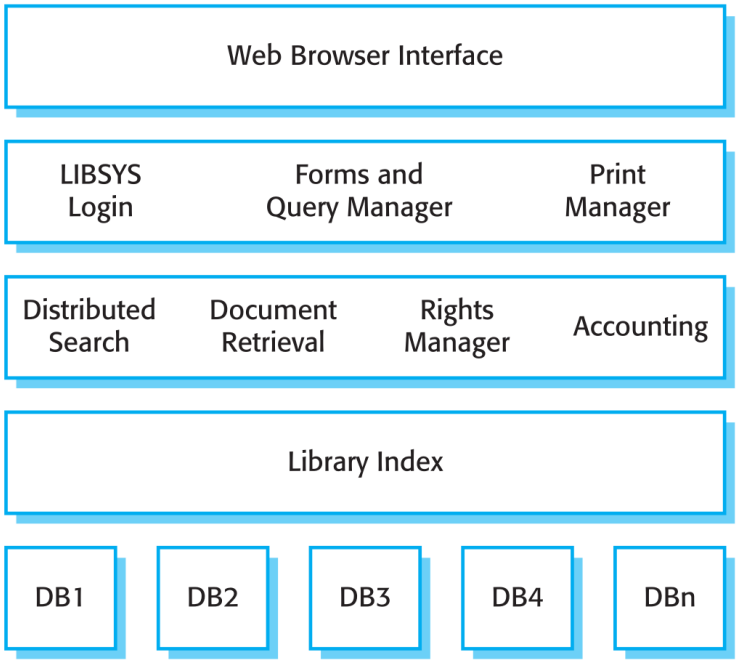
Software Architecture - MVC



+ Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways with changes made in one representation shown in all of them.

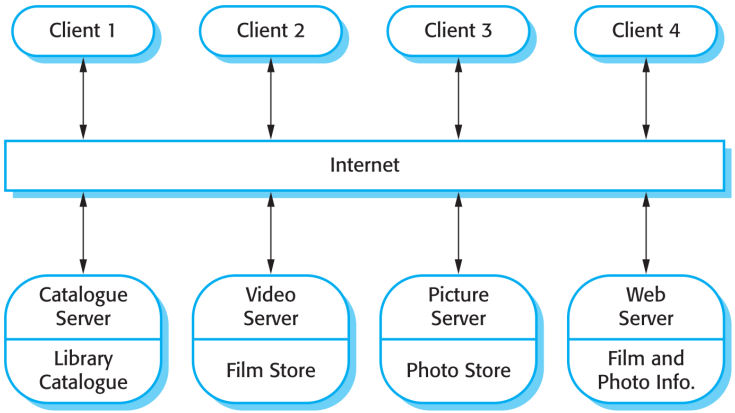
- Can involve additional code and code complexity when the data model and interactions

Software Architecture - Layered



Allows replacement of entire layers so long as the interface is maintained. Redundant facilities (e.g., authentication) can be provided in each layer to increase the dependability of the system.

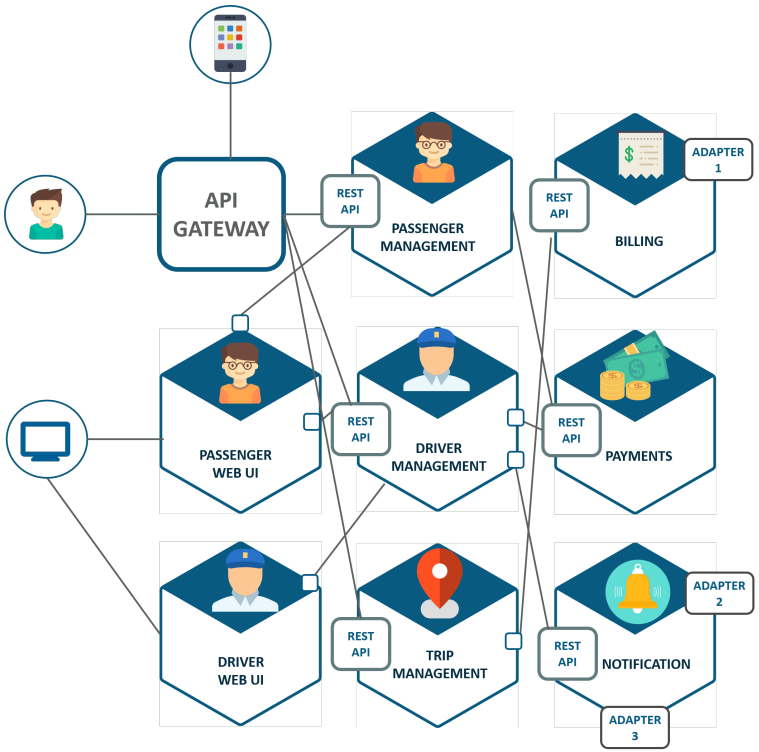
Software Architecture - Client Server



The principal advantage of this model is that servers can be distributed across a network. Each service is a **single point of failure** so susceptible to denial of service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the system.

Software Architecture - Microservices

https://www.edureka.co/blog/microservice-architecture/



Microservices allow a large application to be separated into smaller independent parts, with each part having its own realm of responsibility.
Each service is a single point of failure so susceptible to denial of service attacks or server failure.
Performance may be unpredictable because it depends on the network as well as the system.

Design Pattern

Design Patterns

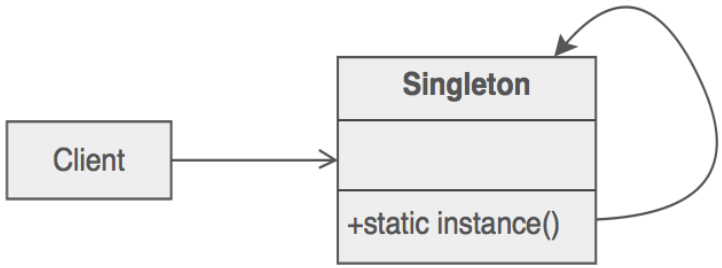
In software engineering, a design pattern is a general repeatable solution to a commonly occurring problem in software design.

- A design pattern isn't a finished design that can be transformed directly into code. **It is a description or template** for how to solve a problem that can be used in many different situations.
- Design patterns can **speed up** the development process by providing tested, proven development paradigms.
- Reusing design patterns **helps to prevent** subtle issues that can cause major problems and improves code readability for coders and architects familiar with the patterns.
- Design patterns provide **general solutions**, documented in a format that doesn't require specifics tied to a particular problem.

In addition, patterns allow developers to communicate using well-known, well understood names for software interactions.

Design vs Implementation - Singleton

https://sourcemaking.com/design_patterns/singleton

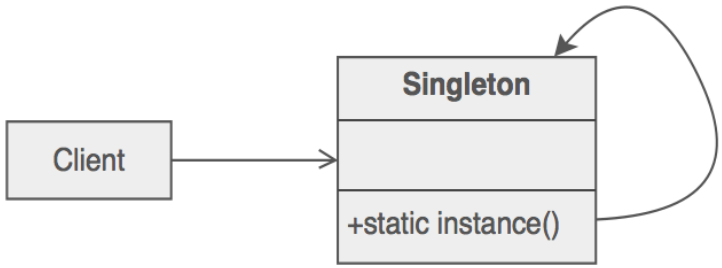


- The advantage of Singleton over global variables is that you are **absolutely sure** of the number of instances when you use Singleton, and, you can change your mind and manage any number of instances.
- The Singleton design pattern is one of the most inappropriately used patterns.

Singletons are intended to be used when a class must have **exactly** one instance, no more, no less.

Design vs Implementation - Singleton

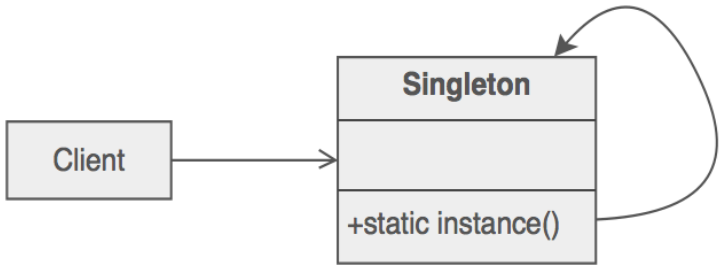
https://sourcemaking.com/design_patterns/singleton



When is Singleton unnecessary?

Design vs Implementation - Singleton

https://sourcemaking.com/design_patterns/singleton

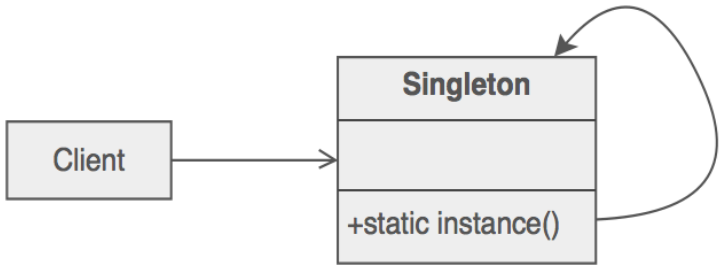


When is Singleton unnecessary?

Short answer: most of the time.

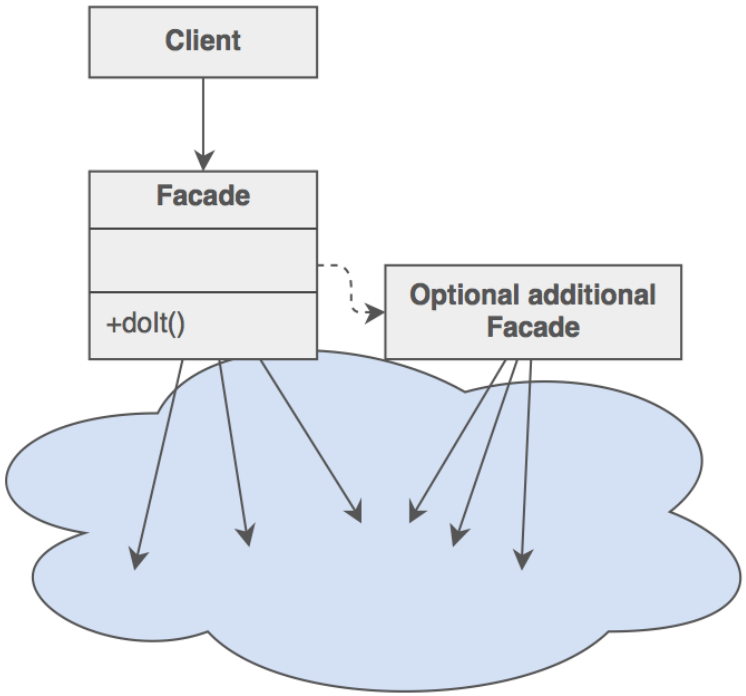
Design vs Implementation - Singleton

https://sourcemaking.com/design_patterns/singleton



The real problem with Singletons is that they give you such a good excuse not to think carefully about the appropriate visibility of an object. **Finding the right balance of exposure and protection for an object is critical for maintaining flexibility.**

Design vs Implementation - Façade



- The intent of Facade is to produce a simpler interface.
- Facade routinely wraps multiple objects
- Facade could front-end a single complex object

Design vs Implementation

https://www.mountaingoatsoftware.com/blog/agile-design-intentional-yet-emergent

The agile process favors an incremental, just-in-time approach to design.

- Scrum projects do not have an upfront analysis or design phase; all work occurs within the repeated cycle of sprints.
- This does not mean, however, that design on a Scrum project is not intentional.
- An intentional design process is one in which the design is guided through deliberate, conscious decision making.

Design vs Implementation

https://www.mountaingoatsoftware.com/blog/agile-design-intentional-yet-emergent

- On a Scrum project, design is both intentional and emergent.The design emerges because there is no up-front design phase (even though there are design activities during all sprints).
- Design is intentional because product backlog items are deliberately chosen with an eye toward pushing the design in different directions at different times."

Takeaways

- No big upfront
- Some upfront is sometimes needed
  - E.g. the problem of safety-critical domain
- Focused on Requirements
  - Software
  - Architecture
  - Design

Requirements, Architecture, Design

[DT-0540] Metodi di sviluppo agile

Daniele Di Pompeo

daniele.dipompeo@univaq.it

UNIVERSITÀ DEGLI STUDI DELL'AQUILA A.A. 2024-2025