Politecnico
di Milano


# Software Design Description and Design Principles

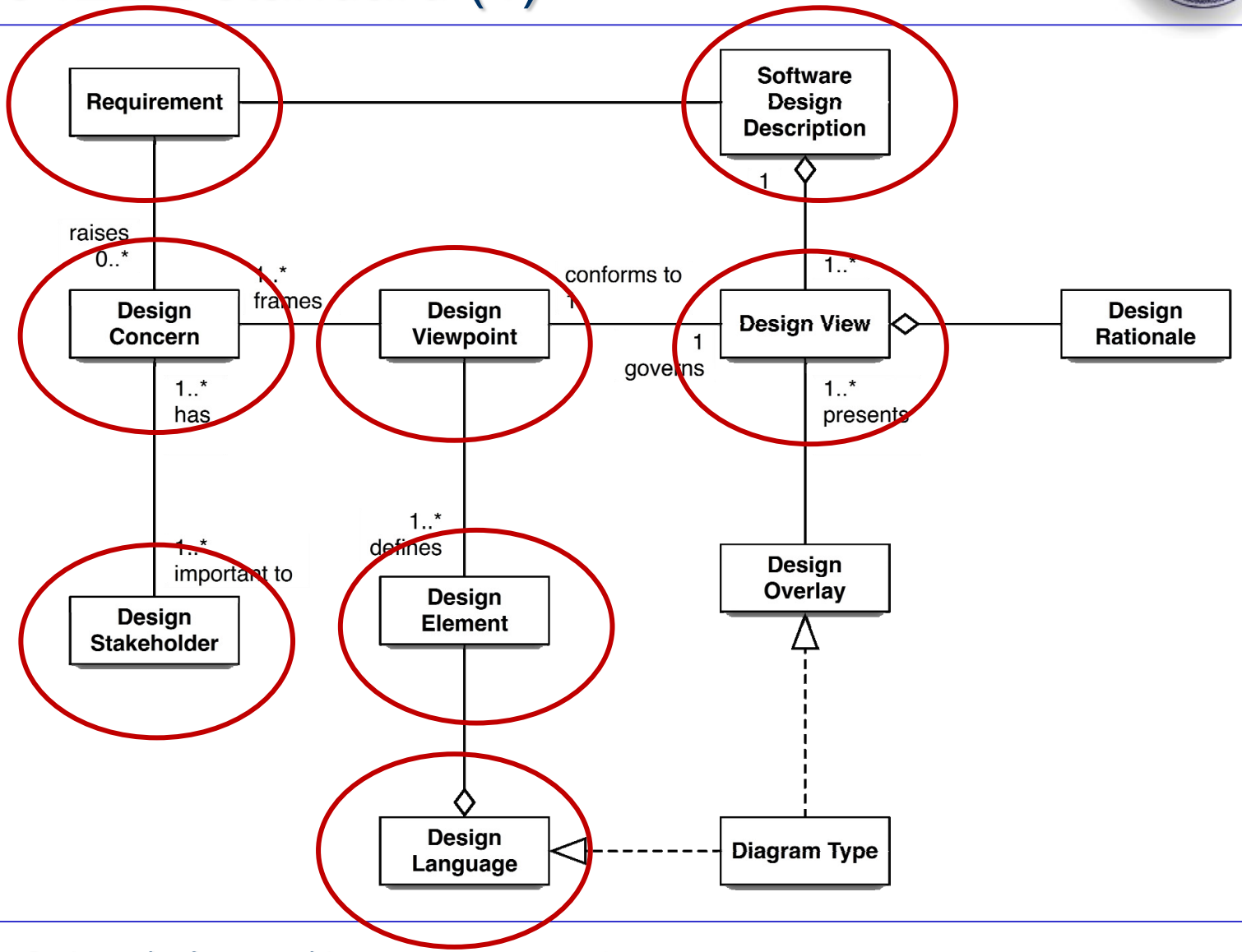
Slides used in the video available here
https://polimi365-my.sharepoint.com/:v:/g/personal/10143828_polimi_it/EaE5ix1izNxGiLLtzBsr7GABI6M_BqvtBPeH4qtuVMOnhQ?e=hydFes


# Software Design Description

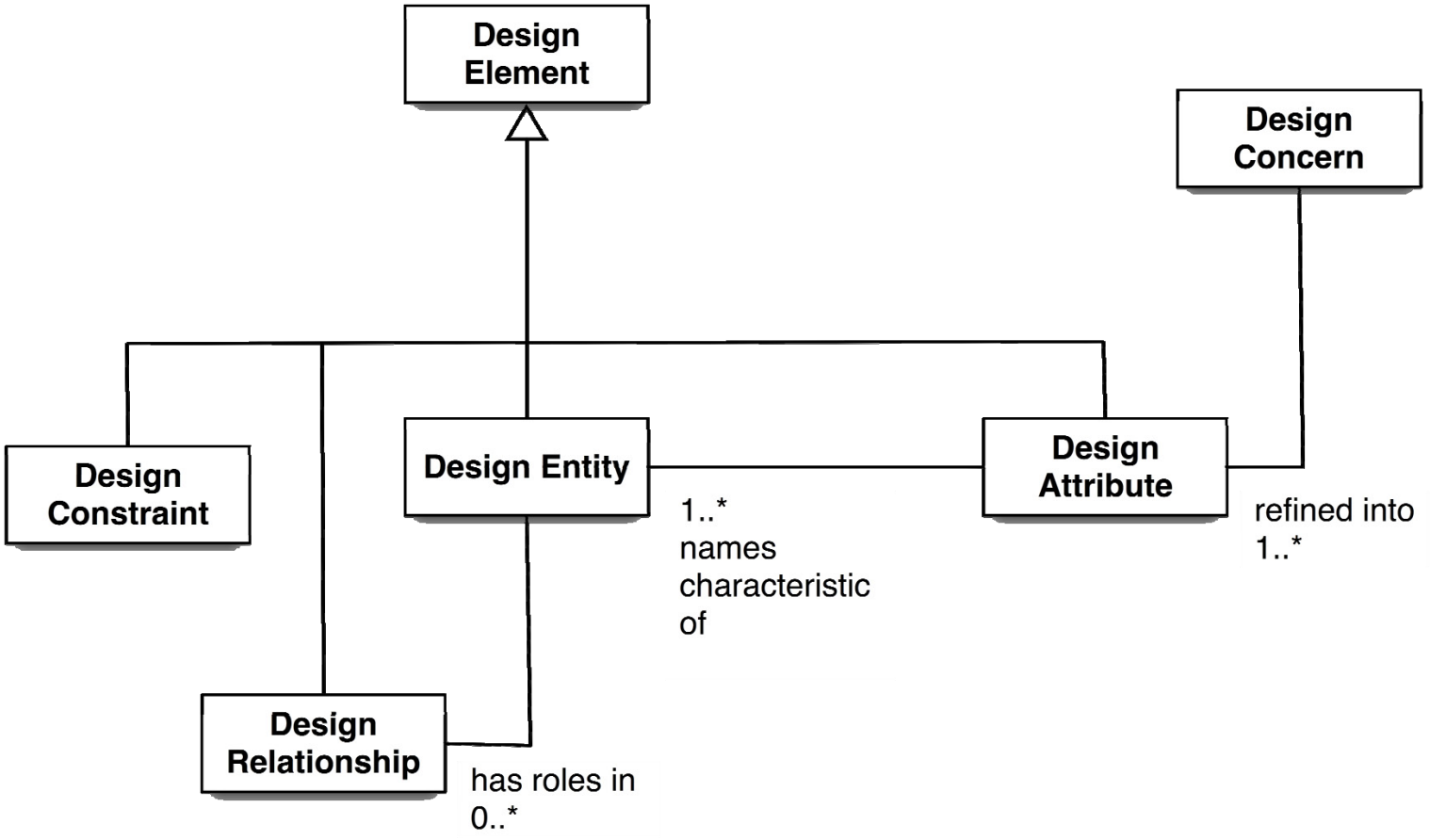
# Software Design Description (SDD)


- The IEEE Standard for Information Technology—Systems Design—Software Design Descriptions
- The IEEE Standard for Systems and software engineering — Architecture description
  - Specifies “the manner in which architectural descriptions of systems are organized and expressed”

# The key concepts of a SDD according to the IEEE standard (1)



# The key concepts of a SDD according to the IEEE standard (2)



# The required contents of an SDD according to the IEEE standard


- Identification of the SDD (date, authors, organization,…)
- Description of design stakeholders
- Description of design concerns
- Selected design viewpoints
- Design views
- Design overlays
- Design rationale

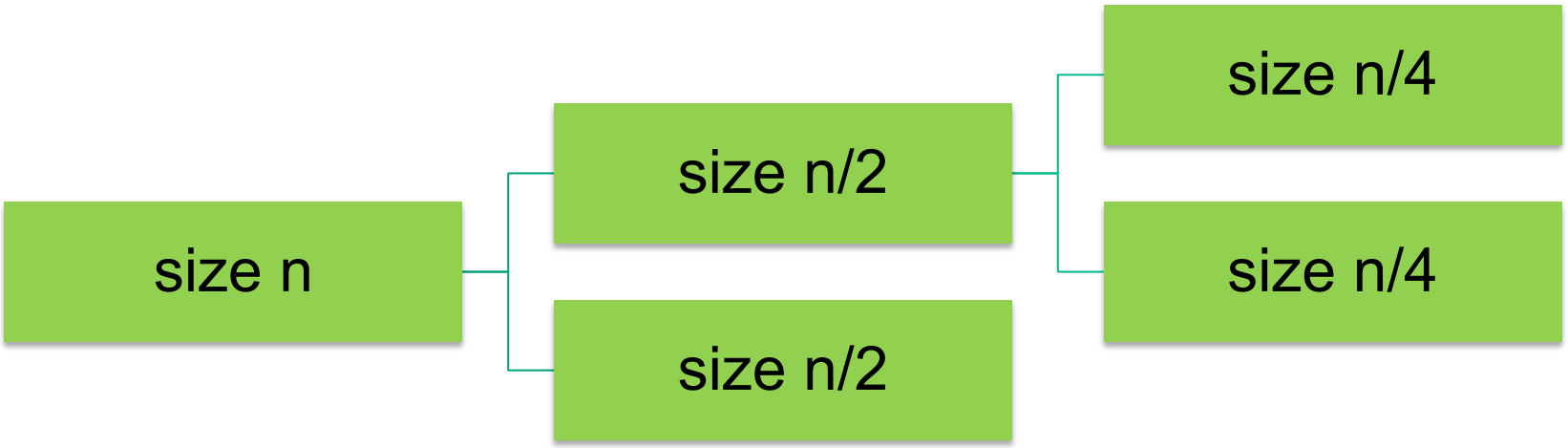

# Design principles

# Design Principles




- Design Principle 1: Divide and conquer
- Design Principle 2: Keep the level of abstraction as high as possible
- Design Principle 3: Increase cohesion where possible
- Design Principle 4: Reduce coupling where possible
- Design Principle 5: Design for reusability
- Design Principle 6: Reuse existing designs and code
- Design Principle 7: Design for flexibility
- Design Principle 8: Anticipate obsolescence
- Design Principle 9: Design for portability
- Design Principle 10: Design for testability
- Design Principle 11: Design defensively

# Design Principle 1: Divide & Conquer: the binary search example

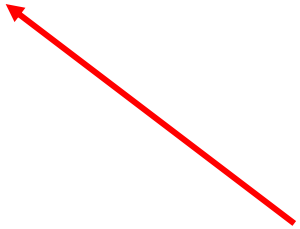


# Design Principle 2: Keep the level of abstraction as high as possible


- Ensure that your designs allow you to hide or defer consideration of details, thus reducing complexity
  - A good abstraction is said to provide *information hiding*
  - Abstractions allow you to understand the essence of a subsystem without having to know unnecessary details

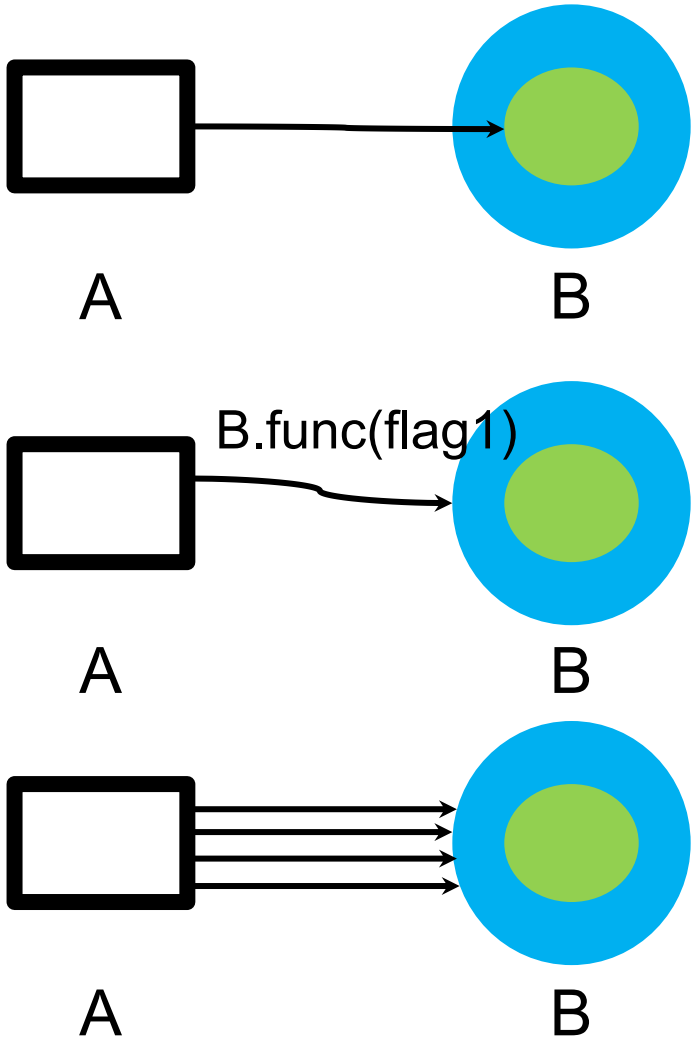
# Design Principle 3: Cohesion


- Example of a non-cohesive module
  - Class Utility {
    - ComputeAverageScore(Student s[])
    - ReduceImage(Image i)
  - }



Is this a cohesive class???

# Design Principle 4: Coupling
# What to avoid




A is accessing the data structure in B breaking encapsulation

→ *content coupling*

```
class b {
  func(flag f) {
    if(f == flag1) do this
    else if (f == flag2) do that
    else…
  }
}
```

→ *control coupling*

Do we really need so many messages from A to B?

# Design Principle 5: Design for reusability


- Design the various aspects of your system so that they can be used again in other contexts
  - Generalize your design as much as possible
  - Simplify your design as much as possible
  - Follow the preceding all other design principles
  - Design your system to be extensible

# Design Principle 6: Reuse existing designs and code


- Design with reuse is complementary to design for reusability
  - Take advantage of the investment you or others have made in reusable components
  - NOTE: Cloning should not be seen as a form of reuse

# Design Principle 7: Design for flexibility


- Actively anticipate changes that a design may have to undergo in the future, and prepare for them
  - Reduce coupling and increase cohesion
  - Create abstractions
  - Use reusable code and make code reusable
  - Do not hard-code anything

# Design Principle 8: Anticipate obsolescence


- Plan for changes in the technology or environment so the software will continue to run or can be easily changed
  - Do not rush using early releases of technology
  - If possible
    - Avoid using software libraries that are specific to particular environments
    - Avoid using undocumented features or little-used features of software libraries
    - Avoid using software or special hardware from companies that are less likely to provide long-term support
    - Use standard languages and technologies that are supported by multiple vendors

# Design Principle 9: Design for Portability


- Have the software run on as many platforms as possible
  - Avoid, if possible, the use of facilities that are specific to one particular environment
    - E.g. a library only available in Microsoft Windows

# Design Principle 10: Design for Testability


- Take steps to make testing easier
  - Design a program to automatically test the software
    - Ensure that all the functionality of the code can by driven by an external program, bypassing a graphical user interface
  - In Java, you can
    - create a main() method in each class in order to exercise the other methods
    - Use Junit and related approaches to build an automated testing framework for your system

# Design Principle 11: Design defensively


- Be careful when you trust how others will try to use a component you are designing
  - Handle all cases where other code might attempt to use your component inappropriately
  - Check that all of the inputs to your component are valid: the preconditions
    - Unfortunately, over-zealous defensive design can result in unnecessarily repetitive checking