


**SWE30003**  
**Software Architectures and Design**

Lecture 4  
The Role of Abstraction in Software

1



## Logistical matters

- Weekly submissions – A & Q
  - ☐ Week 2: 362 and 341 out of 459;
  - ☐ Week 3: 397 and 375 out of 459;
  - ☐ Week 4:
  - ☐ Week 5:
  - ☐ Note that this is **a hurdle requirement**
  - ☐ No late submission
- Assignment 1: should be well on the way ...

2

## Question to Answer from Week 3



*Why can quality attributes, in general, **not** be considered in isolation? Give two examples of quality attributes that impact other quality attributes for a system under consideration.*

3

3

## Outline



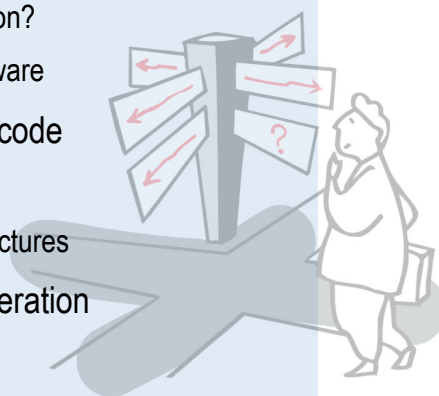
### ■ Abstractions

- ☐ What is an abstraction?
- ☐ Abstractions in Software

### ■ Abstractions beyond code

- ☐ Domain Models
- ☐ Patterns and Architectures

### ■ Questions for Consideration



4

4

## Principal References



- Len Bass, Paul Clements, and Rick Kazman, *Software Architecture in Practice* (4<sup>th</sup> or 3<sup>rd</sup> Edition), Addison-Wesley, 2021, Chapter 1
- Guy Steele, *Growing a Language*, Higher-Order and Symbolic Computation, Number 12, 1999 (available from Canvas)

5

5

## Complexity in Software Systems



*“It is not possible to remove the inherent complexity of large-scale software systems, but it is possible to use adequate development techniques to master complexity!”*

— Anonymous

- Moving towards solution of the problem ... Analysis & Design

6

6

## Design vs. Design



*“Design is the creative process of transforming the problem into a solution. The description of the solution is also called the design.”*

**Activity & Artefact** — Shari Lawrence Pfleeger, 1998

*“The design of a system determines a set of components and inter-component interfaces that satisfy a specified set of requirements.”*

— DeMarco, 1982

**Artefact**

7

7

## Design (cont.)



*“A design is a plan how to build a thing. To design is to build a thing in one’s mind but not yet in the real world — or, better yet, to plan how the real thing can be built.”*

— Guy Steele, OOPSLA 1998

8

8

## Software Design Methods



Many design methods have been defined:

- Modular:
  - ☐ focus on functions and procedures
- Data-oriented:
  - ☐ focus on external data sources
- Event-oriented:
  - ☐ how events change states
- Middle-out:
  - ☐ Jackson System Development (JSD)
- Object-oriented:
  - ☐ identify classes of objects and their inter-relationships
- Process-oriented:
  - ☐ focus on communication between independent processing entities

... abstractions

9

9

Common to all software design  
methods: support for -

*abstraction*,

decomposition & composition!

10

10

## Abstraction

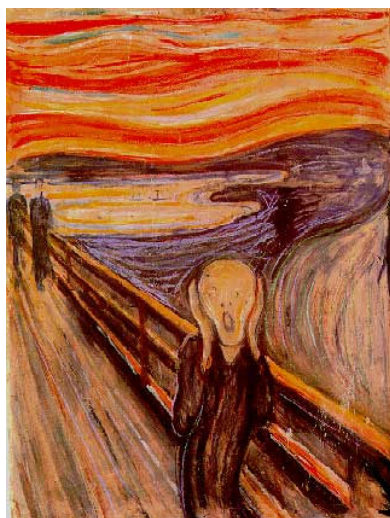


- A *concept* or *idea* not associated with any specific instance,
- Leaving out irrelevant detail, *highlighting important elements*,
- Mathematically, an abstraction is a *mapping*.

11

11

## Abstraction - Example



12

12

# Abstraction - Example (cont.)



13

13

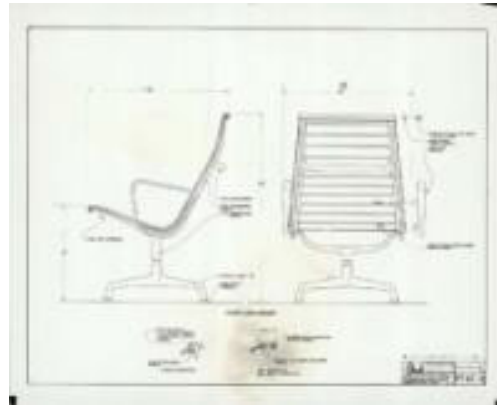
# Abstraction - Example (cont.)



14

14

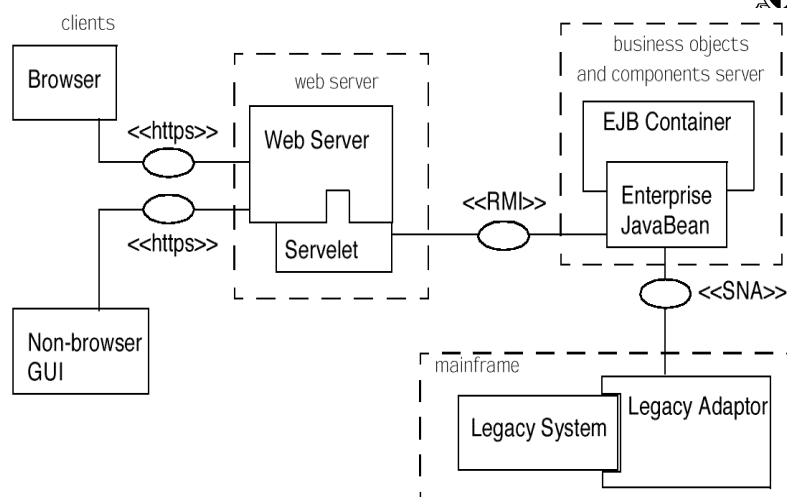
## Abstraction - Example (cont.)



15

15

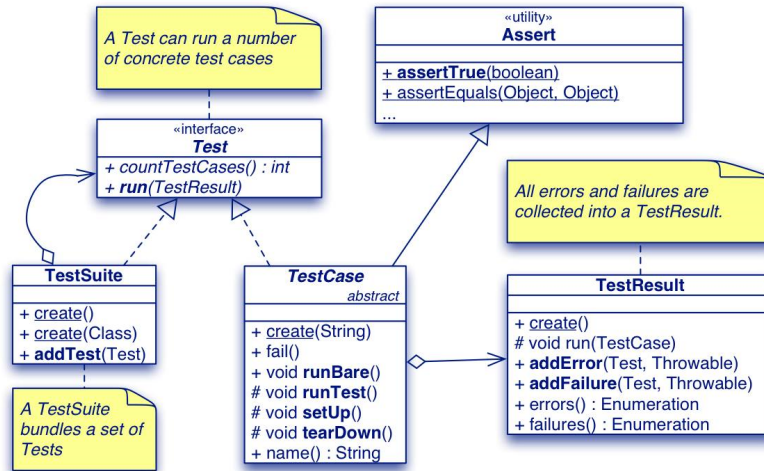
## Abstraction - Example (cont.)

*4-Tier Architecture*

16

16

## Abstraction - Example (cont.)



17

17

## Abstraction - Example (cont.)



```

function sort(array)
    var list less, greater
    if length(array) ≤ 1
        // array of zero or one elements is already sorted
        return array
    // select and remove a pivot value pivot from array
    for each x in array
        if x ≤ pivot then append x to less
        else append x to greater
    return concatenate(sort(less), pivot, sort(greater))
  
```

18

18

## Abstraction - Example (cont.)



```
deferred class STACK [G]
feature -- Element change
  push (v : G) is      -- Push v onto top
    deferred
    ensure
      item_pushed: top = v
      size_increased: size = old size + 1
    end;
  pop is              -- Remove top
    deferred
    require
      not_empty: size > 0
    ensure
      item_removed: size = old size - 1
    end;
  top : G is          -- Get top element
    deferred
    require
      not_empty: size > 0
    ensure
      size_invariant: size = old size
    end;
end -- class Stack
```

19

19

## General View of Abstraction



### Real World

### Abstract World

☞ Phenomenon with messy details

☞ Model with “simple” properties

☞ *Facts and laws governing the real world*


☞ *Representation subject to “formal” manipulation*



Abstraction: *mapping from real world to abstract world*

20

20




---

Break

21

21



---

*But what kinds of abstractions  
do exist in software?*

22

22

## Fundamental Programming Abstractions



- Instruction
- Sequence
- Condition
- Repetition
- Abstraction:
  - Data abstraction
  - Functional abstraction

*Most (if not all...) programming languages are based on these fundamental abstractions!*

23

23

## “Abstraction Stack” in Software



- Subsystems
- Components
- Packages
- Interfaces
- Objects
- Abstract Data Types (e.g, list, tree)
- Functional abstractions (e.g., procedures, methods)
- Primitive data types (e.g., int, bool, char)
- Simple arithmetic expressions and control structures
- Macros
- Symbolic names (for instructions and memory cells)
- Machine instructions, memory cells ...

*Directly or indirectly, they all related to program code!*

24

24



*But what about if we abstract  
from the code/program itself?*

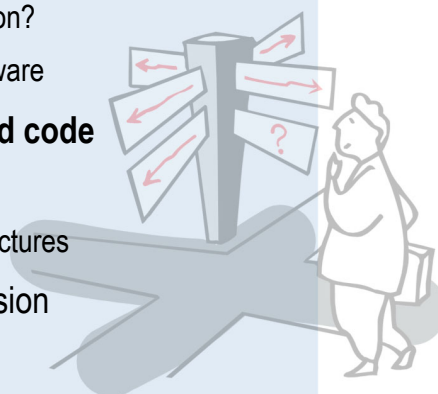
25

25

## Outline



- Abstractions
  - ☐ What is an abstraction?
  - ☐ Abstractions in Software
- **Abstractions beyond code**
  - ☐ Domain Models
  - ☐ Patterns and Architectures
- Questions for Discussion



26

26

## Models are Abstractions



- A model is a *simplification of reality*.
- (many) models are created for better understanding of the problem, domain or system to build.
- A model can capture and represent,
  - ☐ Structure (static model)
  - ☐ Behavior (dynamic model)
- Examples:
  - ☐ Architectural blue-print of a house
  - ☐ Hand-drawn screen



27

27

## Domain Models



- A *domain model* captures the domain entities (or concepts) relevant to a given problem specification
- A *domain vocabulary* generally defines the meaning (or interpretation) of domain entities
- The relationships between domain entities define an *abstraction over the problem domain*.

28

28

## Recap - Domain Model Development



### ■ Process

Step 1. Perform a *textual analysis* of the problem specification for understanding the problem domain

□ Circle all *nouns* and underline the *verbs*

Step 2. Write down relevant *domain entities*. They are normally (some of) the nouns defined above.

Step 3. Revisit the specification to extract the *associations*

Step 4. Record associations with the corresponding domain entities.

☞ *Note: domain modeling is not design; it is merely a way to understand the abstract concepts relevant to the problem!*

29

29

## “Exercise for the Reader”



*Develop a domain model for a chess game that allows a human player to play a game of chess against a computer.*

☞ We may come back to this problem later in the semester...

30

30

## Software Design



*The activity of **software design** can be seen as **translating** and **refining** a (problem) domain model into abstractions that can be directly mapped to an implementation in a programming language.*

- ☞ In general some “intermediate” steps will become necessary... solutions at different levels of abstraction
- ☞ The “closer” the domain abstractions are to the target language, the easier this process will become...

31

31

## Design Patterns are Abstractions



Design patterns document standard solutions to common (software) design problems:

*“Each pattern systematically names, explains, and evaluates an important and recurring design in object-oriented systems. Our goal is to capture design experience in a form that people can use effectively.”*

— Gamma et al., Design Patterns, 1995

*We will cover Design Patterns in more detail in Lecture 7*

32

32

## Software Architecture



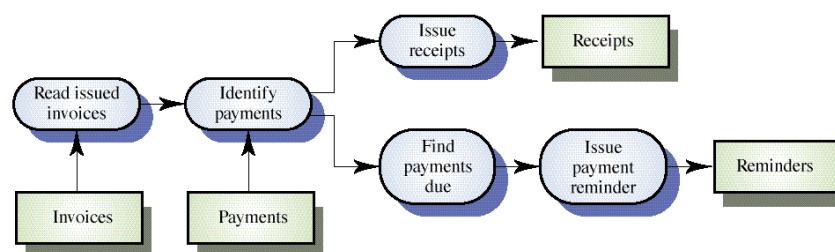
*“In most successful software projects, the expert developers working on that project have a **shared understanding** of the system to be implemented. This shared understanding is called **architecture** and includes how the system is divided into processing elements (i.e. **components**) and their **interaction** through interfaces. These components are usually composed of smaller components, but the architecture only includes the ones that are understood by all developers.”*

— Ralph Johnson, 2003

33

33

## Example - Invoice Processing System



©Ian Sommerville 1995

34

34

## “Growing a Language”



*“A design is a plan how to build a thing. To design is to build a thing in one’s mind but not yet in the real world — or, better yet, to plan how the real thing can be built.”*

— Guy Steele, OOPSLA 1998

35

35

## “Growing a Language” (cont.)



- Keynote address at OOPSLA 1998 (Vancouver)
- Main message: the importance of an appropriate *vocabulary* in communicating software design.
- Lessons to be learnt: software designers need
  - a large vocabulary of design elements,
  - based on a small set of well-founded and well-understood principles,
  - experience when and how to use the vocabulary.
- ☞ Abstraction is an important aspect of such a vocabulary!

36

36

## Closing Remarks



- Abstraction and modeling are used at every phase of the software development process.
- The level, benefit, and value of a particular abstraction depend on its purpose.
- Abstraction is key to mastering complexity in software systems.
- Abstraction is key to decomposition.

37

37

## Question for Review



1. Explain the relation between words, vocabulary and language, in the context of computer programming.
2. Explain why designing and implementing a small language that has limited functionality, but has the potential to grow, is a better alternative to developing a larger, more expressive language in the first place.
3. What is the difference between abstraction and modularity in a system and how can they be used to enhance the system?
4. Explain the analogy of Cathedrals, shopping malls and bazaars in relation to programming languages.

38

38

## Question to Answer - Week 4 (for week 5)



A good programmer in these times does not just write programs, *he/she builds a working vocabulary*. Provide examples and explain this statement.

39

39

## Required Reading Lecture 5



- Rebecca Wirfs-Brock and Alan McKean, *Object Design: Roles, Responsibilities, and Collaborations*, Addison-Wesley, 2003, Chapter 1 (available from Google Books)
- Timothy A. Budd, *An Introduction to Object-Oriented Programming* (3<sup>rd</sup> Edition), Addison-Wesley, 2002, Chapter 3 (available from Canvas)

40

40