SSW 322: Software Engineering Design VI

Introduction to Software Engineering Design
2018 Spring

Prof. Lu Xiao
lxiao6@stevens.edu
Babbio 513
Office Hour: Monday/Wednesday 1 to 3 pm
Software Engineering
School of Systems and Enterprises

Today’s topics

- An overview of engineering design
- Software engineering design
    - Design life cycle
    - User centered design
    - Domain driven design
    - Design goals
    - Design fundamental principles

Speaking of Design…

An Overview of Engineering Design

Who: Engineering Designers, i.e. design and development engineers

Does What: apply their scientific and engineering knowledge to the solution of technical problems.

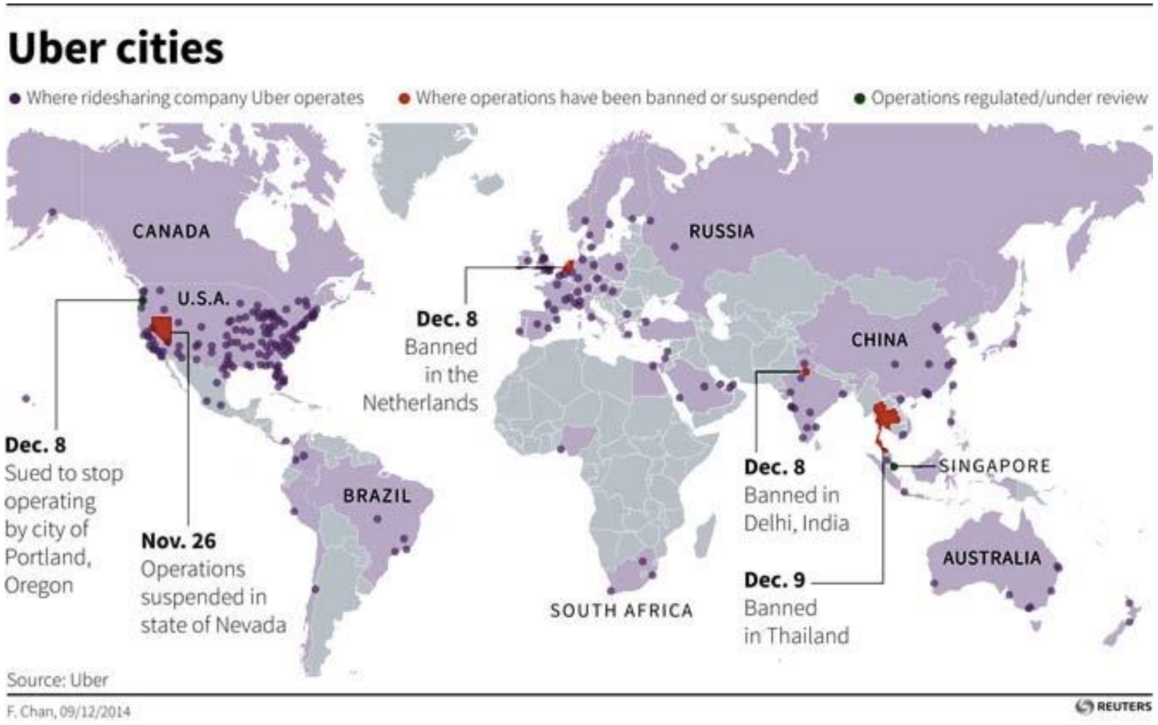
An Overview of Engineering Design

Design is an ***interesting*** engineering activity that:

- affects almost all areas of human life
- uses the laws and insights of science
- builds upon special experience
- provides the prerequisites for the physical realization of solution ideas
- requires professional integrity and responsibility

Affects all Areas of Human Life

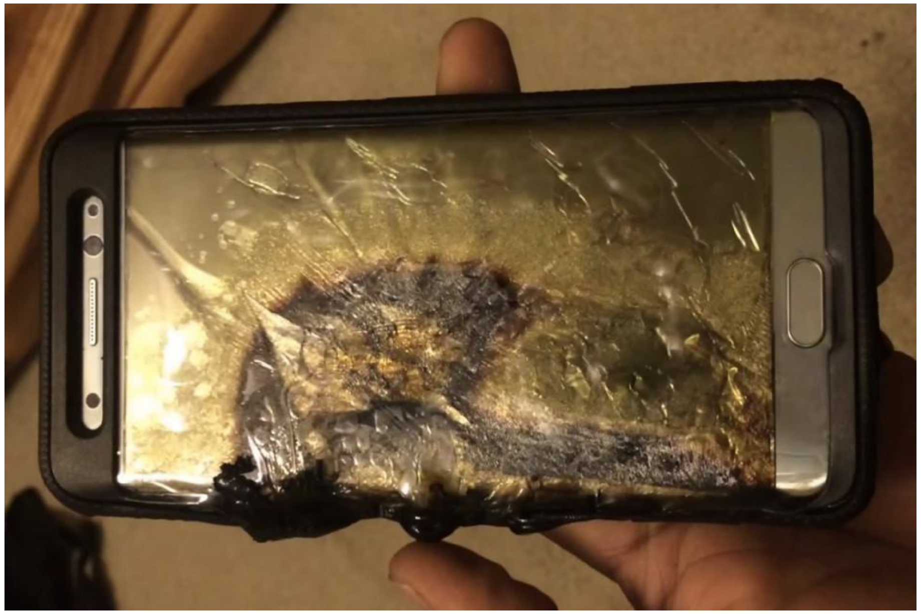
Uses Laws and Insights of Science


Uber cities
● Where ridesharing company Uber operates
● Where operations have been banned or suspended
● Operations regulated/under review
CANADA
U.S.A.
RUSSIA
CHINA
BRAZIL
SOUTH AFRICA
AUSTRALIA
SINGAPORE
Dec. 8 Banned in the Netherlands
Dec. 8 Sued to stop operating by city of Portland, Oregon
Nov. 26 Operations suspended in state of Nevada
Dec. 8 Banned in Delhi, India
Dec. 9 Banned in Thailand
Source: Uber
F. Chan, 09/12/2014
REUTERS

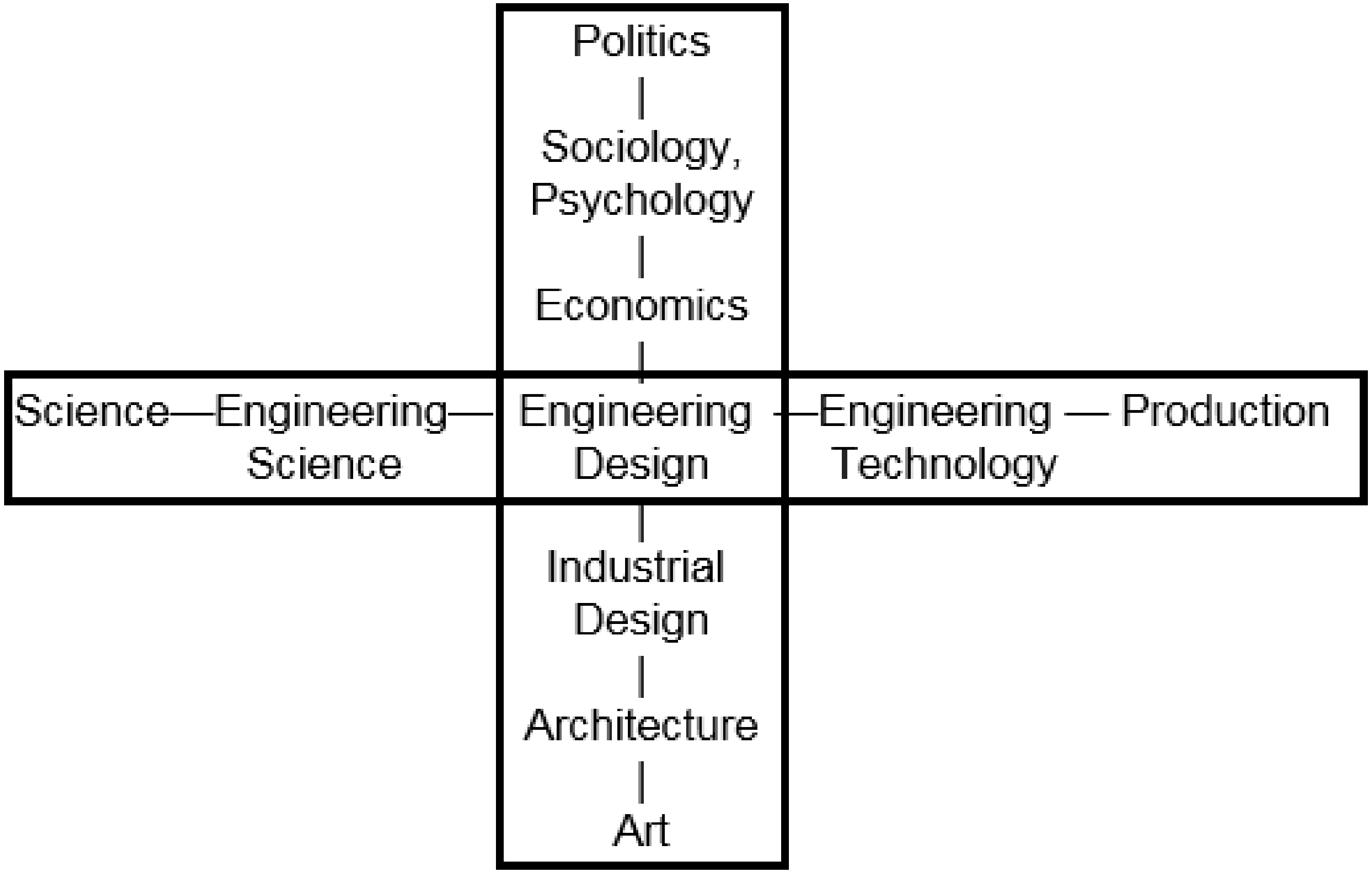
Builds upon special experiences

Provides the prerequisites for realization

Requires integrity and responsibility



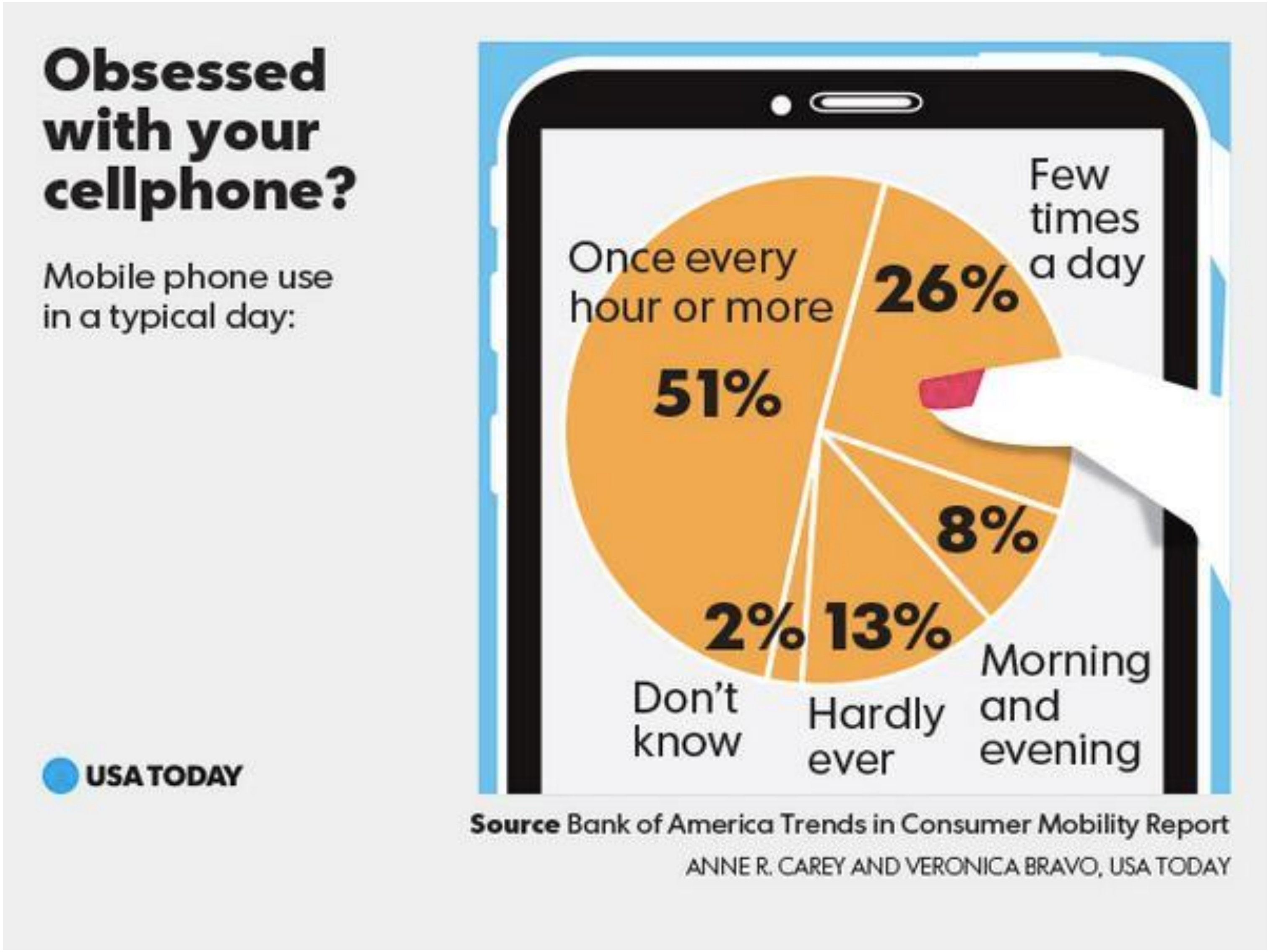
An Overview of Engineering Design



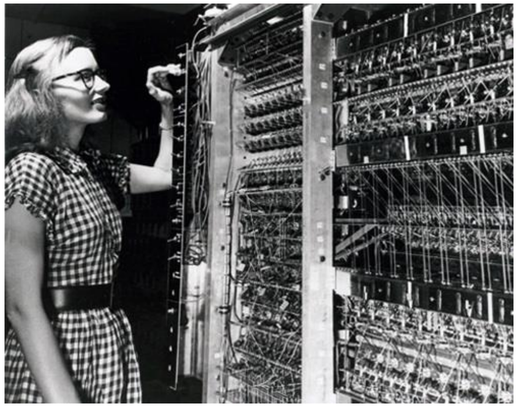
Software Engineering Design

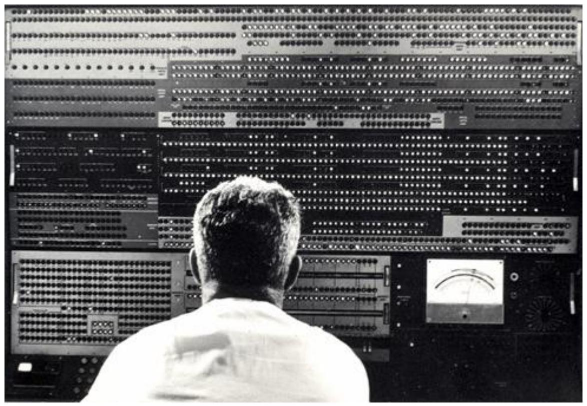
- An ever-important area of engineering design
- Can you think of any modern system/engineering fields without software?

Are you obsessed with your cellphone?



The evolution of computer interface


1952 LAS computer operational

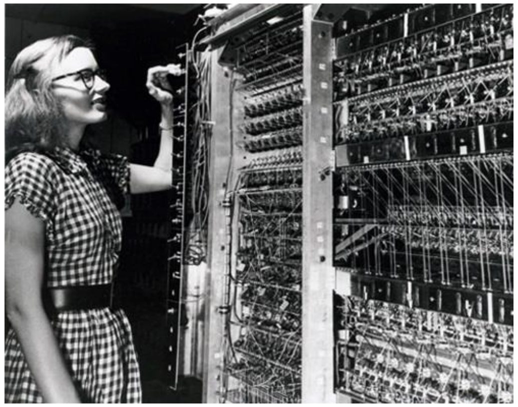

1961 IBM 7030 "Stretch"

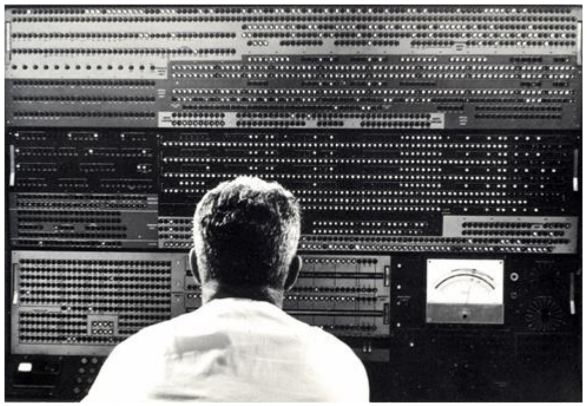
1971 Nova Minicomputer

1977 Apple II introduced

1984 Apple Macintosh

The evolution of computer interface


1952 LAS computer operational


1961 IBM 7030 "Stretch"

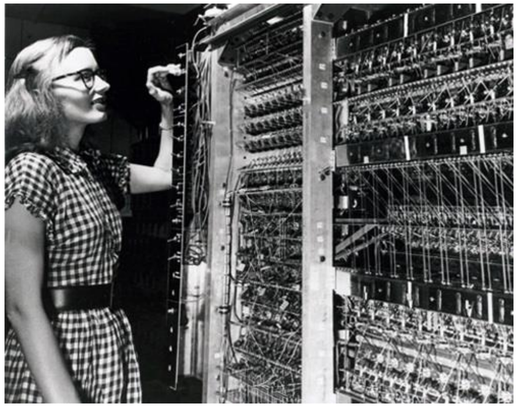
1971 Nova Minicomputer

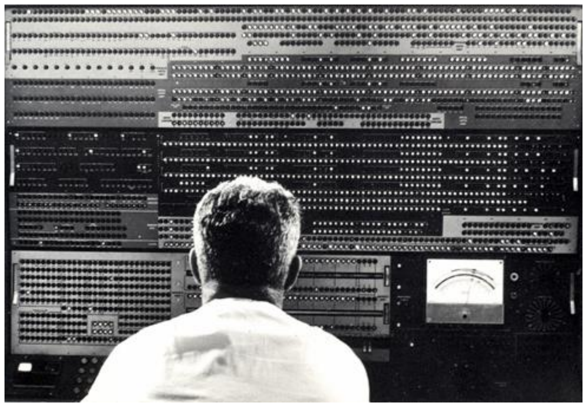
1977 Apple II introduced

1984 Apple Macintosh

3.5 inch display
960 x 640 pixels (4X more)
326 pixels per inch
800:1 contrast ratio (4X better
IPS technology for superb cc
and wide viewing angle

The evolution of computer interface


1952 LAS computer operational


1961 IBM 7030 "Stretch"

1971 Nova Minicomputer

1977 Apple II introduced

1984 Apple Macintosh

User centered design highlights

- Simplifying the structure of the tasks such that the possible actions at any moment are intuitive.
- Make things visible, including the conceptual model of the system, actions, results of actions and feedback.
- Getting the mappings right between intended results and required actions.
- Embracing and exploiting the constraints of systems

Domain Driven Design

- What is domain?
    - It is a particular business area
- What is domain driven design?
    - An approach to software development for complex needs by connecting the implementation to an evolving model.
- The premise of domain-driven design is the following:
    - placing the project's primary focus on the core domain and domain logic;
    - basing complex designs on a model of the domain;
    - initiating a creative collaboration between technical and domain experts to iteratively refine a conceptual model that addresses particular domain problems.

Banking Domain

- A bank is a business that sells financial services such as Vehicle loans, home mortgage loans, business loans, checking accounts, credit card services, certificates of deposit, and individual retirement accounts etc...
    - Types of Banks: commercial, industrial, federal
    - Types of Bank Accounts: savings, certificates
    - Types of Loans: personal, home, business
    - Types of Deposits: saving, fixed, recurring
    - Banking Terms: Appraisal, Forged check

Software Design Life Cycle

Software requirements analysis

System Architecture

Software (Detailed) Design

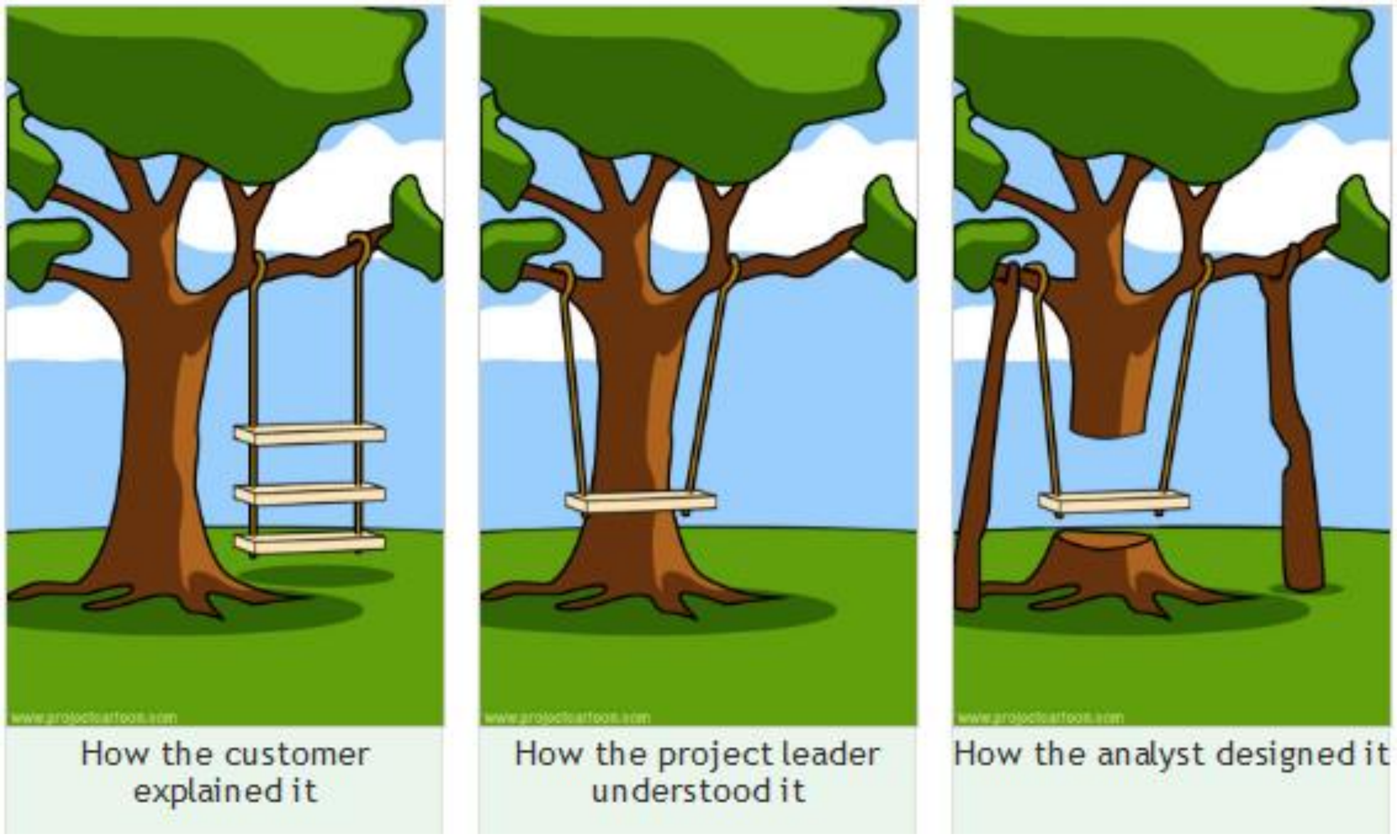
System Verification & Validation

Software requirement

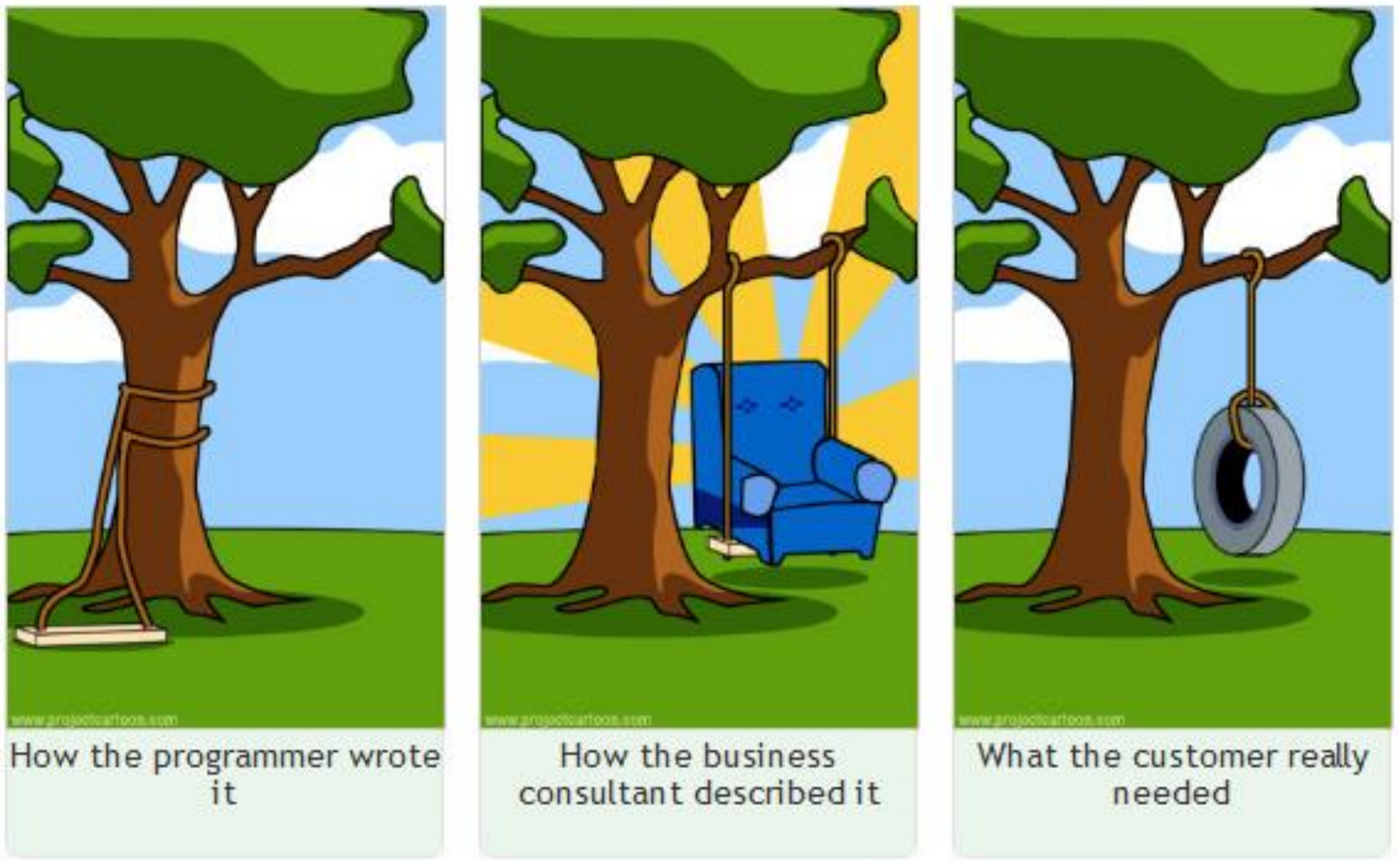
“The hardest single part of building a software system is deciding what to build. No part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.”

–Fred Brooks

Software requirement analysis


www.projectcartoon.com
How the customer explained it
www.projectcartoon.com
How the project leader understood it
www.projectcartoon.com
How the analyst designed it

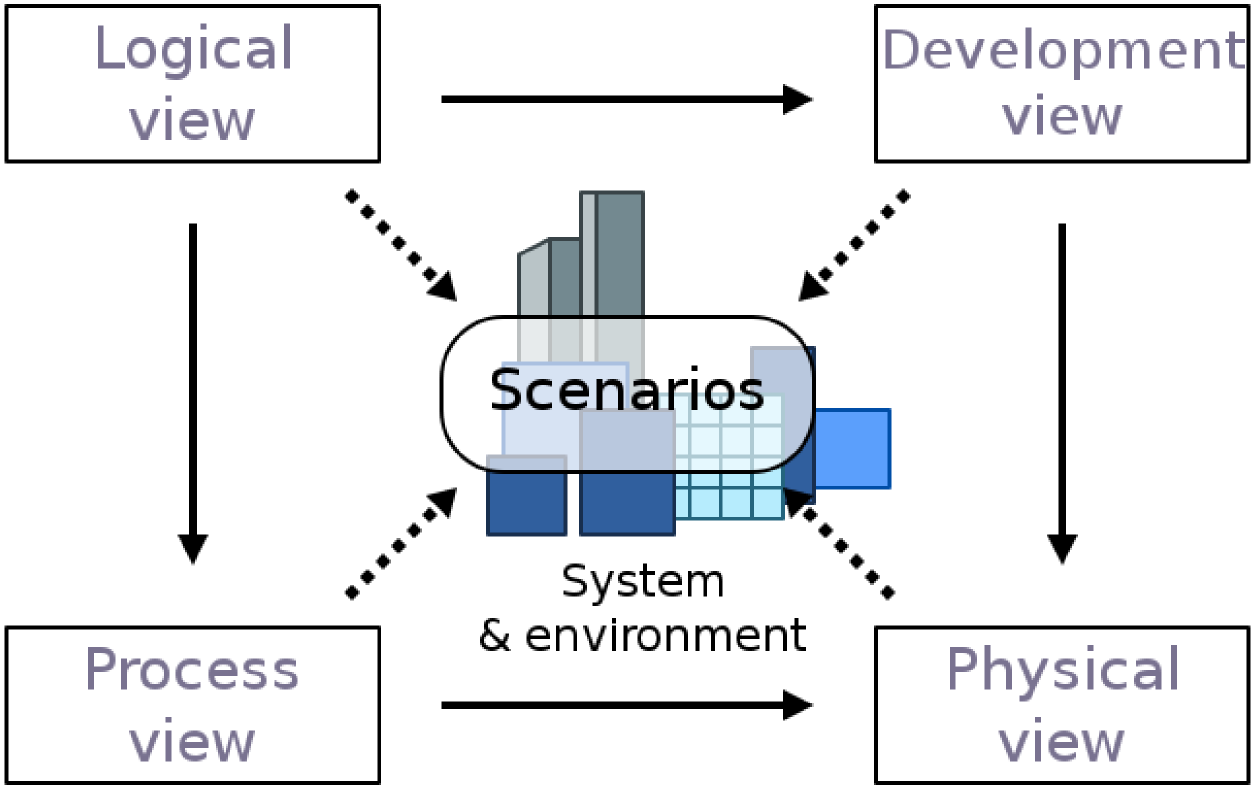
Software requirement analysis


www.projectcartoon.com
How the programmer wrote it
How the business consultant described it
What the customer really needed

What is software architecture?

- It is the high level structures of a software system, the discipline of creating such structures, and the documentation of these structures.
- These structures are needed to reason about the software system. 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.

What is software architecture?



4+1 View Model

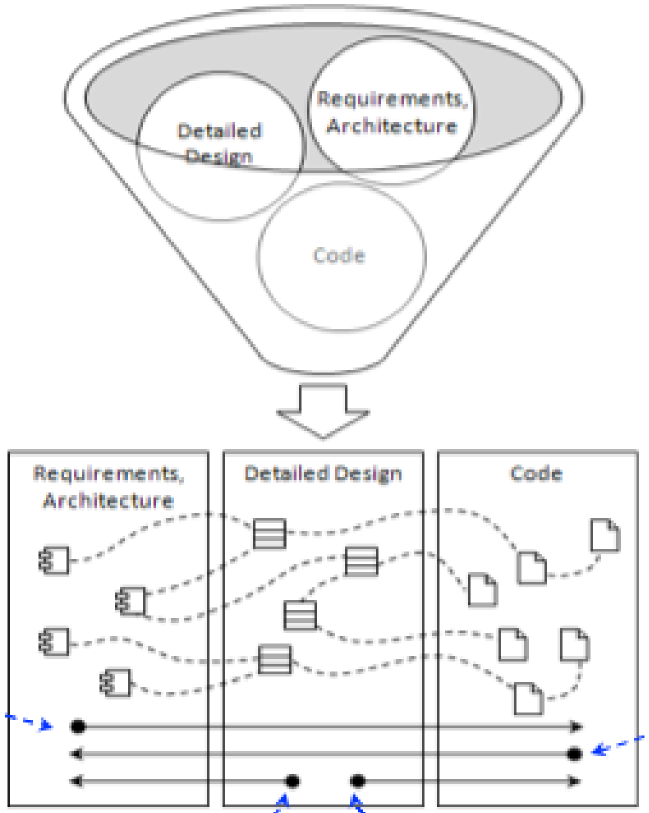
What is detailed design?

- According to the IEEE,
    - The process of refining and expanding the preliminary design (or *software architecture*) of a system or component to the extent that the design is sufficiently complete to be implemented.
    - The result of the above process.

- During *Detailed Design* designers go deep into each component to define its internal structure and behavioral capabilities, and the resulting design leads to natural and efficient construction of software.

What is detailed design?

Designer's Mental Model During Detailed Design!



Important:
During detailed design, the use of industry-grade development tools are essential for modeling, code generation, compiling generated code, reverse engineering, software configuration management, etc.

If given requirements and architecture, detailed designers must move the project forward all the way to code

If given code, detailed designers must be able to reverse engineer the code to produce detailed and architectural designs.

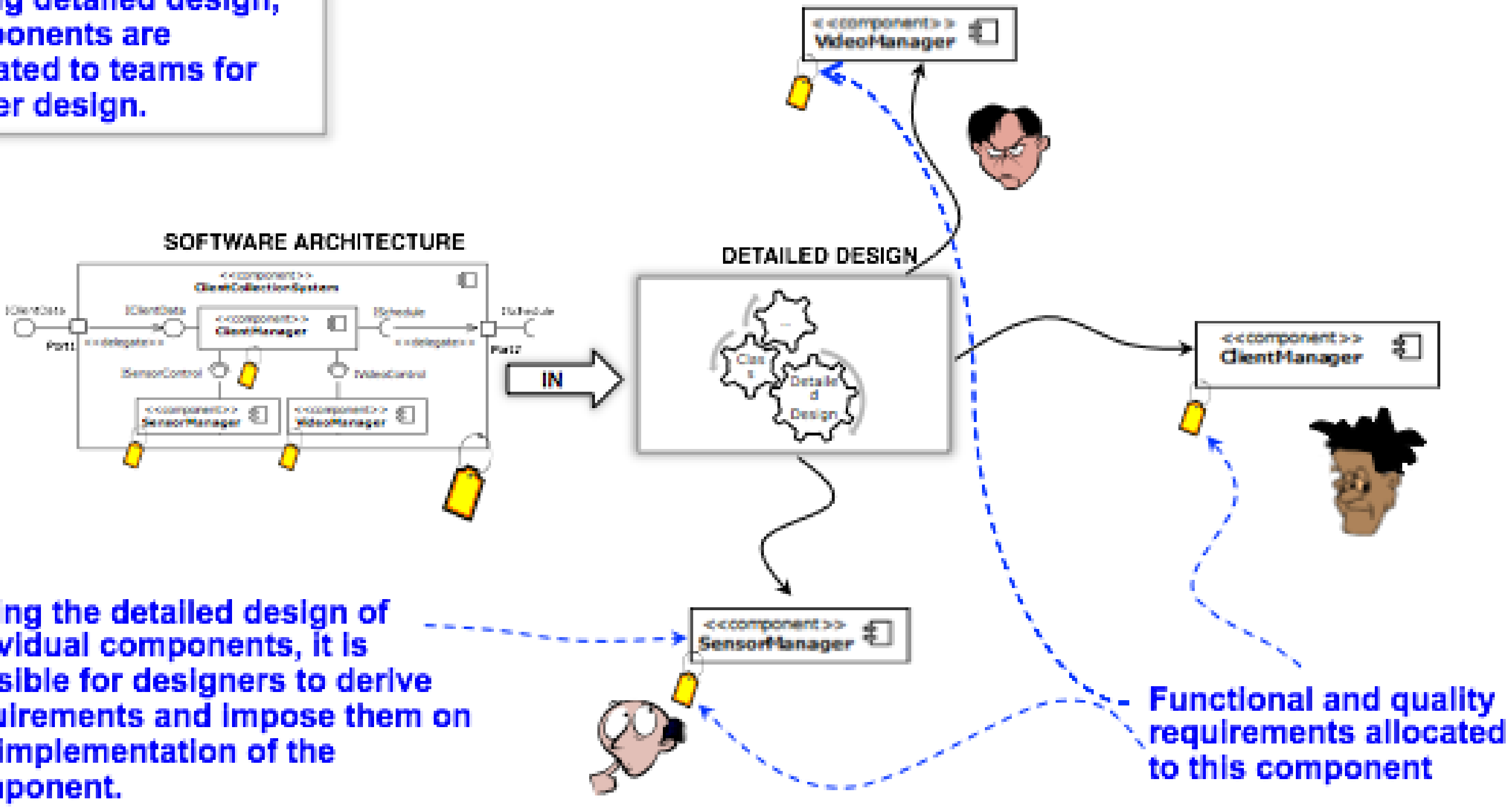
When starting at detailed design, designers must be able to produce both code and architectural designs

What are the tasks in detailed design?

- The major tasks identified for carrying out the detailed design activity include:
  1. **Understanding the architecture and requirements**
  2. **Creating detailed designs**
  3. **Evaluating detailed designs**
  4. **Documenting the design**
  5. **Monitoring and controlling implementation**

How detailed design relates to architecture design?

Important:
During detailed design, components are allocated to teams for further design.



SOFTWARE ARCHITECTURE
<<component>> ClientCollectionSystem
IClientData
<<component>> ClientManager
ISchedule
<<delegate>>
Port1
Port2
SensorControl
VideoControl
<<component>> SensorManager
<<component>> VideoManager
IN
DETAILED DESIGN
Class
Detailed Design
<<component>> VideoManager
<<component>> ClientManager
<<component>> SensorManager

During the detailed design of individual components, it is possible for designers to derive requirements and impose them on the implementation of the component.

Functional and quality requirements allocated to this component

What’s the focus of detailed design?

- Interface Design
  - Internal interface design: How the components interact with other internal components.
  - External interface design: How the components interact with other external components (i.e., components outside the scope of the system being developed).
- Internal Component Design
  - Structural
  - Behavioral
- Data Design
- Graphical User Interface Design
  - This may be a continuation of designs originated during architecture.

What are the challenges of design?

- Complexity in a domain area:
  - Functional requirements
    - functionality, users, etc.
  - Non-functional requirements
    - performance
    - usability
    - reliability
    - cost
  - Interfacing with the outside world (e.g., other evolving s/w systems)
  - Unstable requirements.

What are the goals of design?

- Correctness: Ensure that the code does what's intended to do.
- Robustness: Handle miscellaneous and unusual conditions, such as bad data, user error, programmer error, and environmental conditions.
- Flexibility :The requirements will change… many times
(e.g., handle more kinds of (bank) accounts without needing to change the existing design or code.)
- Efficiency: Program execution should be as fast as required (using available resources, such as memory, disk, network bandwidth, etc).
- Quality / Reliability: "Clean" designs facilitate the implementation of error-free code.
- Usability: Users should find the product easy to use.
- Reusability: Be able to easily modify [the design / implementation] so that it can be reused.
- Maintainability: Experienced developers (unfamiliar with the code) should be able to easily figure out the organization of the code, how things work, and be able to safely modify the code.

What are design fundamental principles?

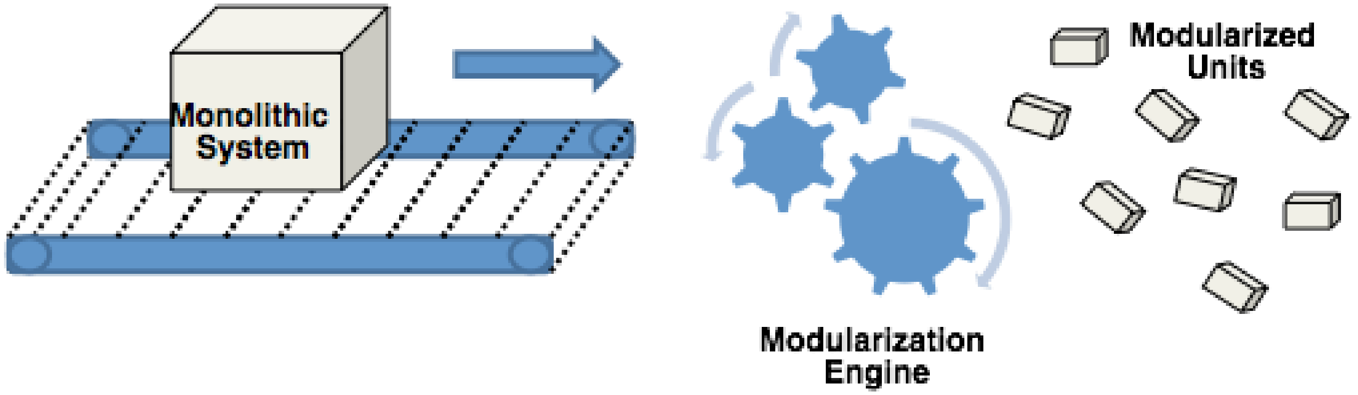
- Refer to knowledge that has been found effective throughout the years in multiple projects on different domains.
- Serve as fundamental drivers for decision-making during the software design process.
- Are not specific to particular design strategies (e.g., object-oriented) or process.
  - They are fundamental to every design effort.
  - Can be employed during architecture, detailed design, construction design, etc.

Software design fundamentals

- *Modularization*
- *Abstraction*
- *Encapsulation*
- *Coupling*
- *Cohesion*
- *Separation of Interface and Implementation*
- *Sufficiency*
- *Completeness*

Modularization

Modularization is the process of continuous decomposition of the software system until fine-grained components are created.



- One of the most important design principle, since it allows software systems to be manageable at all phases of the development life-cycle.

What’s the benefits of modularization?

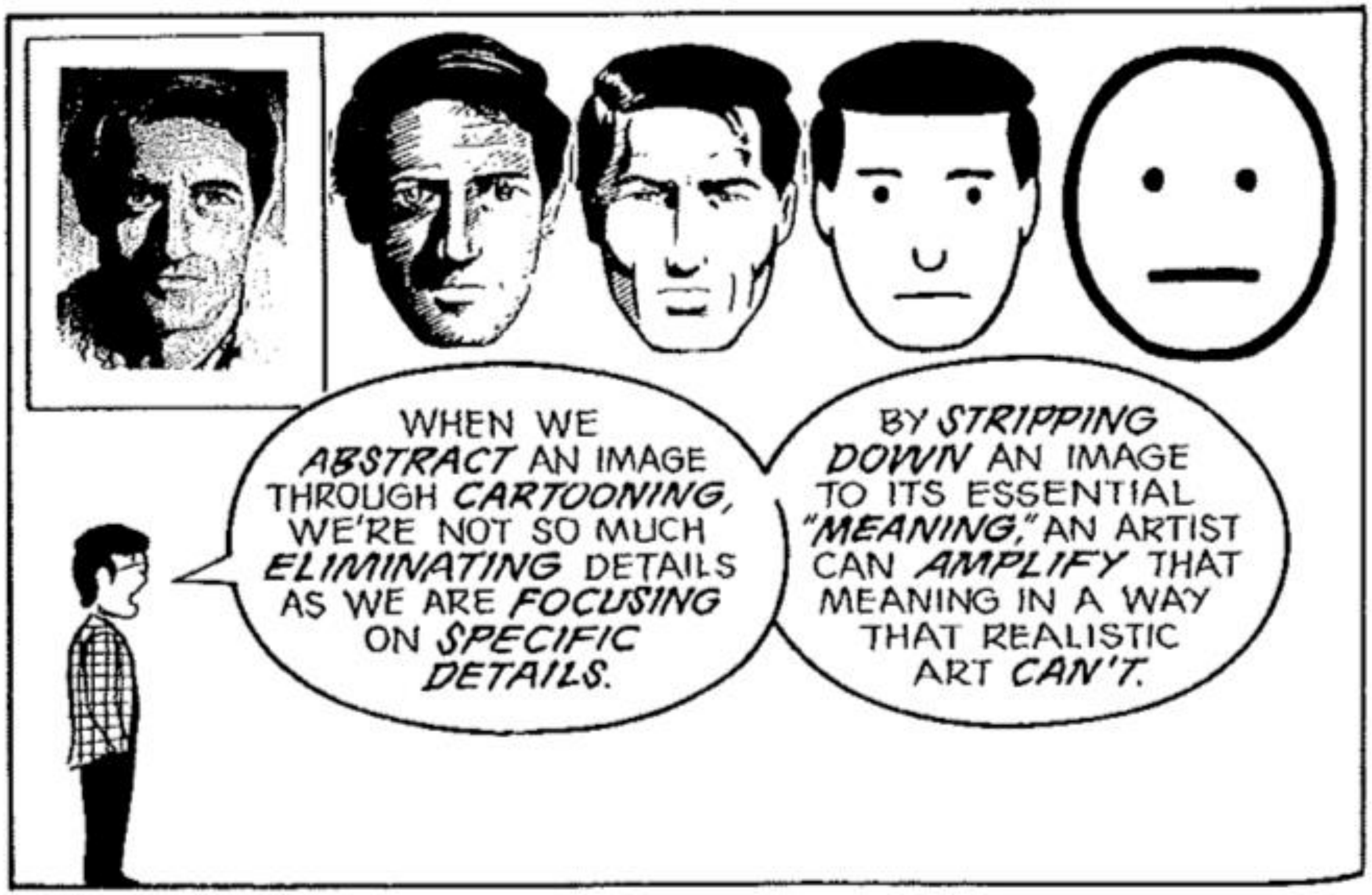
What’s the benefits of modularization?

- Team work/ Parallel development
- Easier to debug
- Reusable code
- More understandable/ Readability
- Technology update/Renovation

Abstraction

- Abstraction is the principle that focuses on essential characteristics of entities—in their active context—while deferring unnecessary details.
- While the principle of modularization specifies what needs to be done, the principle of abstraction provides guidance as to how it should be done.
- Abstraction can be employed to extract essential characteristics of:
  - Procedures or behavior
  - Data

Abstraction


WHEN WE ABSTRACT AN IMAGE THROUGH CARTOONING, WE'RE NOT SO MUCH ELIMINATING DETAILS AS WE ARE FOCUSING ON SPECIFIC DETAILS.
BY STRIPPING DOWN AN IMAGE TO ITS ESSENTIAL "MEANING," AN ARTIST CAN AMPLIFY THAT MEANING IN A WAY THAT REALISTIC ART CAN'T.

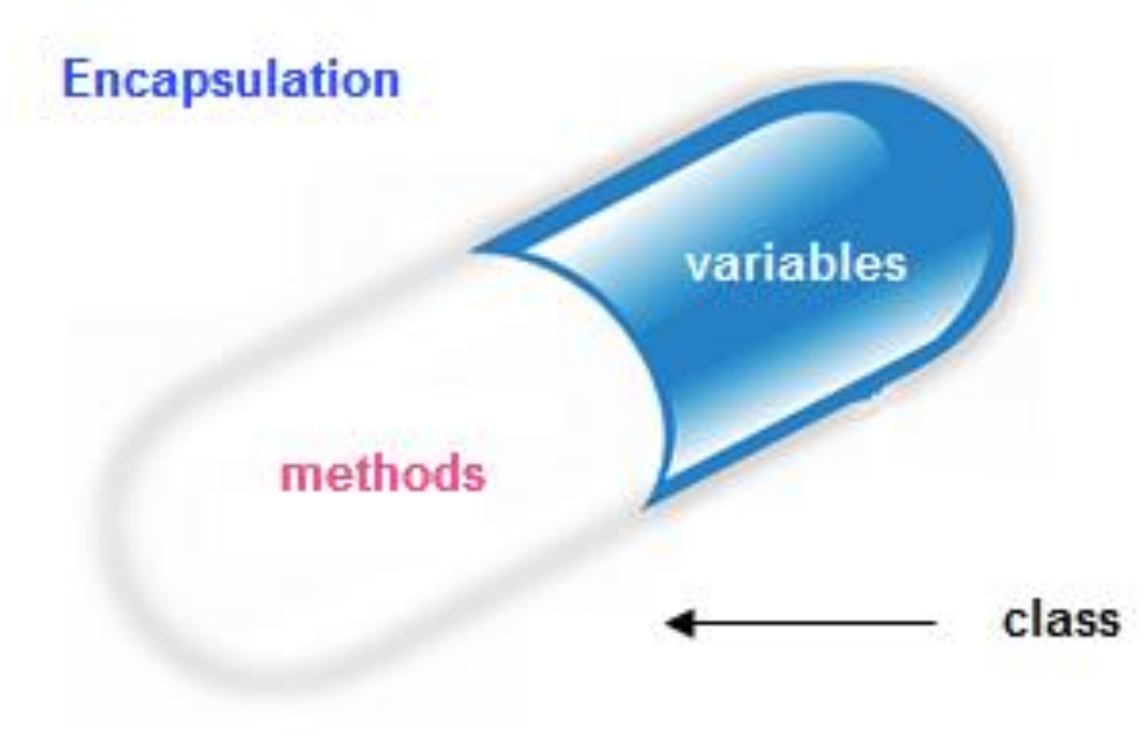
Abstraction

- *Procedural abstraction simplifies reasoning about behavioral operations containing a sequence of steps.*
    - We use this all the time, e.g., consider the statement: "Computer 1 SENDS a message to server computer 2"
        - *Imagine if we had to say, e.g., "Computer 1 retrieves the server's information, opens a TCP/IP connection, sends the message, waits for response, and closes the connection." Luckily, the procedural abstraction SEND helps simplify the operations so that we can reason about this operations more efficiently.*
- *Data abstraction: simplifies reasoning about structural composition of data objects.*
        - In the previous example, MESSAGE is an example of the data abstraction; the details of a MESSAGE can be deferred to later stages of the design phase.

Encapsulation

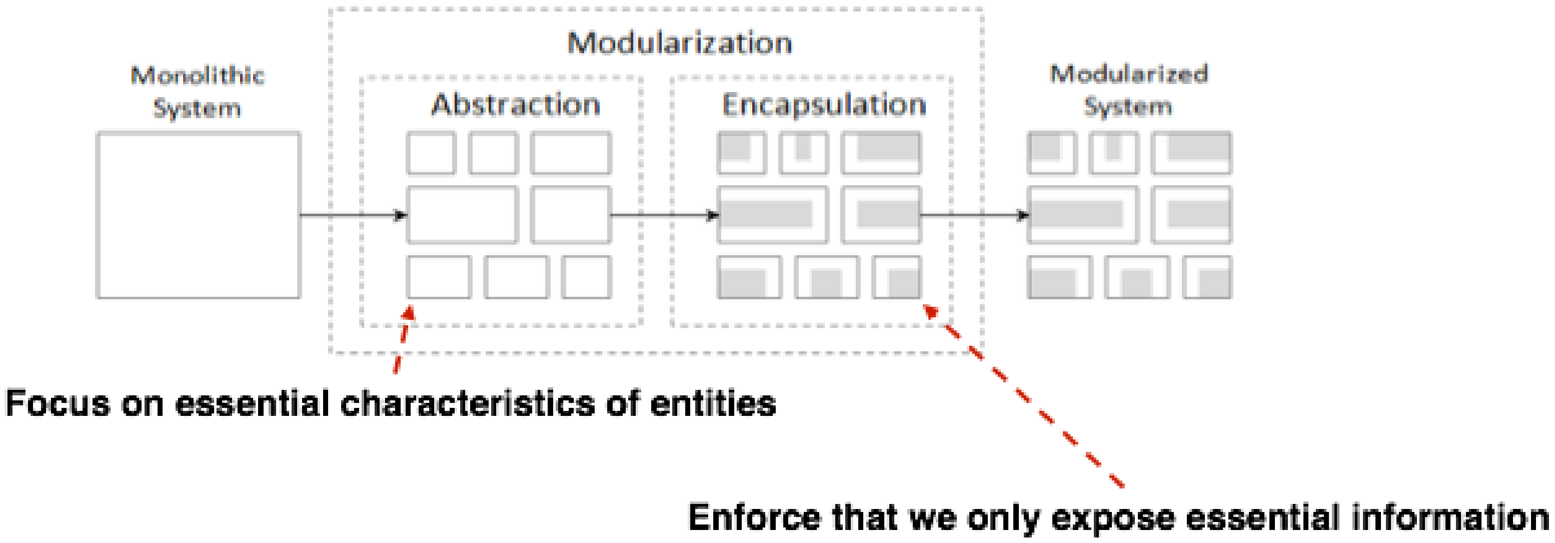
- Principle that deals with providing access to the services of *abstracted entities* by exposing only the information that is essential to carry out such services while hiding details of how the services are carried out.
- When applied to data, encapsulation provides access only to the necessary data of abstracted entities, no more, no less.
- Encapsulation and abstraction go hand in hand.
  - When we do abstraction, we hide details…
  - When we do encapsulation, we revise our abstractions to enforce that abstracted entities only expose essential information, no more, no less.
  - Encapsulation forces us to create good abstractions!

Encapsulation



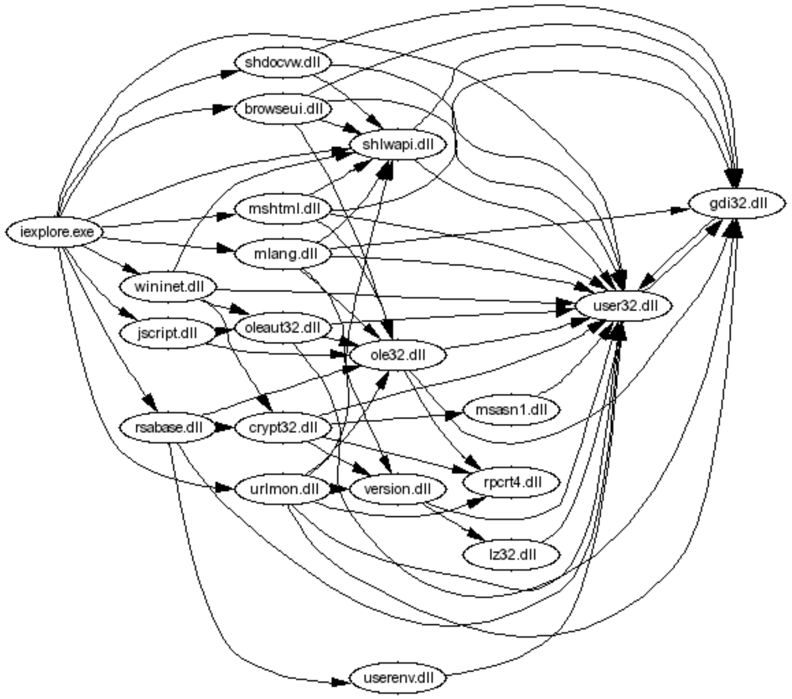
Modularization, abstract, encapsulation

- The principles of *modularization*, *abstraction*, and *encapsulation* can be summarized below.


Modularization
Monolithic System
Abstraction
Encapsulation
Modularized System
Focus on essential characteristics of entities
Enforce that we only expose essential information

Coupling

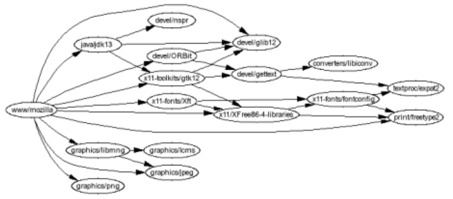
- Refers to the manner and degree of interdependence between software modules.
- Measurement of dependency between units. The higher the coupling, the higher the dependency and vice versa.

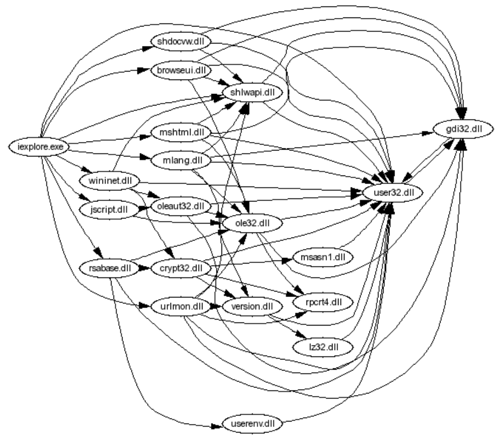


Different types of coupling

- Content coupling
    - Refers to modules that modify and rely on internal information of other modules.
    - The most severe type.
- Common coupling
    - Refers to dependencies based on common access areas, e.g., global variables.
    - When this occurs, changes to the global area causes changes in all dependent modules.
    - Lesser severity than content coupling.
- Data coupling
    - Dependency through data passed between modules, e.g., through function parameters.
    - Does not depend on other modules’ internals or globally accessible data, therefore, design units are shielded from changes in other places.

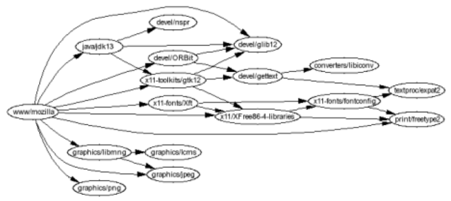
Should coupling be high or low?

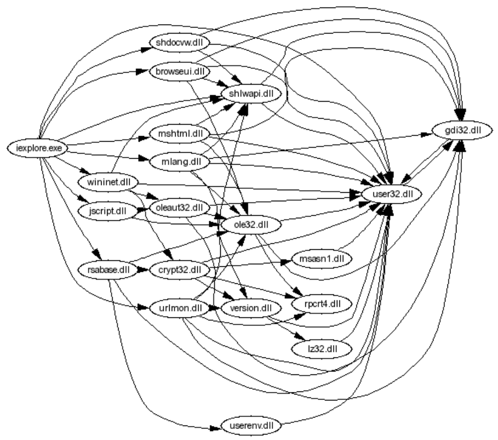




Should coupling be high or low?

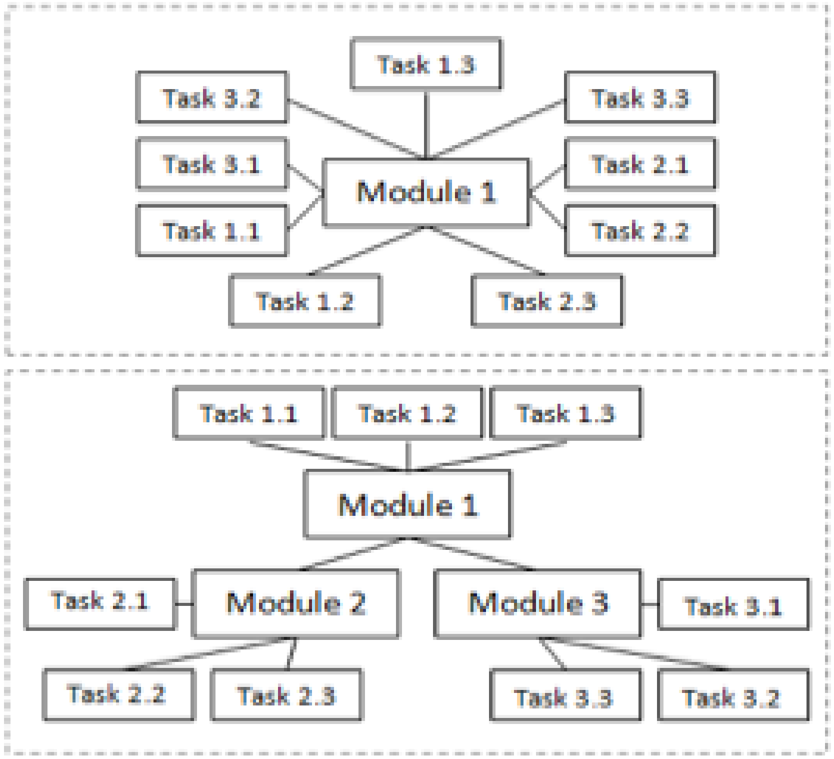
- In all cases, a high degree of coupling gives rise to negative side effects.
  - Quality, in terms of reusability and maintainability, decrease.
  - When coupling increase, so does complexity of managing and maintaining design units.





Cohesion

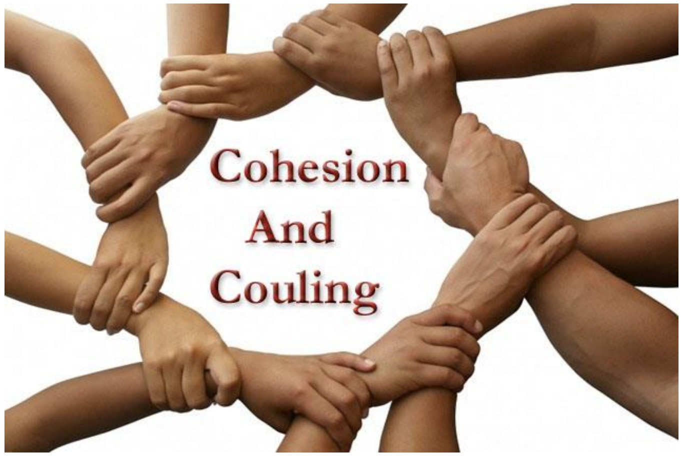
- The manner and degree to which the tasks performed by a single software module are related to one another.
- Measures how well design units are put together for achieving a particular tasks.



Coupling and Cohesion

A. High cohesion good,
low cohesion bad…

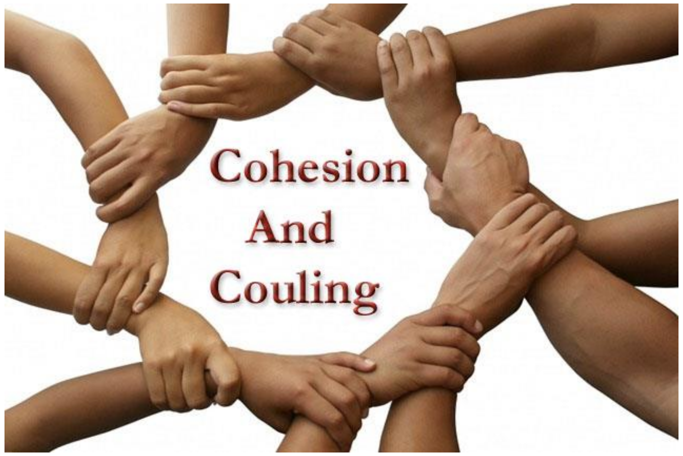
B. High cohesion bad,
low cohesion good…



Coupling and Cohesion

A. High cohesion good, low cohesion bad… ✓

B. High cohesion bad, low cohesion good…



Separation of Interface and Implementation

- Deals with creating modules in such way that a stable interface is identified and separated from its implementation.
- Not the same thing as encapsulation!
- While encapsulation dictates hiding the details of implementation, this principle dictates their separation, so that different implementations of the same interface can be swapped to provide modified or new behavior.

Sufficiency

- Deals with capturing enough characteristics of the abstraction to permit meaningful interaction.
- Must provide a full set of operations to allow a client proper interaction with the abstraction.
- Implies minimal interface.

Completeness

- Deals with interface capturing all the essential characteristics of the abstraction.
- Implies an interface general enough for any prospective client.
- Completeness is subjective, and carried too far can have unwanted results.