SOFTWARE ENGINEERING DESIGN CONCEPTS

Dr. Tony D. Barber / Prof. Sahana Kini

Fall, 2022

PRESENTATIONS (NEXT WEEK)

OBJECTIVES AND OUTCOME

- Present and review LMS personas and user interface prototype
  - Each team: presentation + 5 minutes Q&As/feedback/comments
  - Demonstrate
    - Verbal communication skills
    - Time management
    - Critical thinking
  - Practice providing and receiving feedback

BEFORE YOU SAY (WRITE) ANYTHING, ASK YOURSELF:

- Is it **TRUE**?

- Is it **TIMELY**?

- Is it **USEFUL**?

- Is it **KIND**?

In a world where
You can be
anything, be
kind.

TODAY'S CLASS OBJECTIVES AND OUTCOME

- Presentation from Mr. Eugene L. Vickers, U.S. Army Combat Capabilities Development Command (DEVCOM)
- Define design, and specifically software engineering design
- Identify design stakeholders
- Explain the software design process
- Describe design challenges and principles
- Recognize similarities and differences between architecture and detailed design
- Identify major software design drivers
- Identify major design decisions, perform tradeoff analyses, and select candidate solution

U.S. ARMY COMBAT CAPABILITIES DEVELOPMENT COMMAND (DEVCOM)

Mr. Eugene L. Vickers

DEVCOM CBC Diversity, Equity and Inclusion/Wellness Officer (DEI)

OUTLINE

- Introduction to Software Design
- Design definition and concepts
- Design challenges, principles, and process
- Software architecture definition, concepts
- Architecture stakeholders and their need for/use of design
- Architecture and Detailed design
- Class exercises: LMS major architecture elements
- Assignments

RESOURCES

- **Books**
  - Textbook Chapters 8 and 9
  - *Software Design Methodology - From principles to Architectural Styles*, by Hong Zhu
  - *Software Design – From Programming to Architecture*, by Eric Braude
  - *Software Architecture in Practice, (SEI Series in Software Engineering),* by Len Bass and Paul Clements
  - *Software Modeling & Design*, by Hassan Gomaa
  - *Architecting Software Intensive Systems*, by Anthony Lattanze
- Software Engineering Institute (SEI) web site: http://www.sei.cmu.edu/architecture/
- **Papers:**
  - DeRemer, Frank; Kron, Hans (1975). *Programming-in-the large versus programming-in-the-small*. Proceedings of the international conference on Reliable software. Association for Computing Machinery. pp. 114–121
  - David Garlan, Mary Shaw, *An Introduction to Software Architecture*

OUTLINE

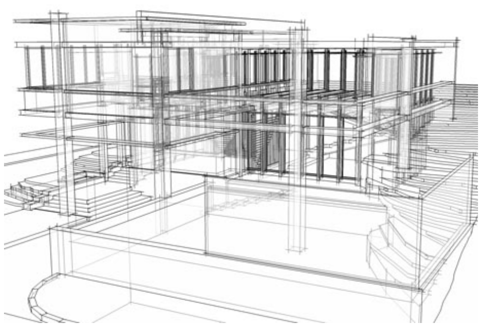
We are here

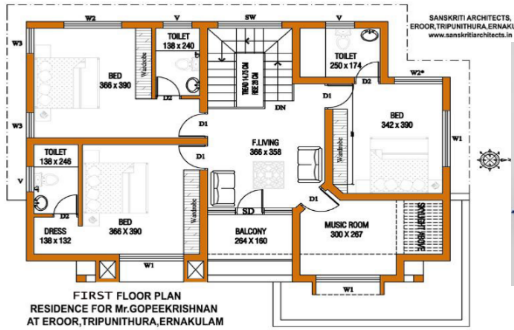
- Introduction to Software Design
- Design definition and concepts
- Design challenges, principles, and process
- Software architecture definition, concepts
- Architecture stakeholders and their need for/use of design
- Architecture and Detailed design
- Class exercises: LMS major architecture elements
- Assignments

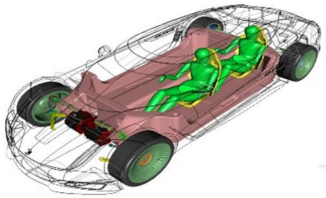
ENGINEERING DESIGN

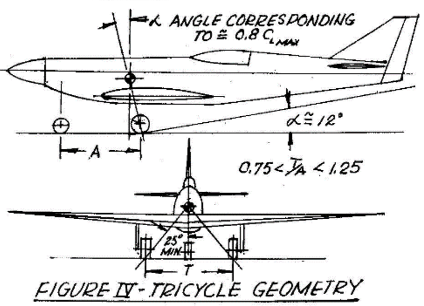
- Design – as a *verb* (process) and a *noun* (artifact resulting from the process)
- Design – as a *process*:
    - Industrial design is a discipline historically known for *creating* **products** and **systems** that *optimize* **function**, **value** and **appearance** for the mutual benefit of stakeholders involved (Industrial Designers Society of America)
    - The engineering design process is a **methodical series of steps** that engineers use in creating functional products and processes
    - Engineering design is the **method** that engineers use to ***identify and solve problems***
- Design – as an *artifact,* is an abstraction of the product (object, system, software)

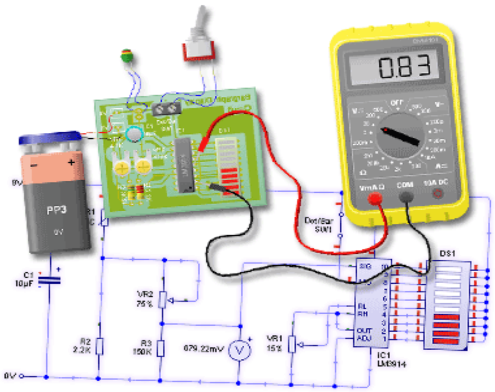
IN WHAT APPLICATIONS DOMAINS AND (ENGINEERING) DISCIPLINES IS DESIGN PERFORMED?

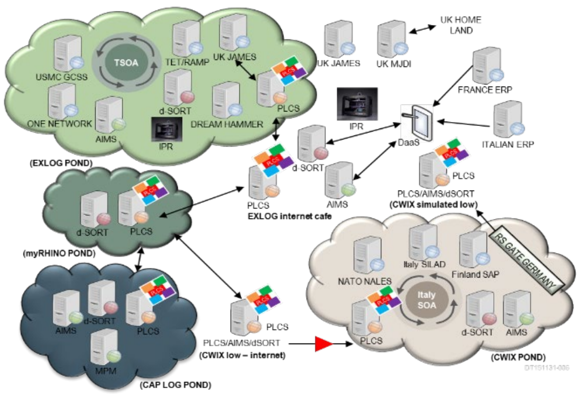


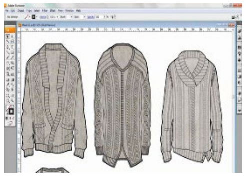


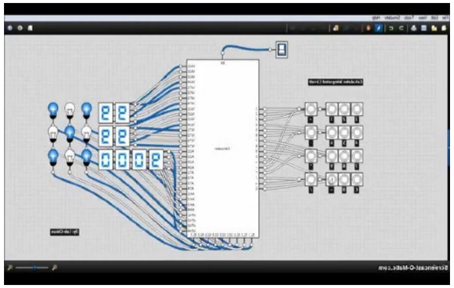


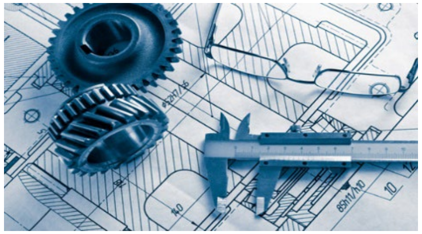


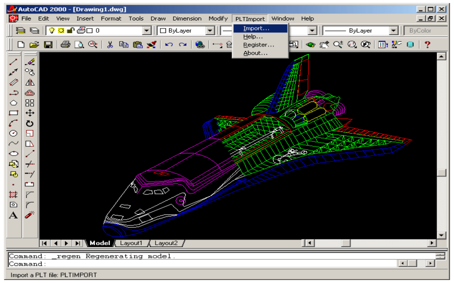


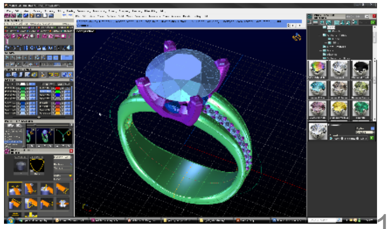












IN WHAT ENGINEERING DISCIPLINE IS DESIGN PERFORMED?

- Civil
- Systems
- Mechanical
- Electrical
- Industrial processes
- Software
- ….

- And in any *application* domain

OUTLINE

We are here

- Introduction to Software Design
- Design definition and concepts
- Design challenges, principles, and process
- Software architecture definition, concepts
- Architecture stakeholders and their need for/use of design
- Architecture and Detailed design
- Class exercises: LMS major architecture elements
- Assignments

WHAT IS *SOFTWARE* ENGINEERING DESIGN?

- "Software engineering design is the activity of *specifying* **programs** and **sub-systems**, and **their constituent parts and workings**, to meet **software product specifications**." (C. Fox)
- *Deriving* a **solution** which satisfies **software requirements**

- The software to be developed is meant to *solve a problem*
  - Requirements engineering - we *specify* **the problem;**
  - Design - we *develop* and *specify* **a solution**

WHY DO WE NEED DESIGN?

- To reason about solutions
- To document solutions
- To communicate solution(s) to stakeholders and reach agreement

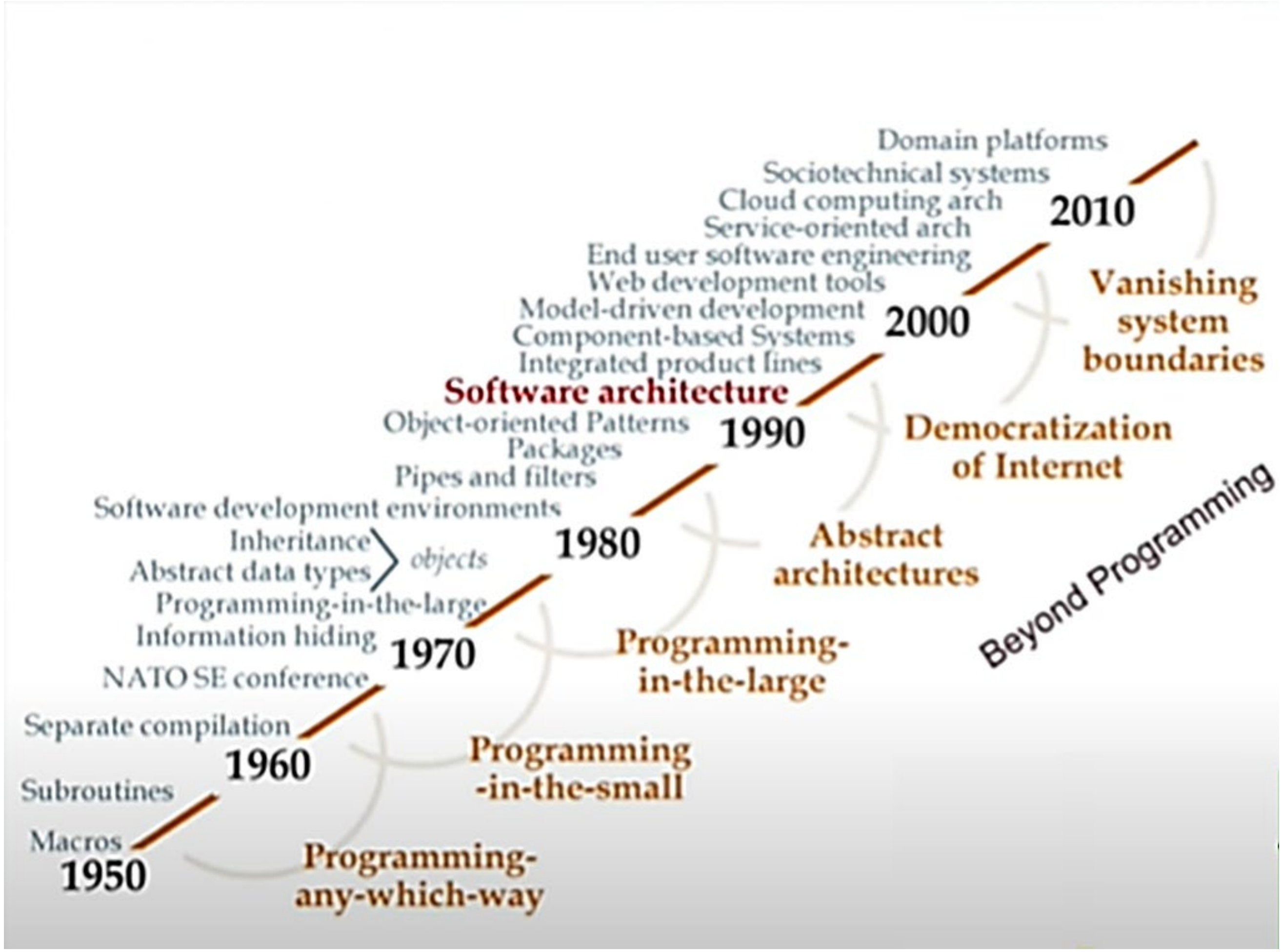
"A survey of 400 companies with 100,000 employees each cited an ***average loss per company of $62.4 million per year because of inadequate communication to and between employees****. "*
From: The Cost of Poor Communications

PROGRAMMING ”IN THE SMALL” AND “IN THE LARGE”

[DeRemer, Frank; Kron, Hans (**1975**). "Programming-in-the large versus programming-in-the-small". *Proceedings of the international conference on Reliable software*. Association for Computing Machinery. pp. 114–121]

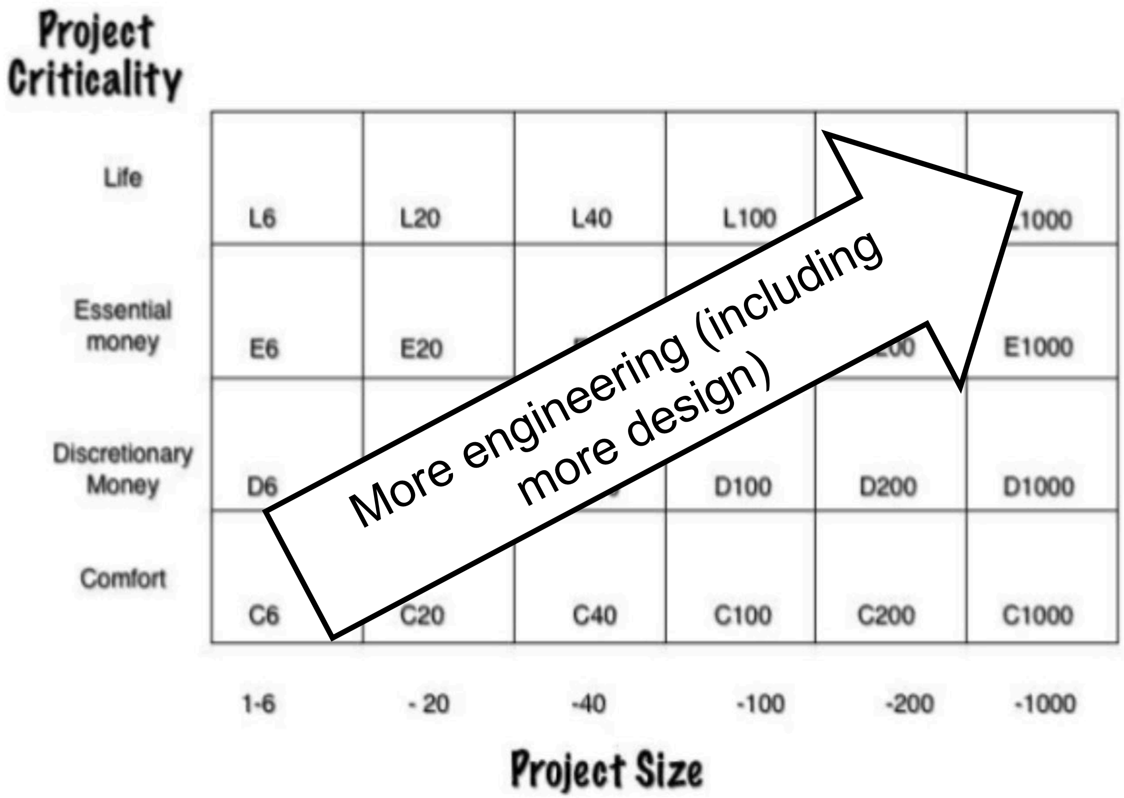
- We distinguish the activity of **writing large programs** from that of **writing small ones**.
  - By large programs we mean systems consisting of many small programs (modules), possibly written by different people.
- We need languages for **programming-in-the-small**, i.e., languages not unlike the common programming languages of today, for writing modules
- We also need a "**module interconnection language**" for knitting those modules together into an integrated whole and for providing an overview that formally records the intent of the programmer(s) and that can be checked for consistency by a compiler

SOFTWARE DEVELOPMENT EVOLUTION



From: *Progress Towards an Engineering Disciple of Software*, by Mary Shaw

HOW MUCH DESIGN?



Alistair Cockburn's project characteristics grid

OUTLINE

- Introduction to Software Design
- Design definition and concepts

We are here

- Design challenges, principles, and process
- Software architecture definition, concepts
- Architecture stakeholders and their need for/use of design
- Architecture and Detailed design
- Class exercises: LMS major architecture elements
- Assignments

WHAT MAKES SOFTWARE DESIGN EASY OR HARD?

- Identify software properties that can make design easy or difficult

WHAT MAKES SOFTWARE DESIGN HARD?

- Complexity
  - Increases non-linearly with size
- Conformity
  - With standards and constraints
- Changeability
- Invisibility

HOW DO WE OVERCOME DESIGN DIFFICULTIES? – APPLYING DESIGN PRINCIPLES AND HEURISTICS

- **Design heuristics**, e.g.:
  - Decomposition and Separation of concerns
    - Design intermediate solution in terms of simpler independent problems
  - Abstraction
    - Taking away or removing characteristics from something in order to reduce it to a set of essential characteristics

HOW DO WE OVERCOME DESIGN DIFFICULTIES? - APPLYING DESIGN PRINCIPLES AND HEURISTICS (CONTINUED)

- **Design principles** - are statements about what determines a design to be "good"

- **Basic principles -** characteristics that make a design ***able to meet stakeholder needs and desires*** *(fit for purpose)*
- **Constructive principles -** engineering design characteristics that make a "good" design
  - Are based on engineering experience

DESIGN PRINCIPLES

- Engineering Design Principles
  - Basic
    - Feasibility
    - Adequacy
    - Changeability
    - Economy
  - Constructive
    - Modularity
      - Small Modules
      - Information Hiding
      - Least Privilege
      - Coupling
      - Cohesion
    - Implementability
      - Simplicity
      - Design with Reuse
      - Design for Reuse
    - Aesthetic
      - Beauty

BASIC PRINCIPLES

- **Feasibility**—A design is acceptable only if it can be realized
- **Adequacy** —Designs that meet more stakeholder needs and desires, subject to constraints, are better
- **Economy** —Design that can be built for less money, in less time, with less risk, are better
- **Changeability** —Design that make a program easier to change are better

CONSTRUCTIVE PRINCIPLES

- *Modularity principles*—Good design are *modular*; these principles help evaluate whether designs specify good modules
- *Implementability principles*—Good designs are *easier to build*; these principles help evaluate whether designs will be easy to implement
- *Aesthetic principles*—Good design are "*beautiful*"; these principles help pick out beautiful designs

MODULARITY

- A **modular** program/software is
  - *composed of* **well-defined, conceptually simple, and independent units**
  - that *communicate* through **well-defined interfaces**

- A **module** is a program **unit**
  - At different levels of the system, the (sub)"units" are different:
    - Program
      - Sub-programs or sub-systems
        - Packages, compilation units
          - Classes, functions
            - Attributes, operations, blocks

BENEFITS OF MODULARITY

- A modular design (and software) is easier to:
  - Understand and explain
  - Document
  - Change
  - Test and debug
  - Reuse
  - Resist to architecture erosion

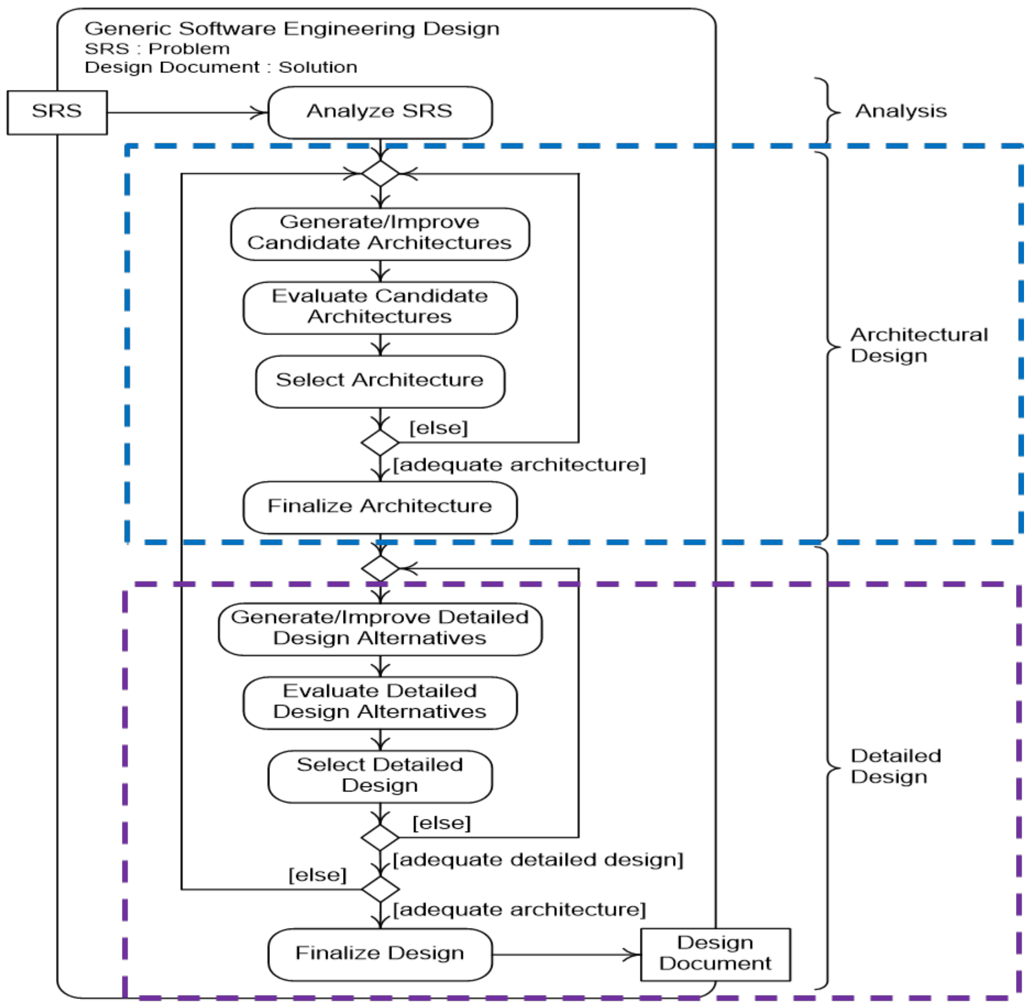
MODULARITY (SUB)PRINCIPLES

- **Small Modules** —Designs with small modules are better
- **Coupling** —Module coupling should be minimized
  - Coupling is the degree of connection between modules
- **Cohesion** —Modules cohesion should be maximized
  - Cohesion is the degree to which a module’s parts are related to one another
- **Information Hiding** —Each module should shield the details of its internal structure and processing from other modules
- **Least Privilege** —Modules should not have access to unneeded resources

IMPLEMENTABILITY PRINCIPLES

- **Simplicity** —Simpler designs are better
- Software **reuse** is the use of existing artifacts to build new software products; reusable artifacts are called assets
  - Design ***with*** reuse—Designs that reuse existing assets are better
  - Design ***for*** reuse—Designs that produce reusable assets are better

SOFTWARE DESIGN PROCESS



OUTLINE

- Introduction to Software Design
- Design definition and concepts
- Design challenges, principles, and process

We are here

- Software architecture definition, concepts
- Architecture stakeholders and their need for/use of design
- Architecture and Detailed design
- Class exercises: LMS major architecture elements
- Assignments

WHAT IS SOFTWARE ARCHITECTURE?

- An abstraction of the working software
  - Helps cope with complexity
- "fundamental concepts or properties of a system in its environment embodied in its **elements**, **relationships**, and in **the principles of its design and evolution**" (*International ISO/IEC/ STANDARD IEEE 42010 - Systems and software engineering — Architecture description*)
- The set of **structures** needed to reason about the system, which comprises **software elements**, **relations** among them, and **properties** of both (*Documenting Software Architectures: Views and Beyond*, by Paul Clements et all)

MORE DEFINITIONS OF SOFTWARE ARCHITECTURE

- ***The set of decisions*** that:
  - are hard to change
  - need to be made (you wish you could make) early
- ***Expert developers' shared understanding of the system design***
  - Architecture is "a social thing"
  - Projects need a good shared understanding
- ***The important "stuff"***

MORE DEFINITIONS OF SOFTWARE ARCHITECTURE (CONTINUED)

- The Software Engineering Institute (SEI) has compiled a list of modern, classic, and bibliographic definitions of software architecture.
  - Modern definitions - from Software Architecture in Practice and from ANSI/IEEE Std 42010, Recommended Practice for Architectural Description of Software-Intensive Systems.
  - Classic definitions - in some of the more prominent or influential books and papers on architecture.
  - Bibliographic definitions - from papers and articles in our software architecture bibliography.
  - Several hundred community definitions

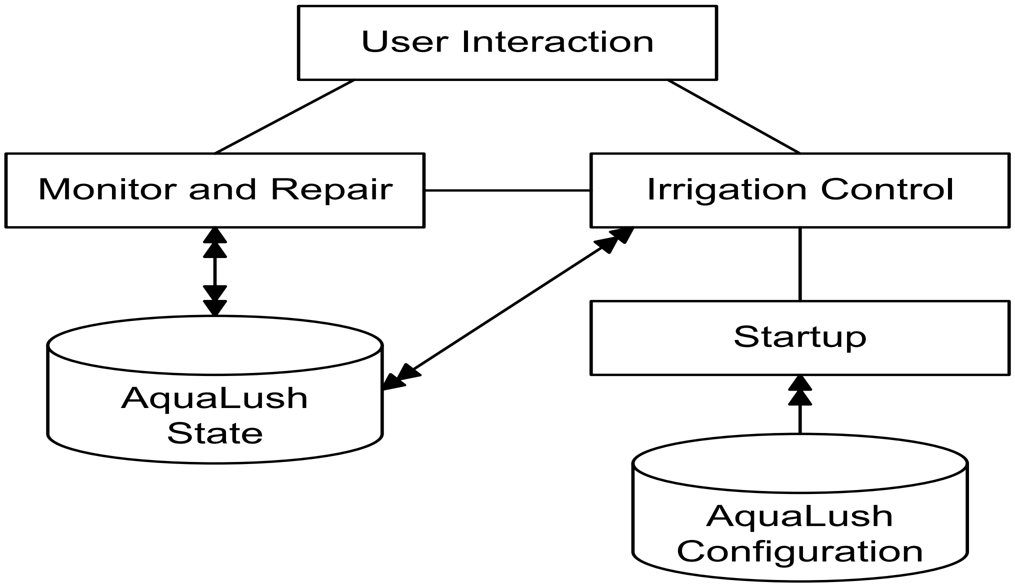
- Watch What is software architecture – George Fairbanks (video)

ARCHITECTURE INFORMATION

- Decomposition
  - Program parts/modules/elements
- Responsibilities of each element
  - Data and behavior
- Interfaces between elements
  - An interface is a boundary across which entities communicate
- Collaborations
  - What each element does, and when
- Relationships between elements
  - Uses, dependencies, etc.
- Properties of elements and software as a whole
  - Performance, reliability, etc.
- States and Transition, externally visible

EXAMPLE 1 SOFTWARE ARCHITECTURE STRUCTURE

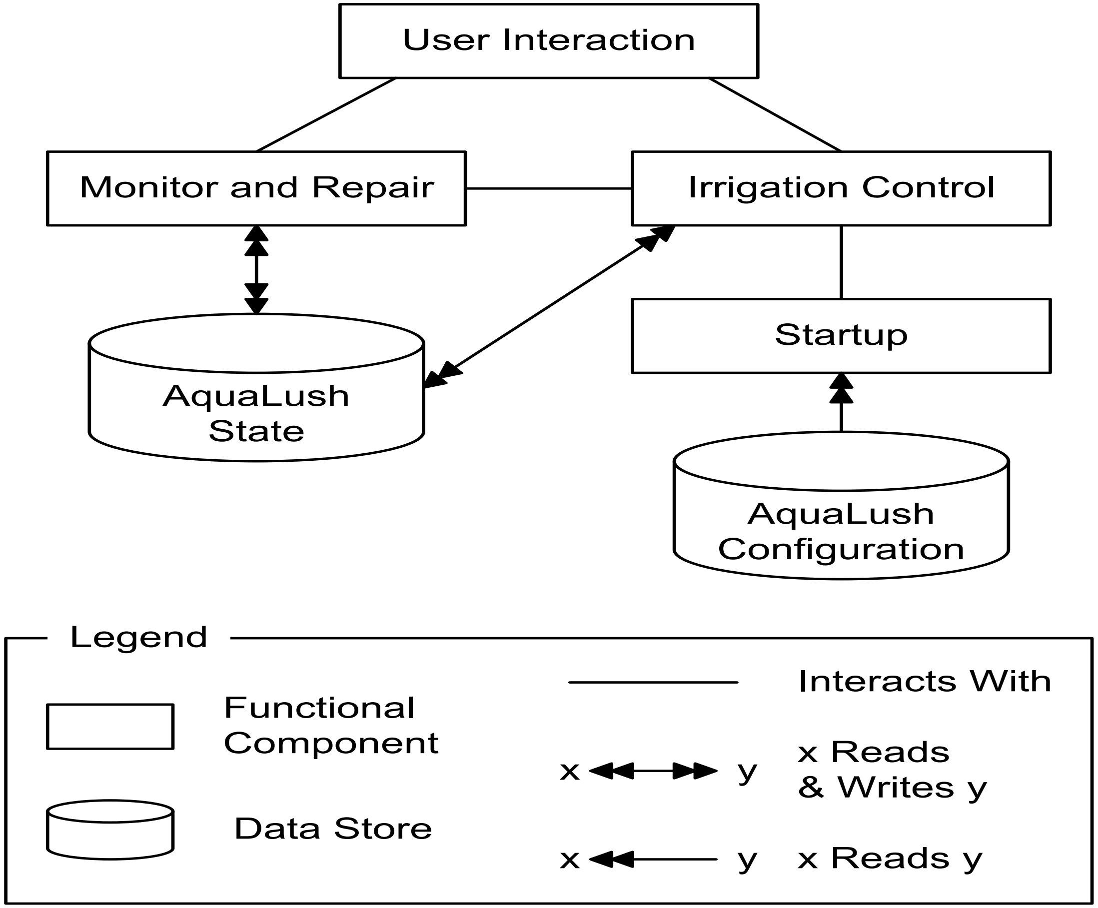
Architecture of AquaLush – a lawn and garden irrigation system



[From: *Introduction to Software Engineering Design*, by Christopher Fox]

EXAMPLE 1 SOFTWARE ARCHITECTURE STRUCTURE

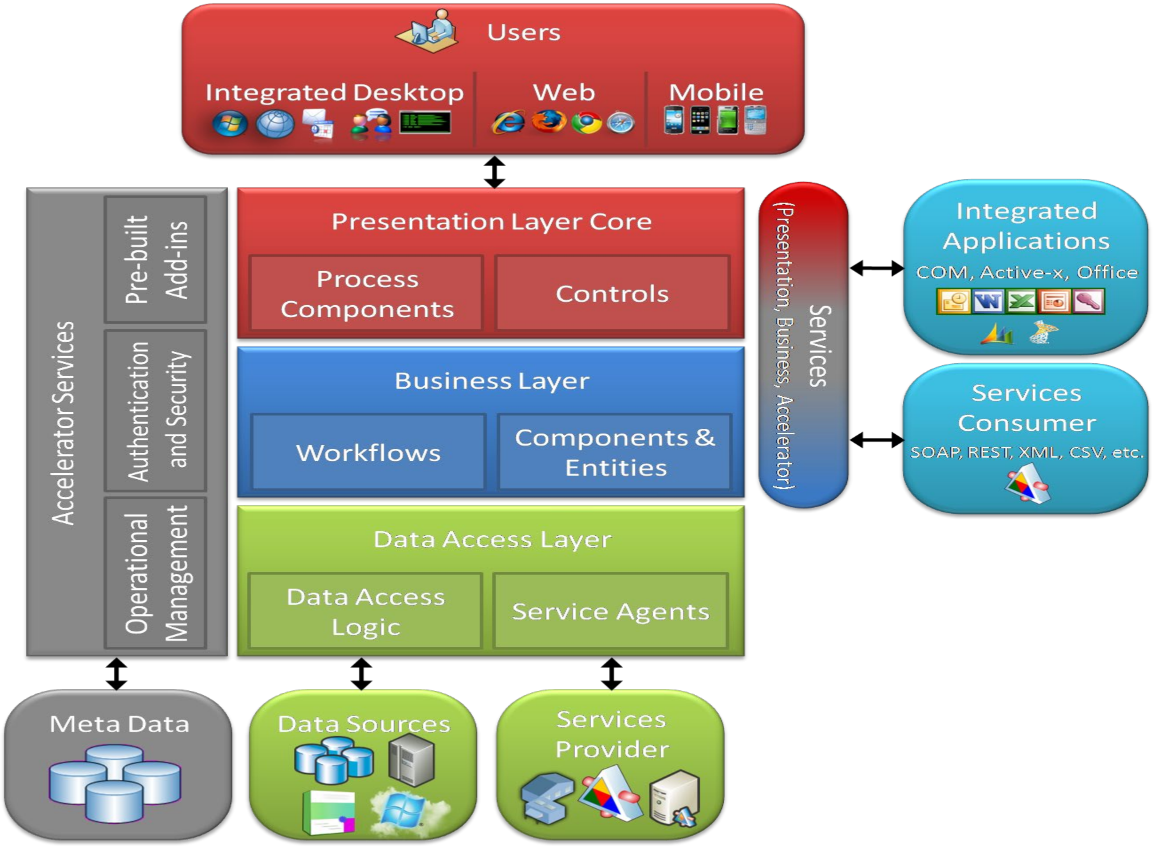
Architecture of AquaLush – a lawn and garden irrigation system



[From: *Introduction to Software Engineering Design*, by Christopher Fox]

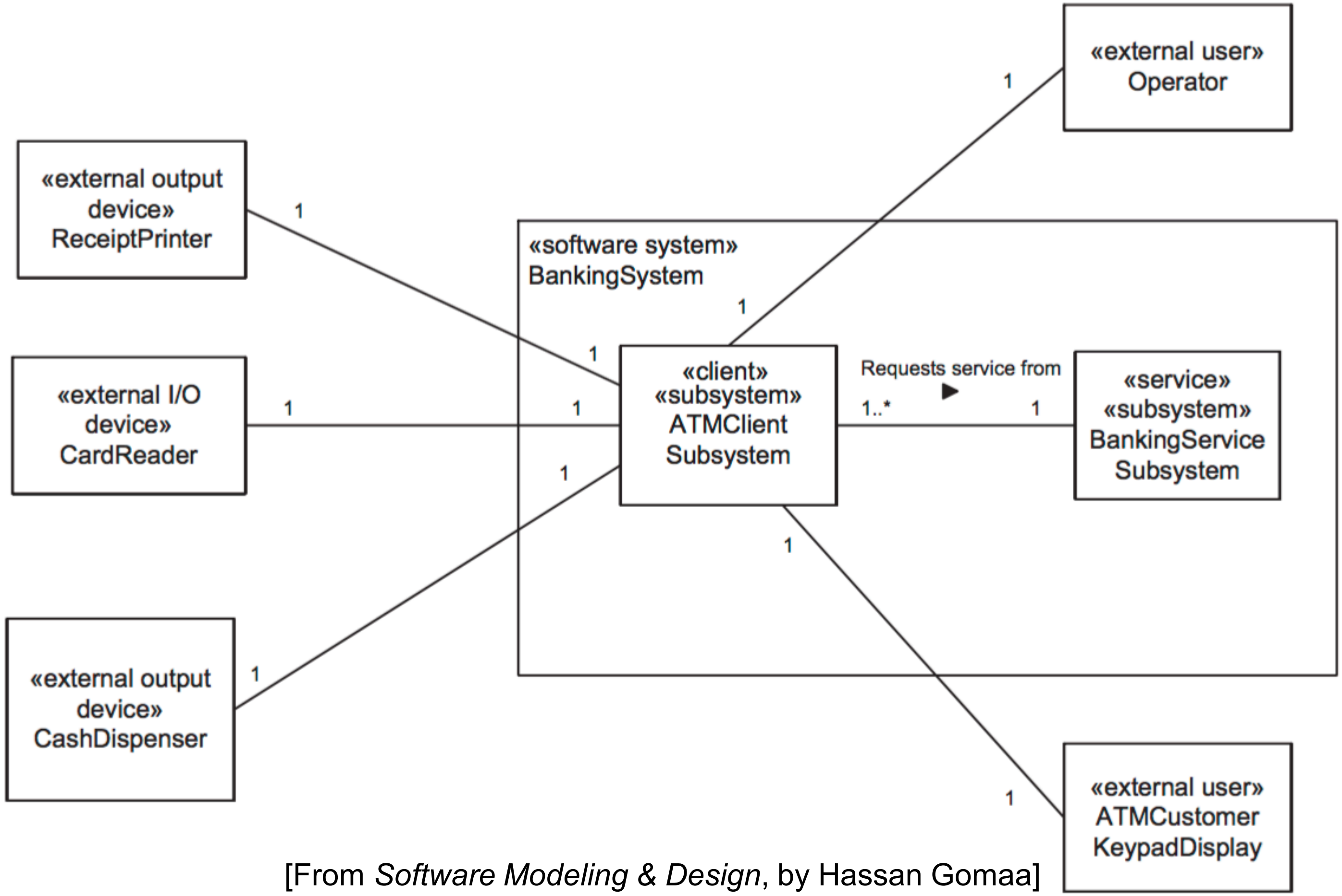
EXAMPLE 2 SOFTWARE ARCHITECTURE STRUCTURE

Web based enterprise software



EXAMPLE 3 SOFTWARE ARCHITECTURE

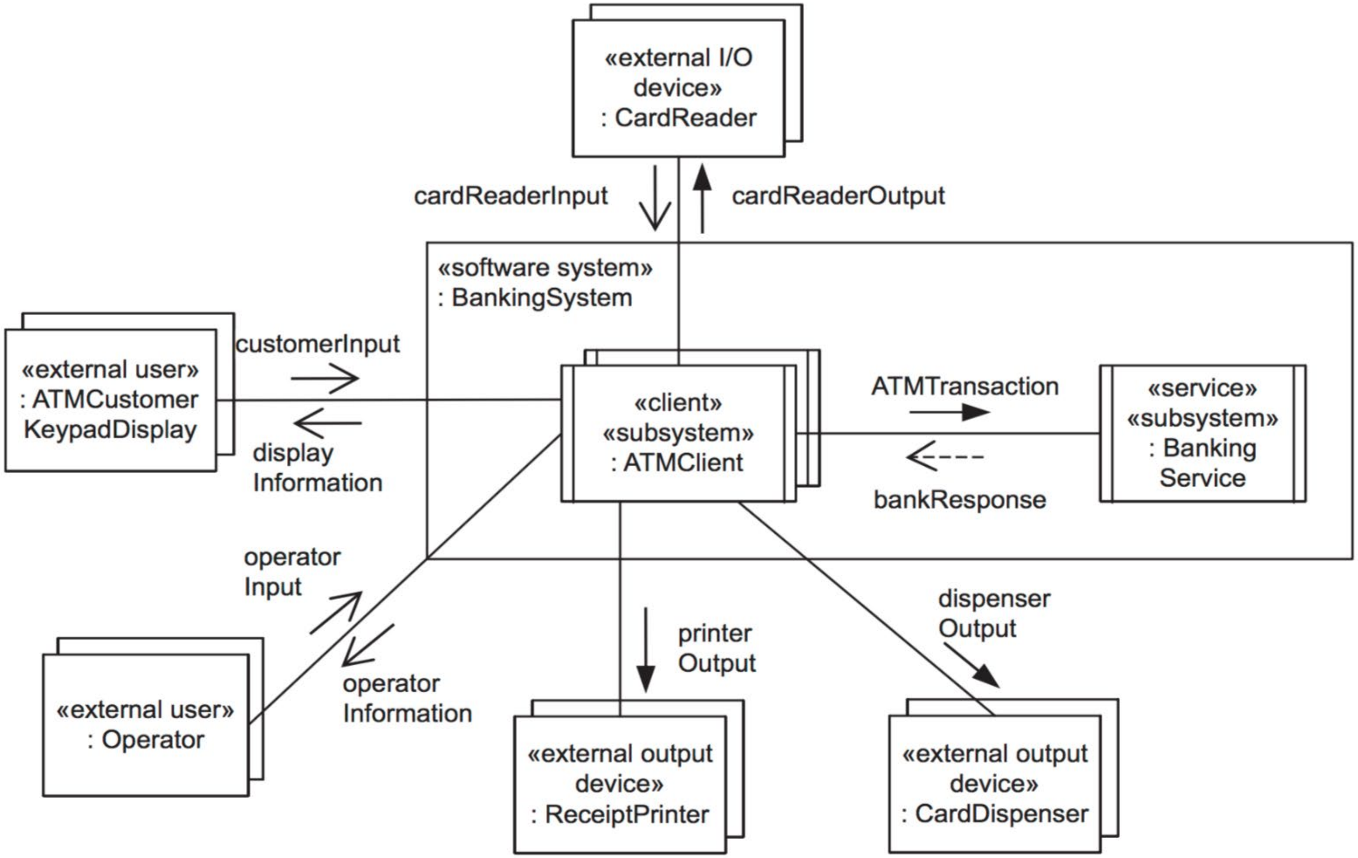
Architecture of a Banking System – structural view



[From *Software Modeling & Design*, by Hassan Gomaa]

EXAMPLE 3 SOFTWARE ARCHITECTURE (CONTINUED)

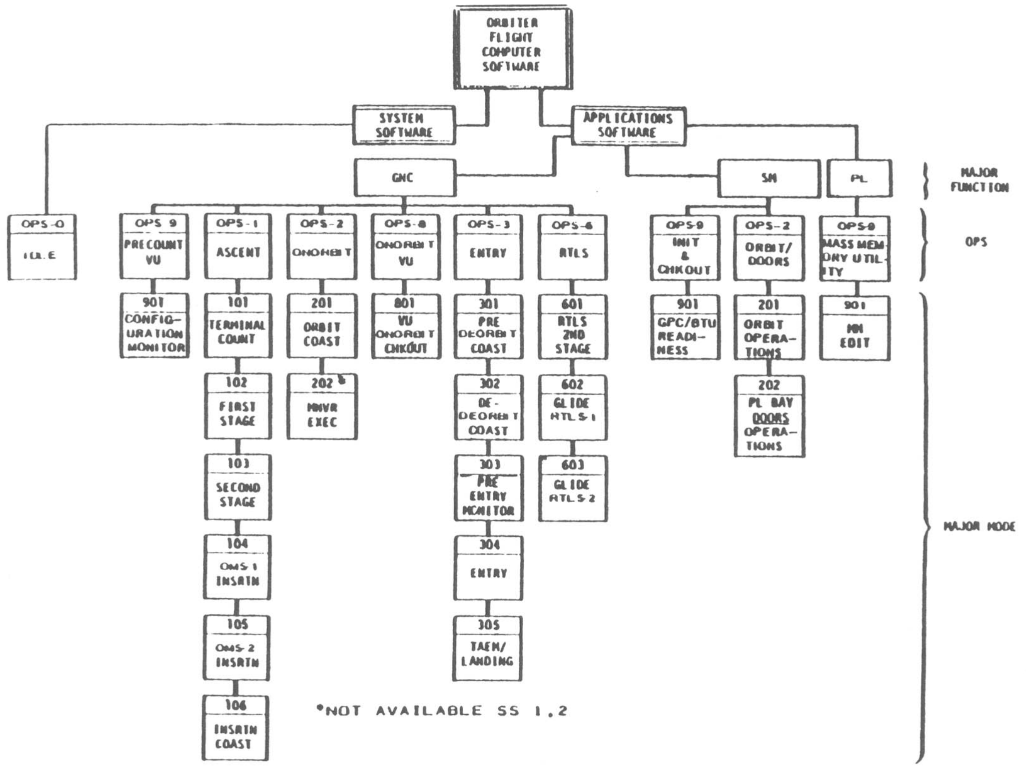
Architecture of a Banking System – dynamic view



[From *Software Modeling & Design*, by Hassan Gomaa]

EXAMPLE 4 SOFTWARE ARCHITECTURE

Space Shuttle flight computer software architecture



[From: Computers in Spaceflight: The NASA Experience]

OUTLINE

- Introduction to Software Design
- Design definition and concepts
- Design challenges, principles, and process
- Software architecture definition, concepts
- Architecture stakeholders and their need for/use of design

We are here

- Architecture and Detailed design
- Class exercises: LMS major architecture elements
- Assignments

SOFTWARE ARCHITECTURE STAKEHOLDERS

Who uses the architecture?

- Developers
- Testers
- Project managers
- System builders
- Maintainers
- Customers

WHAT IS THE NEED FOR ARCHITECTURE?

Preliminary architecture - in early phases (concept development and requirements):

- Assess product feasibility
- Convince stakeholders that their needs can be met
- Conduct tradeoff analyses
- Plan the project

USE OF SOFTWARE ARCHITECTURE BY PROJECT MANAGERS

- Estimate and assign budget and resources for architectural elements (components)
- Coordinate team of architects, developers, testers
- Assign people with the right skills to the right components
- Track project progress based on components
- Decide whether to acquire (COTS, GOTS, open source) or develop components

- Manage large projects through divide-and-conquer strategy

USE OF SOFTWARE ARCHITECTURE BY DEVELOPERS AND TESTERS

- Developers
  - In development - Implement elements based on software architecture
  - In maintenance and evolution - Understand software architecture before making changes to existing implementation

- Testers
  - Define test and integration strategy based on software architecture
    - E.g., should low-level components be tested first, or high-level components be tested first, or a mix
  - May have to write mock-test or stubs to unit-test individual components
    - Architecture shows dependencies to mock/stub

USE OF SOFTWARE ARCHITECTURE BY SYSTEM BUILDERS

- To compile, link, and produce executables
  - Develop Makefiles or build scripts - requires sound understanding of architecture
  - Can generate Makefiles (or build scripts) for each directory for a given architecture
  - Dependency rules are needed to compile the right set of files when source code evolves

OUTLINE

- Introduction to Software Design
- Design definition and concepts
- Design challenges, principles, and process
- Software architecture definition, concepts
- Architecture stakeholders and their need for/use of design
- We are here — Architecture and Detailed design
- Class exercises: LMS major architecture elements
- Assignments

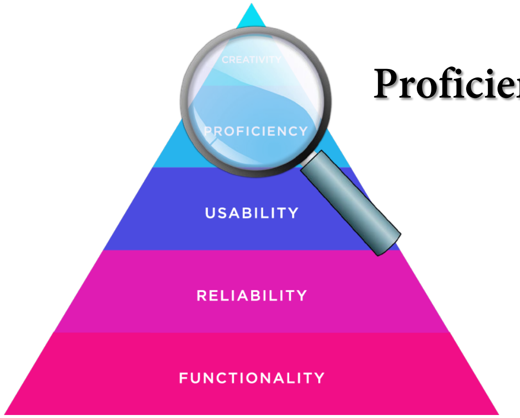
DETAILED DESIGN INFORMATION

- Decomposition
  - Sub-system parts or units
- Responsibilities
  - Data and behavior
- Interfaces
  - Public features
- Collaborations
  - Who does what when?
- Relationships
  - Inheritance, associations, etc.
- Properties
  - Performance, reliability, etc.
- States and Transitions, externally visible
- Packaging and Implementation
  - Scope, visibility, etc.
- Algorithms, Data Structures, and Types (Maybe)

ARCHITECTURE AND DETAILED DESIGN

Levels of Architecture:

1. Context / Operational Architecture
2. System / Solution Architecture
   a. Includes Software
3. Element-Level Architecture
   a. Module / Component
4. **Concrete Nodal Architecture**
   a. **Also known as “Design”**



Proficiency & Creativity

Architectural design

- The activity of specifying
  - A program’s **major parts (elements)**
  - Their responsibilities, properties, and interfaces, and
  - The **relationships** and **interactions** among them
- “High-level” design
- Global, upfront, major design decisions (e.g., technologies, configurations, OS, reuse, styles)
- More abstract

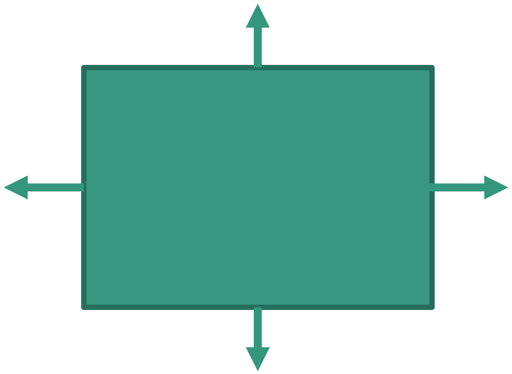
Detailed design

- The activity of specifying
  - The ***internal elements*** of **all major program parts (elements)**
  - Their structure, relationships, and processing, and
  - Often, their algorithms and data structures
- “Mid-level” and “low-level” design
  - Low level detailed design “shades” into coding
- Local decisions, made after architecture decisions
- More detailed

Architecture design suppresses (abstracts out) details of the “internals” of elements and focuses on the systemic/overall design

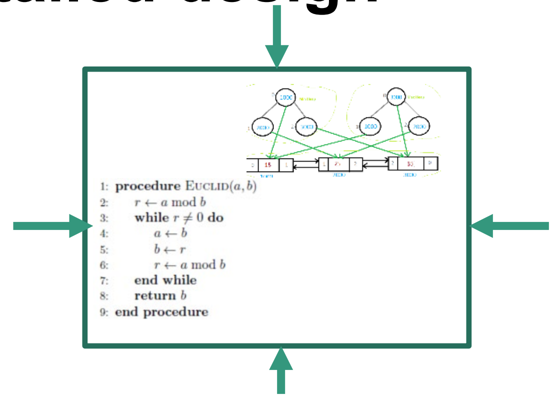
ARCHITECTURE AND DETAILED DESIGN

Architectural design



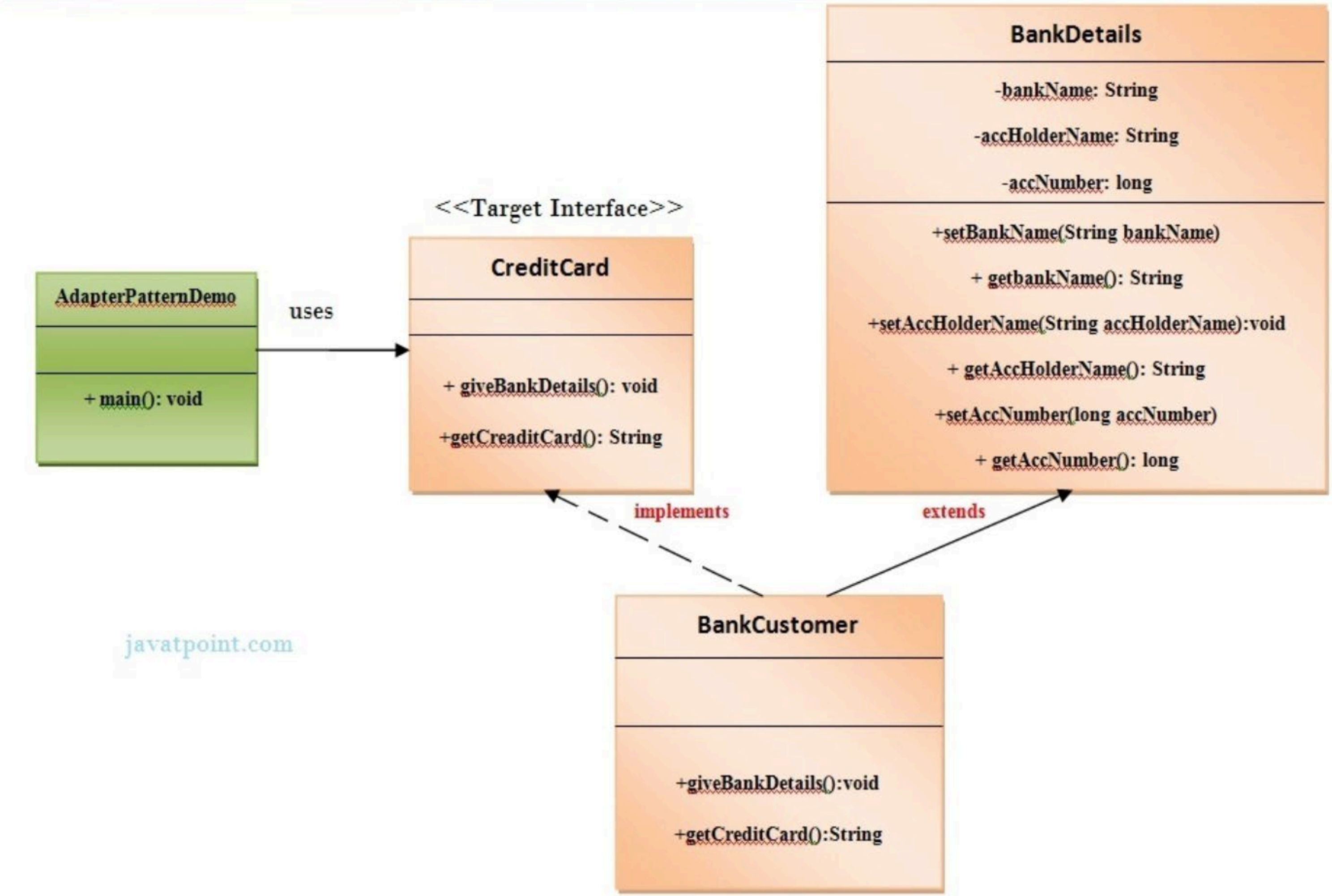
- For each element, focus outwardly:
  - Assign responsibilities
  - Systemic properties
  - Relations and interfaces
  - Behavior (within the system) and in interaction with other parts

Detailed design



- For each element, focus inwardly:
  - Element properties (data and behavior)
    - Data structures and algorithms
  - Details closer to implementation

EXAMPLE OF DETAILED DESIGN COMPONENTS



ARCHITECTURE AND DETAILED DESIGN

- The distinction between architectural and detailed design is not always clear, because it is not easy to define:
  - What is a “major” program part?
  - How abstract should architectural specifications be?
  - What is the architecture of a very small program?

SOFTWARE ARCHITECTURE DRIVERS

Architecture Level	Requirements Documents
Context / Operational Architecture	Stakeholder Requirements
System / Solution Architecture	System Specification
Element-Level Architecture	Element Specifications
Concrete Nodal Architecture (Design)	Build-To Specifications (aka Design)

Requirements

Functionality

Quality Attributes

Software Architecture

"Best practices"

Principles

Styles/patterns

Tactics

Frameworks

Constraints

Hardware

OS

…

Organizational

Structure

People (skills)

Assets (components, Product lines)

FACTORS THAT AFFECT DESIGN (A.K.A. “DRIVERS”)

- The features, functions, services, of the product
- The quality (“ilities”) of the product

The requirements to be satisfied by the design

- The value of the product
- The process of design
- Design best practices (principles, styles, tactics, frameworks)
- The consequence of design, i.e., the change to be brought about
- The people involved in the design process and their working relationships, including
  - Competence (knowledge and experience) of designers
  - Organizational structure (Conway’s Law: “organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations”); organizational culture; team cohesion and communication
- Existing assets (code libraries, reusable components, artifacts, product lines)
- Resource available to the design, development, and use of the product
- Constraints (technology, hardware, operating system, standards, regulations)

DESIGN CHOICES

- Identify problems
- Identify candidate solutions
- Perform tradeoff analysis
  - A **trade-off** (or **tradeoff**) is a situation that involves losing one quality or aspect of something in return for gaining another quality or aspect
  - A trade study or trade-off study is the activity […] to identify the most balanced technical solutions among a set of proposed viable solutions (FAA, *System Engineering Manual Version 3.1, Section 4.6, Trade Studies*). These viable solutions are judged by their satisfaction of a series of measures or cost functions.
- Select best suited solution

EXAMPLE SOLUTIONS / CHOICES

- Develop vs reuse vs buy
  - Tradeoff considerations: cost, time to develop, quality, fit for purpose, quality attributes, risk

- Technology (e.g., frameworks, OS, data persistence)
  - Tradeoff considerations: cost, time to develop, quality, fit for purpose, quality attributes, required expertise, risk

- Tactics, styles
- Deployment

TRADE-OFF ANALYSIS TEMPLATE/EXAMPLE

	Factor1	Factor2	Factor3	Factor4	Factor5	Total
	Weight (%)	Weight (%)	Weight (%)	Weight (%)	Weight (%)	
	w1	w2	w3	w4	w5	100
Decision1						
Option1	RatingforFactor					Weighted sum
Option2						
…..						
Decision2						
Option1						
Option2						
…..						

E.g.
Decision1: front end framework
Option1: Angular
Option2: React
Option3: VueJS
Decision2: data base management system
Option1: Postgres
Option2: MySQL
Decision3: Hosting (self, cloud)
…
Decision4 : back-end technology
…

E.g., **Factors**: Cost, Team experience with the technology, …
Factor weight: percent showing the relative importance of the factor (add up to 100 for all factors)

Rating for Factor = How the *option* on the row rates against the *factor* on the column
Rating Value –> Scale (e.g., 0-5)
Document the rationale for the rating

$$\sum_{\text{All factors}} (\text{RatingForFactor} * \text{Factor Weight})$$


ASK
learn
question
QUESTION
ask


MARYLAND APPLIED
GRADUATE ENGINEERING
THE A. JAMES CLARK SCHOOL of ENGINEERING
UNIVERSITY OF MARYLAND

CLASS EXERCISE – MAJOR ARCHITECTURE ELEMENTS

- Identify identify major architecture elements and relations for your LMS

SUMMARY

- Design in engineering
- Software design definitions and concepts
- Software design challenges, principles, and process
- Software architecture – definition, concepts, stakeholders, drivers
- Architecture and detailed design similarities and differences

OUTLINE

- Introduction to Software Design
- Design definition and concepts
- Design challenges, principles, and process
- Software architecture definition, concepts
- Architecture stakeholders and their need for/use of design
- Architecture and Detailed design

We are here

Class exercises: LMS major architecture elements

- Assignments

DESIGN INTRO QUIZ

- Software design concepts, principles, process, artifacts
- Individual assignment
- Online

SOFTWARE DEVELOPMENT PROJECT ASSIGNMENT

- Tasks:
  - Firm up your LMS’s features (if any different than what was identified in the Requirements Analysis activity)
  - Start developing a preliminary LMS architecture
    - Identify major design decisions
    - Identify and perform trade-off analysis for candidate technologies (toolkits, frameworks, APIs, files vs DBs, DBMS, etc.); select one technology (set of technologies) and document your decision
    - Identify (major) architectural elements, their responsibilities, and their relations

- Team assignment
- Online submission (*Preliminary architecture decisions*)
- Not graded, but mandatory

CONSIDERATIONS

- How well the options serve the purpose of your system
- The advantages and disadvantages of your options
- How well they work together
- What are the risks of choosing a certain technology (or a combination of technologies)


QUESTION
ASK
ask
learn
question


MARYLAND APPLIED
GRADUATE ENGINEERING
THE A. JAMES CLARK SCHOOL of ENGINEERING
UNIVERSITY OF MARYLAND