Course Introduction
An Overview of System Analysis and Design

System Analysis and Design

School of Computer Science and Technology, Tongji University

Index

- Course Overview
- Assessment
- Reference materials and modeling tools
- Main topics
- Overview of System Analysis and Design
  - Needs on analysis and design
  - Key Roles involved and major challenges
  - Development Process
  - Establishing System Perspective
  - What to model: A System Vision
- Visual Modeling Examples

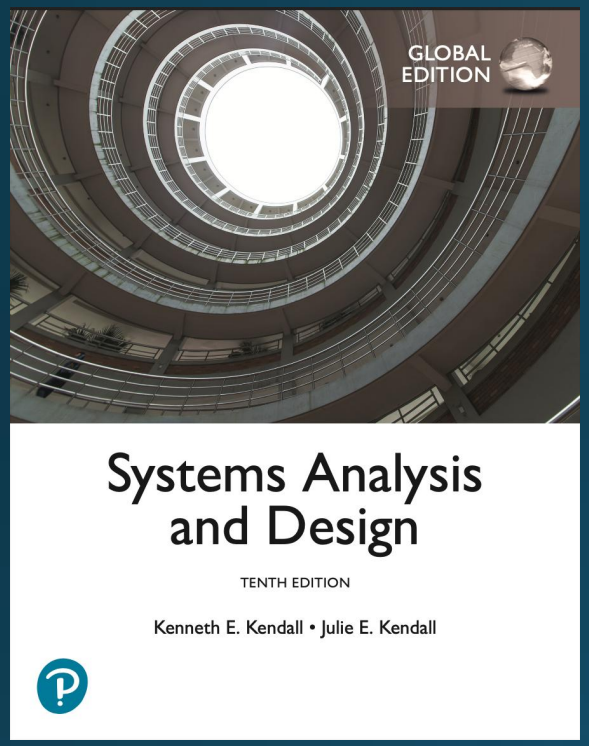
Course Overview

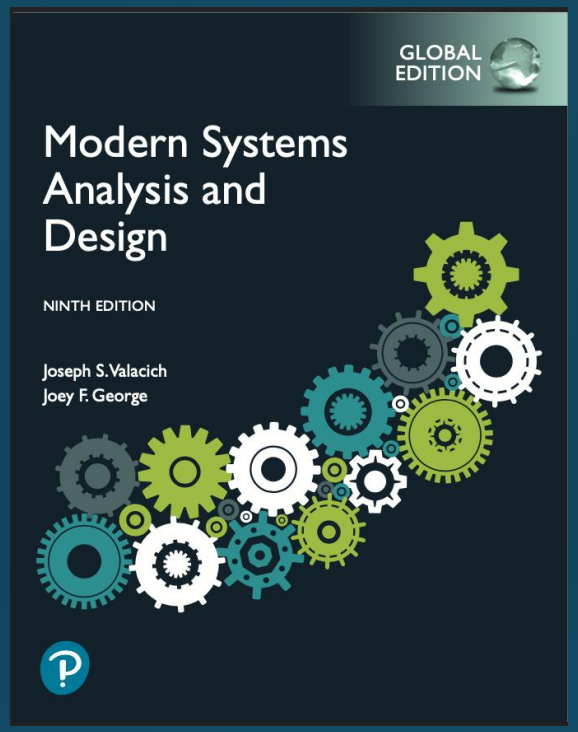
- This course covers **the essential skills of system analysis and design** including requirements analysis, design principles, architecture design, object-oriented design and domain driven design.
  - It emphases the ever changing nature of software requirements and key analysis and design techniques to identify and handle those changes.
  - Both Object-Oriented Methodologies and Agile Approaches are introduced.
  - Best practices in Software System Architecting, Design and Development will be discussed.
- How to apply analysis and design principles and techniques with iterative development will be demonstrated in this course.
- The design of AI powered solutions will also be explored and discussed.

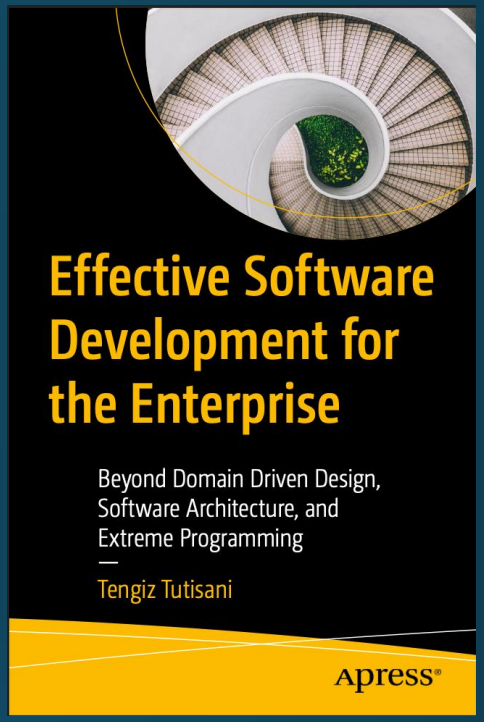
Assessment

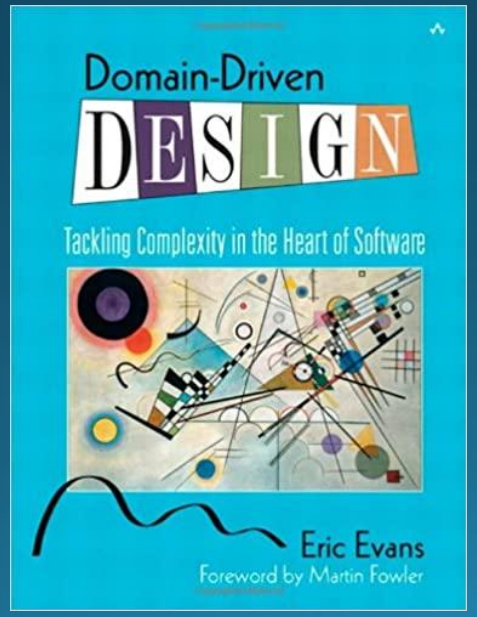
- Attendance, *10%*
- Team Project (no more than 4 members), *90%*
  - Project vision, scope and essential requirements will be released in Week 3.
  - Software Requirements Specification; 30%
  - Analysis Model and Middle-Term presentation; 30%
  - Design Model, Final presentation and qualities of final deliveries; 30%

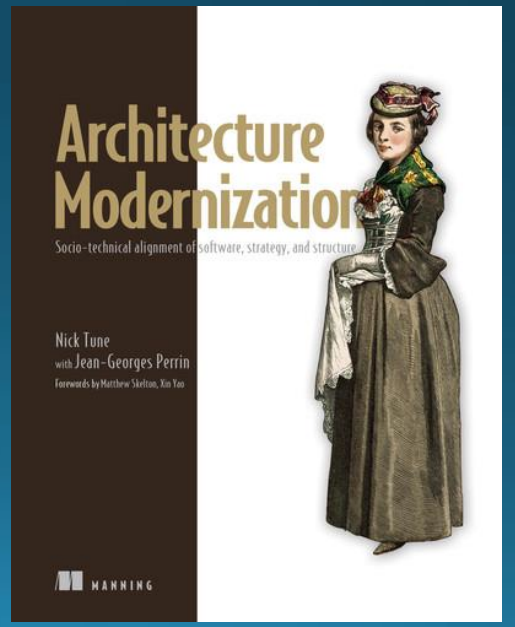
Reference Books

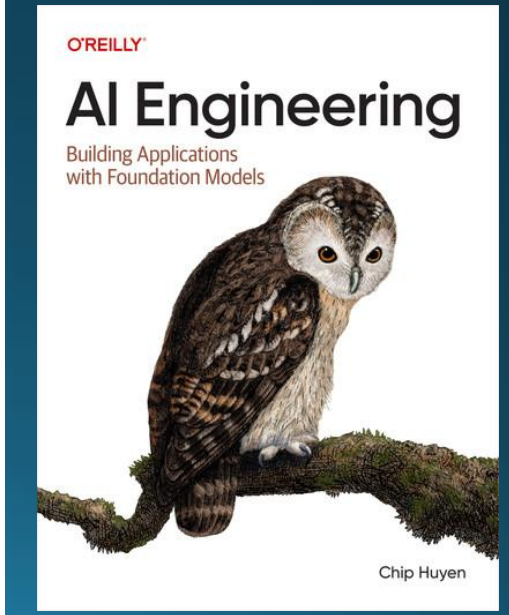






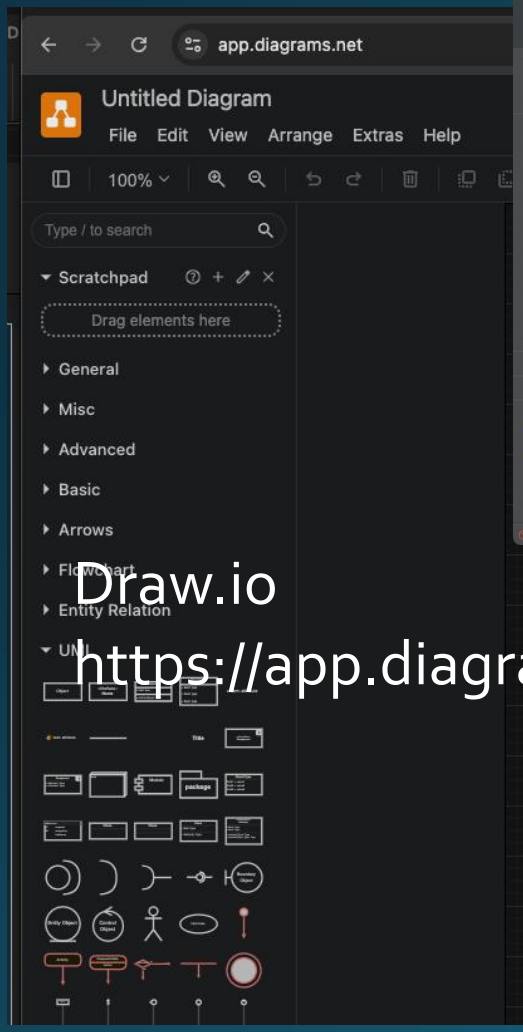






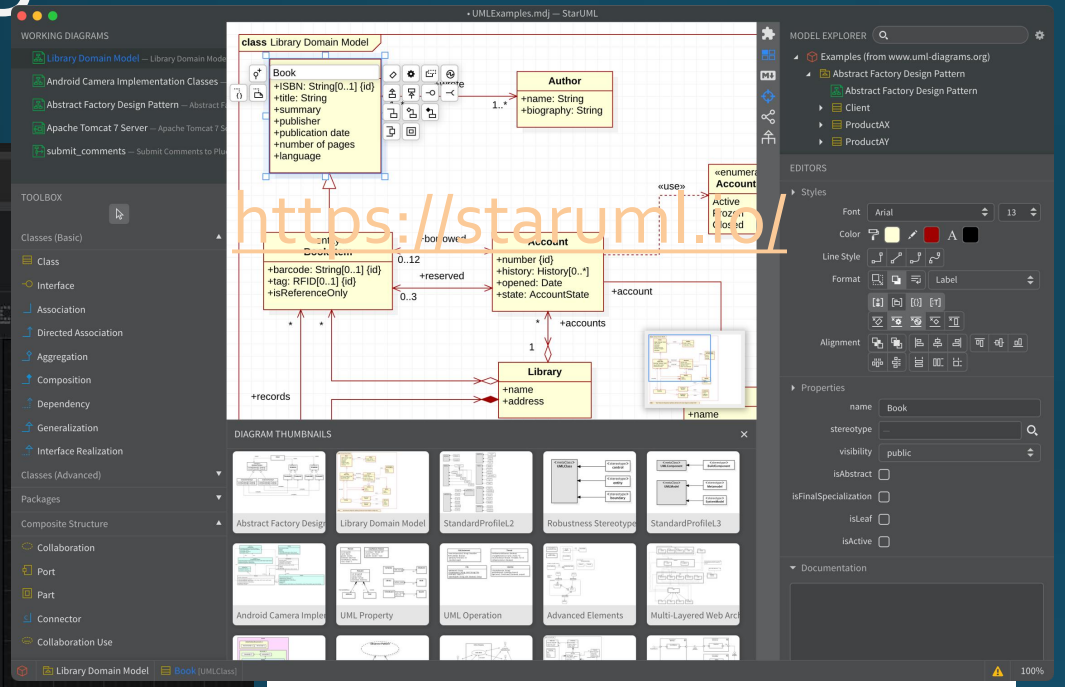
Many online resources…

Modeling Tools and Collaboration Environments

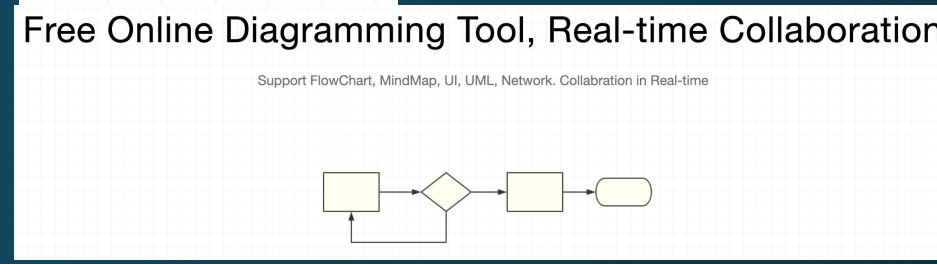


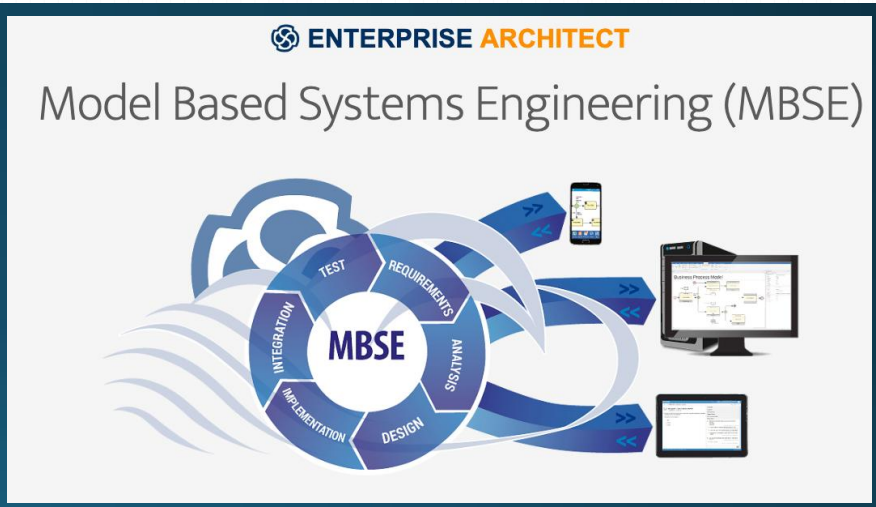
Draw.io

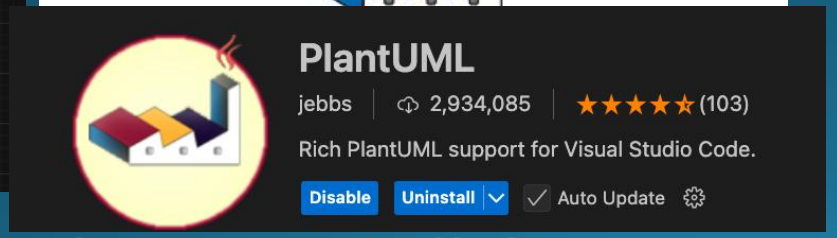
https://app.diagrams.net/



https://staruml.io/







Many online team collaboration tools are also useful for this course

Lecture Topics (1)

1. Course introduction & An Overview of System Analysis and Design
2. Project Management and Agile Approaches
3. OOAD, UP and UML
4. Requirements Analysis and Use Case Modeling
5. More on Agile Modeling
6. Moving into Design (SRS Due)
7. Preliminary Architecture Design
8. Refining the Design
9. Design Principles

Lecture Topics (2)

10. Middle-Term Presentation
11. Architecture Styles and Patterns
12. Design for Deployability and Operation Excellence
13. Embracing New Technology
14. Discussion and Review
15. Final Presentation

Share Your Experiences

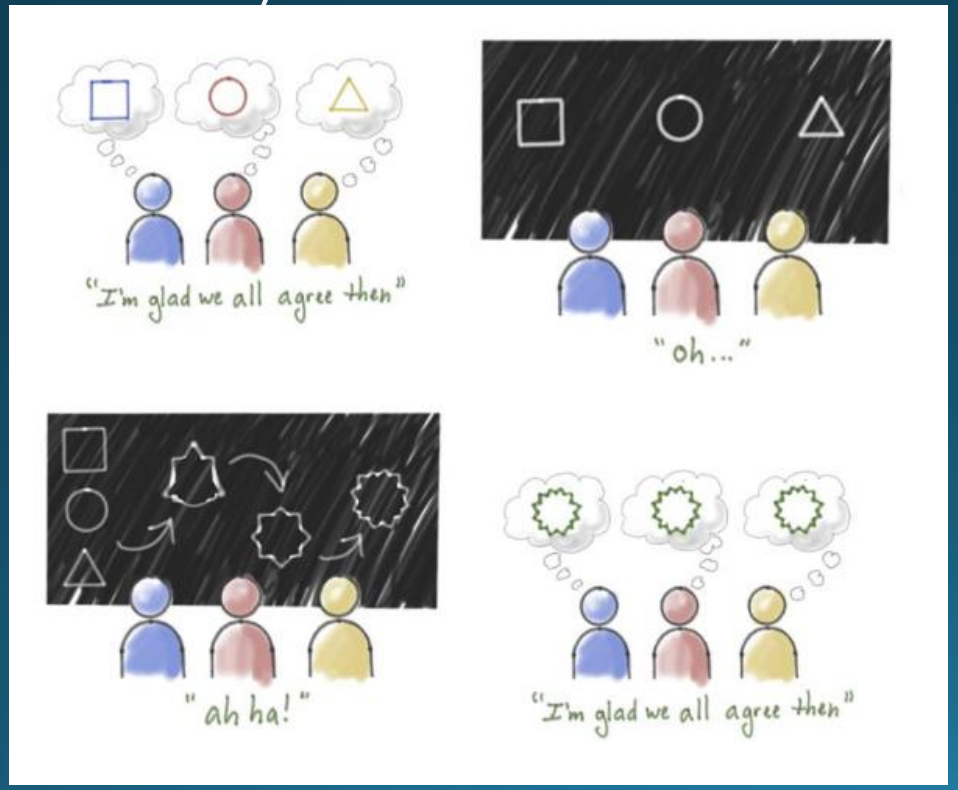
- What programming languages can you work with? Please describe your experience with them.
- What's the most challenging thing about working as a programmer?
- What's your current understanding of system analysis and design?
- Will software developers be replaced by AI in the near future? What is your opinion?

Think – Pair –Share…

Think: Defining the Problem + Solving a Problem

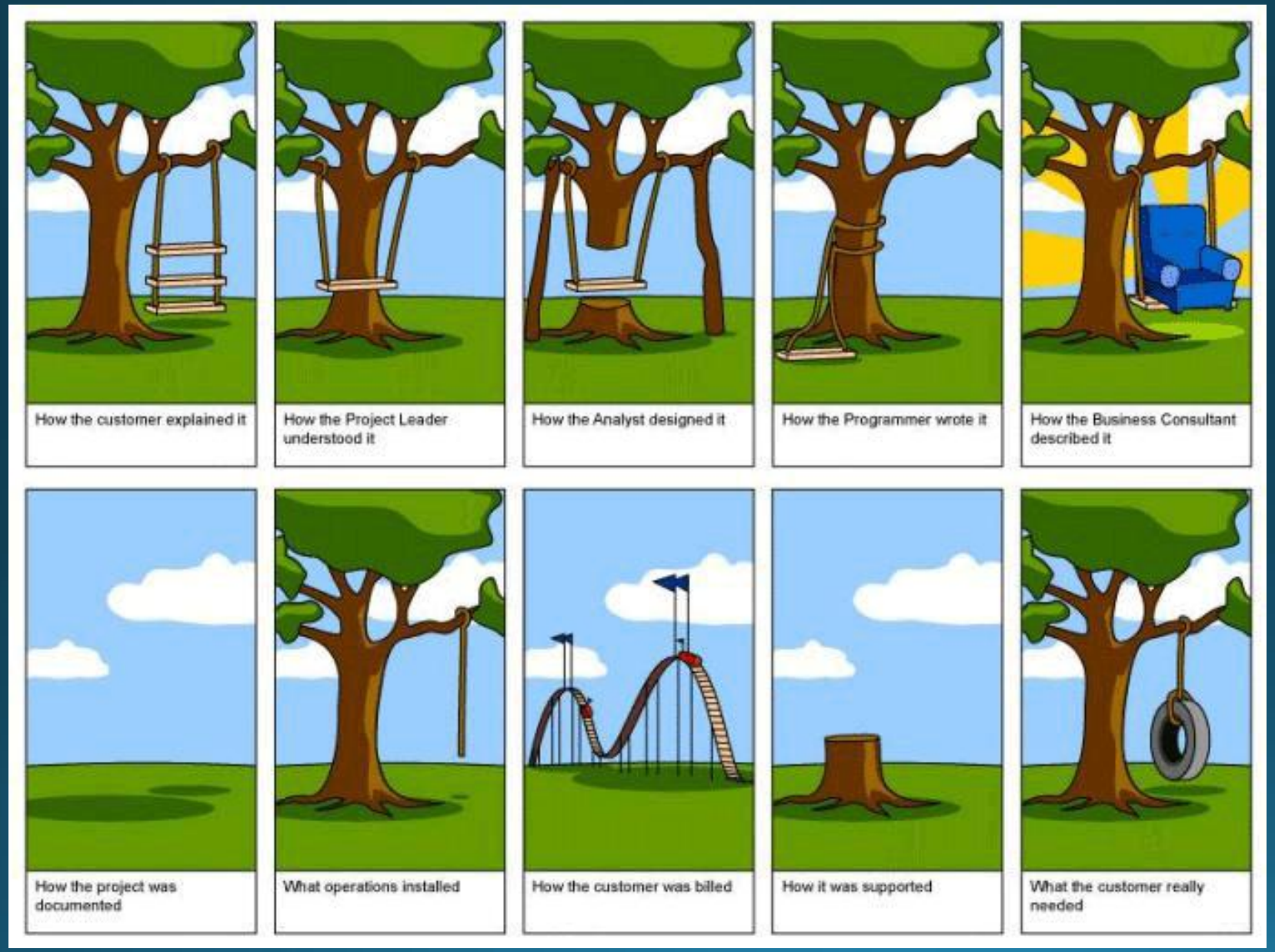
Needs for System Analysis and Design

- Delivering a system without proper planning leads to great user dissatisfaction and frequently causes the system to fall into disuse
- Lends structure to the analysis and design of information systems
- A series of processes systematically undertaken to improve a business through the use of computerized information systems



Cartoon by Barrett illustrating the importance of shared understanding

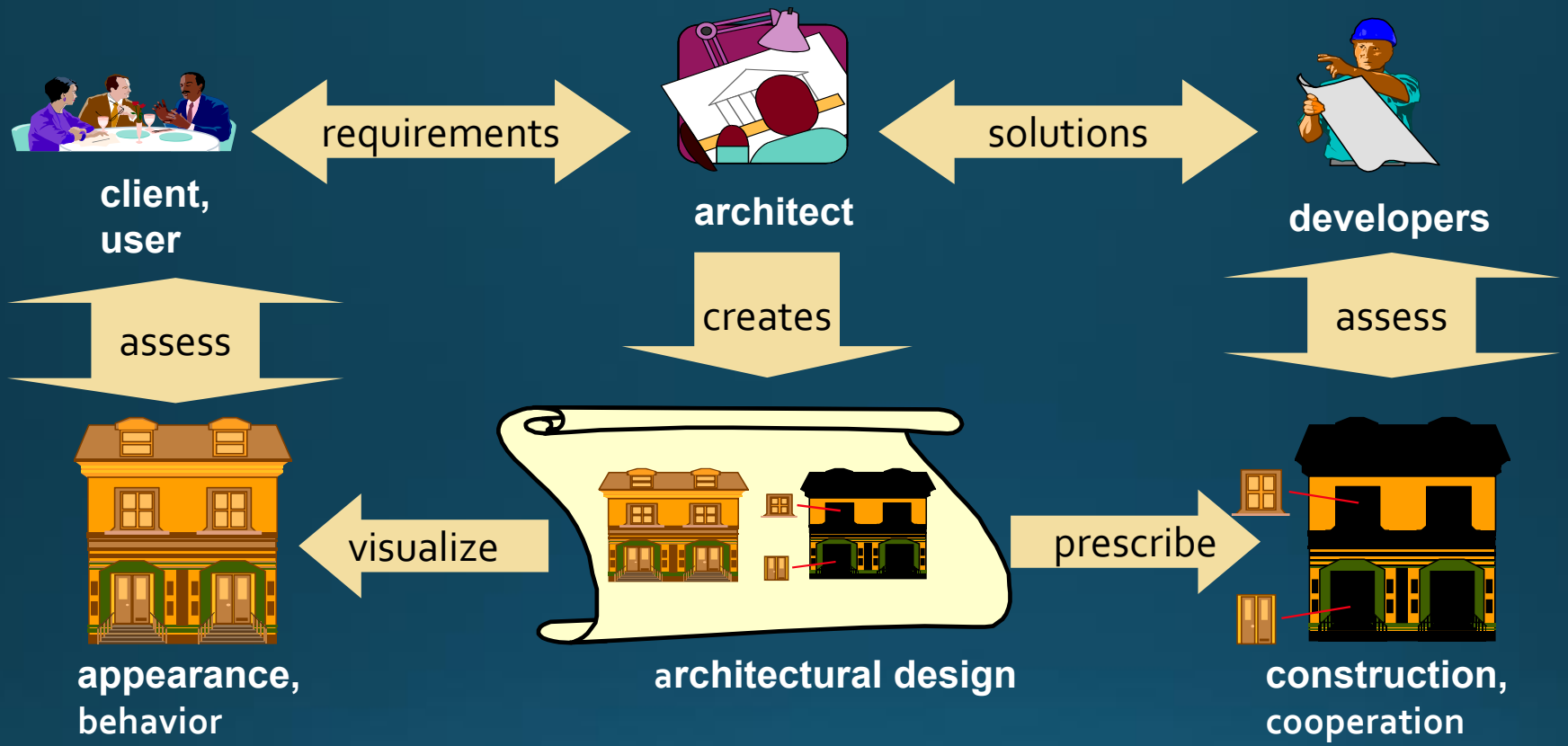
Comic : Software Development Process



Roles of a Systems Analyst

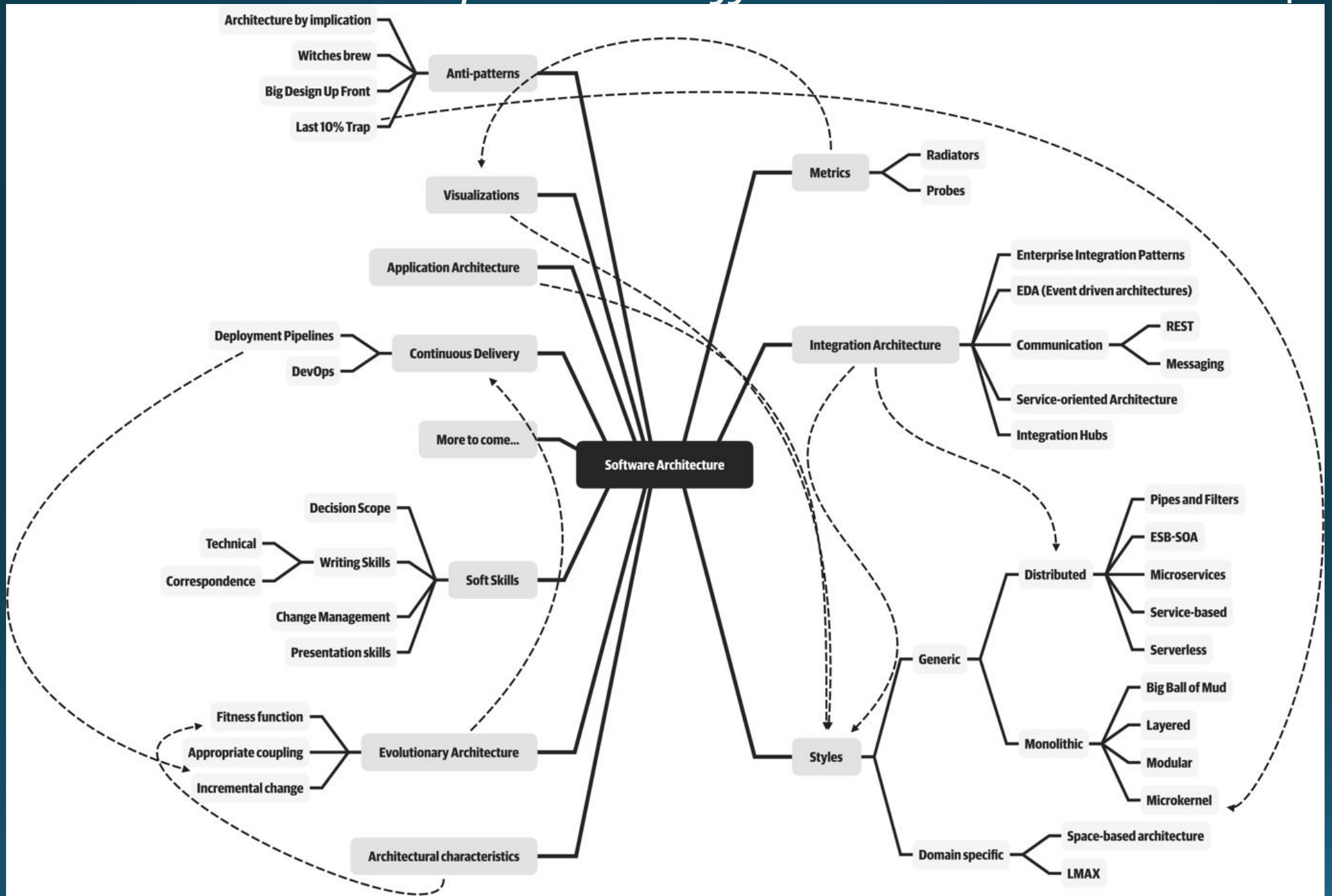
- The analyst must be able to work with people of all descriptions and be experienced in working with computers
- Three primary roles:
  - Consultant
  - Supporting expert
  - Agent of change
- Qualities
  - Problem solver
  - Communicator
  - Strong personal and professional ethics
  - Self-disciplined and self-motivated

Software Architect

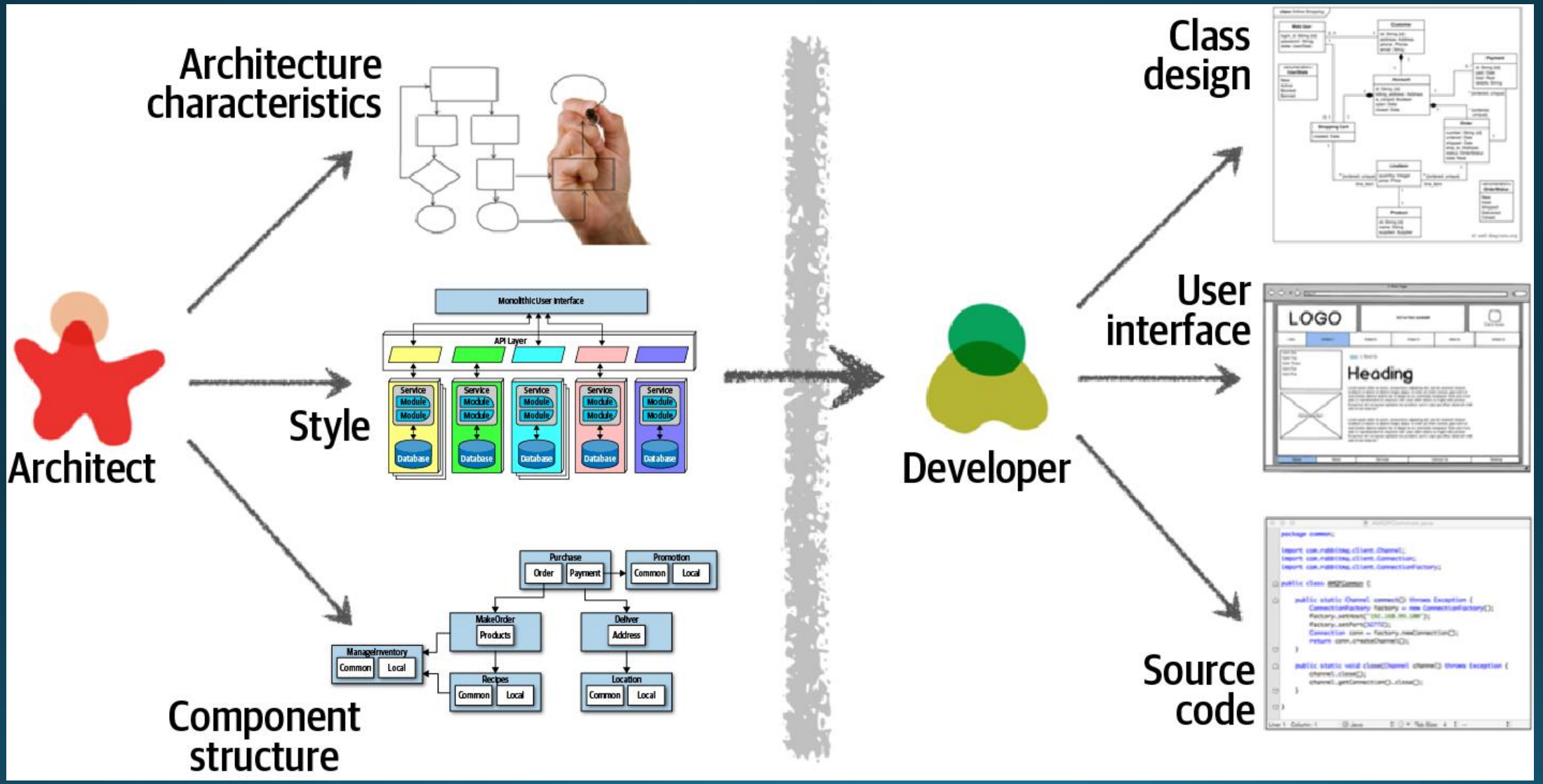


- A successful architect must stand on three "legs":
  - *Skill:* The foundation for practicing architects. It requires knowledge and the ability to apply it to solve real problems.
  - *Impact:* The measure of how well an architect applies his or her skill to benefit the company.
  - *Leadership:* Determines whether an architect advances the state of the practice.

Architecture is about the important stuff…whatever that is. Ralph Johnson

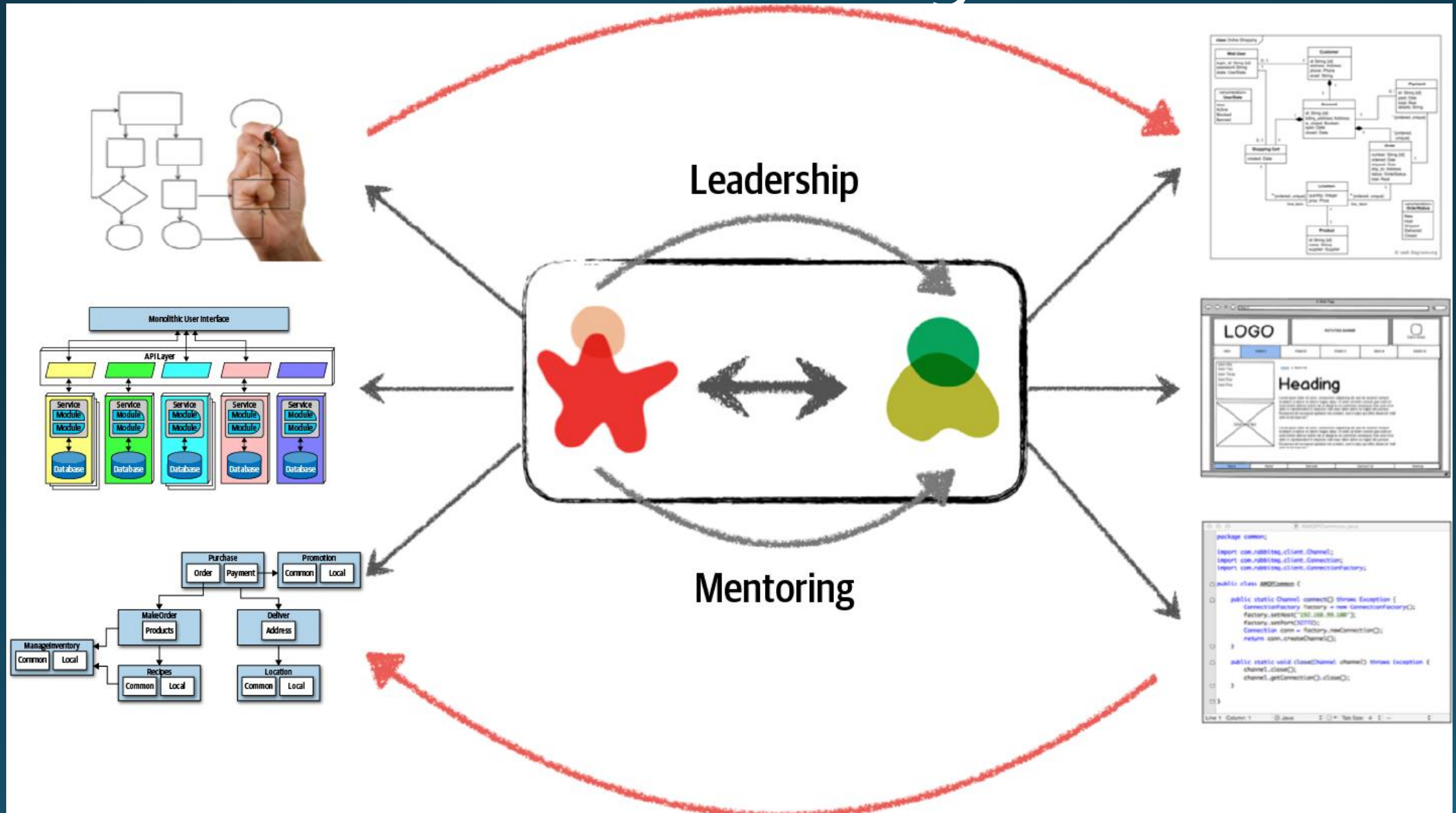


Architecture Versus Design

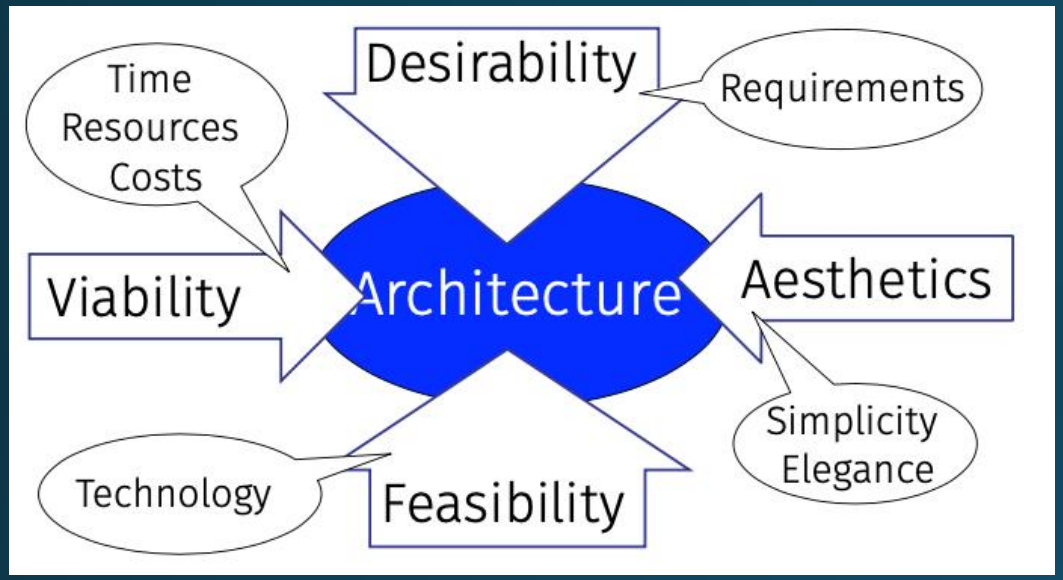


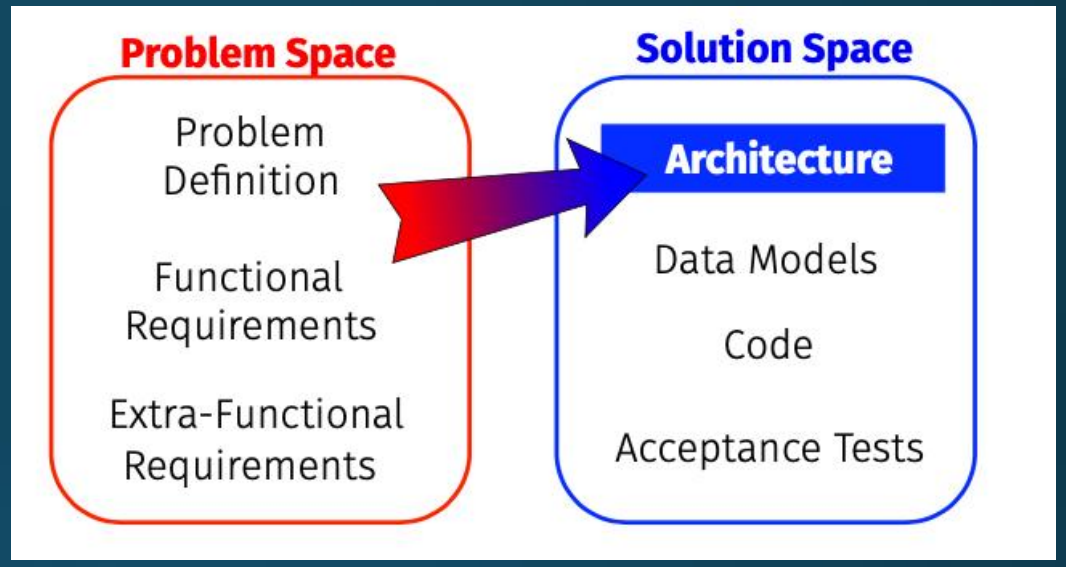
Traditional view of architecture versus design

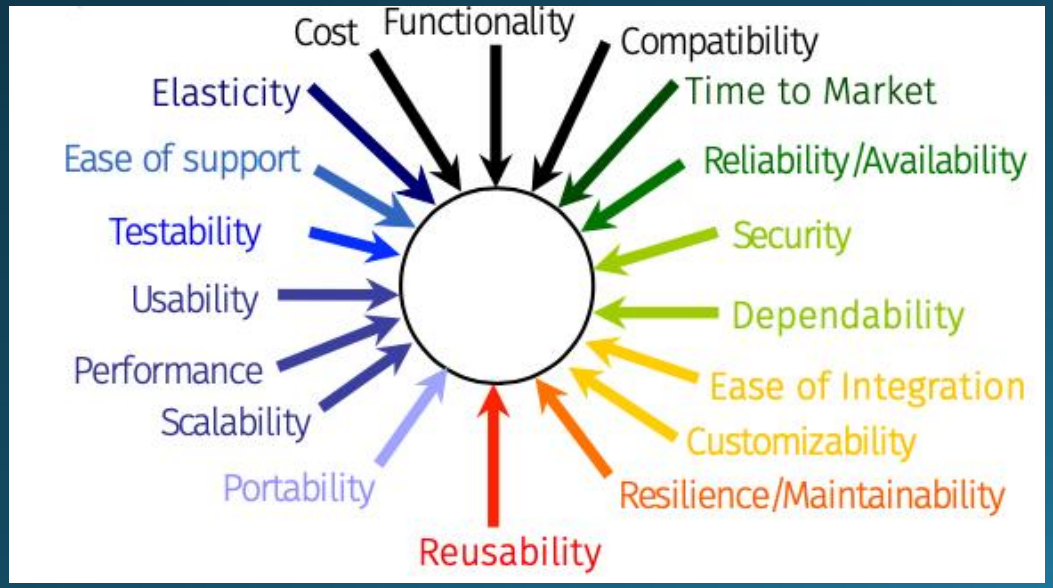
Making Architecture Work Through Collaboration

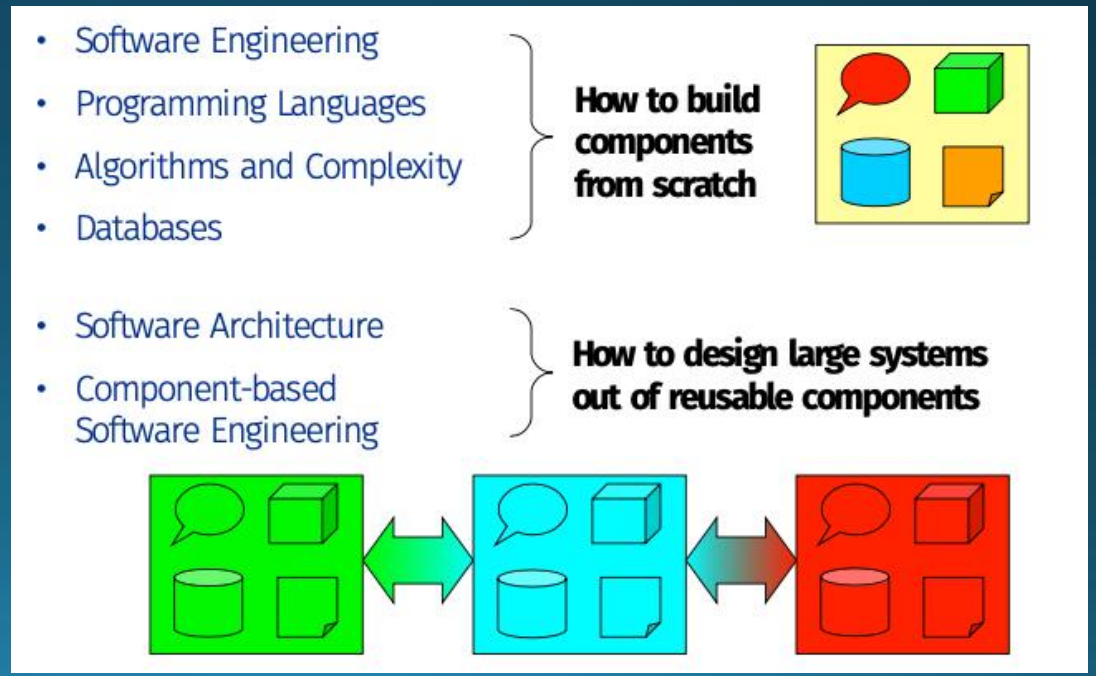


Challenges of Designing an Architecture



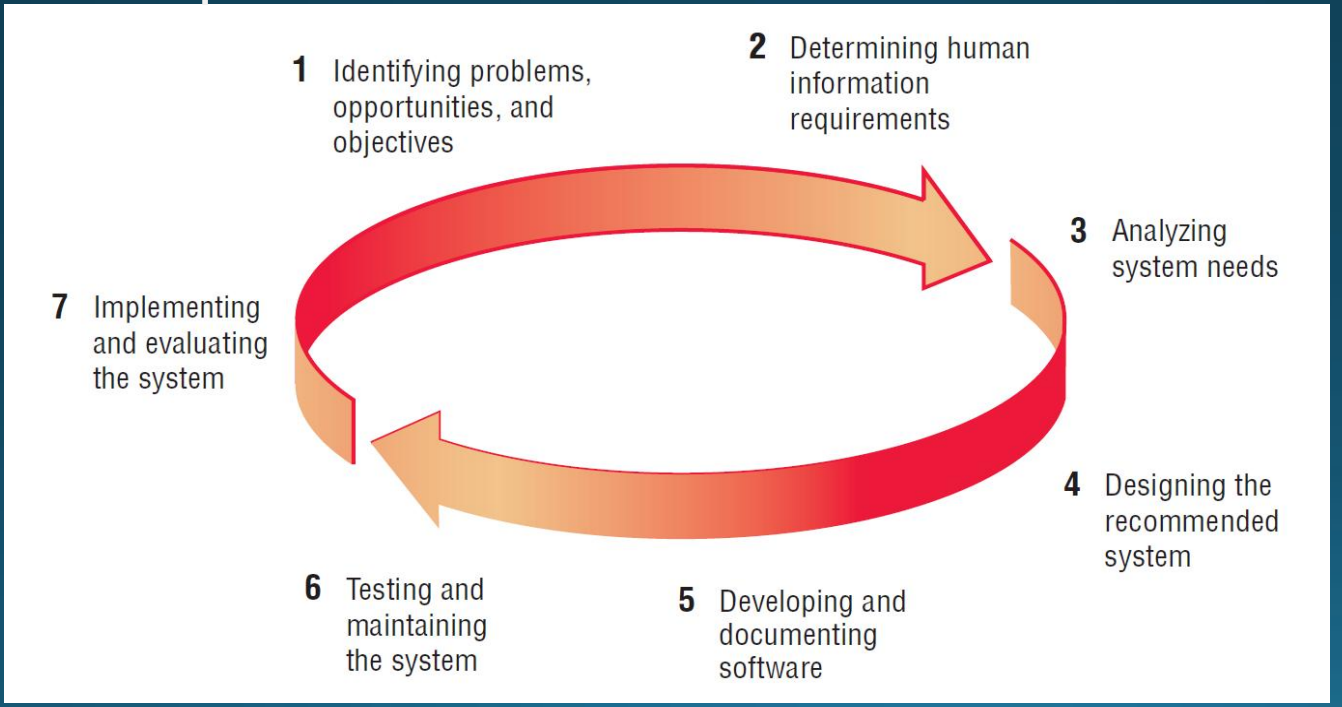






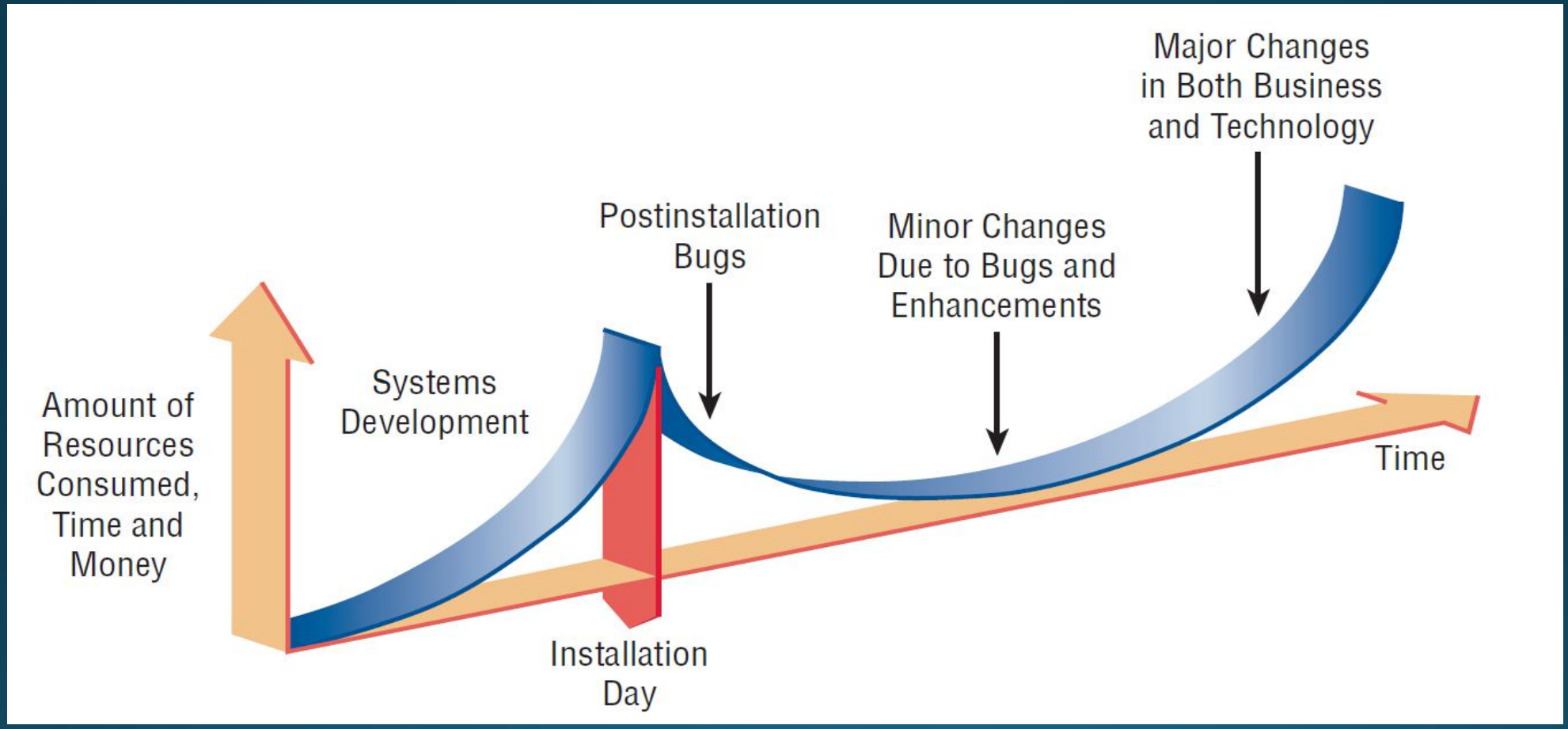
System Development Life Cycle(SDLC)

- The systems development life cycle is a phased approach to solving business problems
- Developed through the use of a specific cycle of analyst and user activities
- Each phase has unique user activities



The Seven Phases of the Systems Development Life Cycle

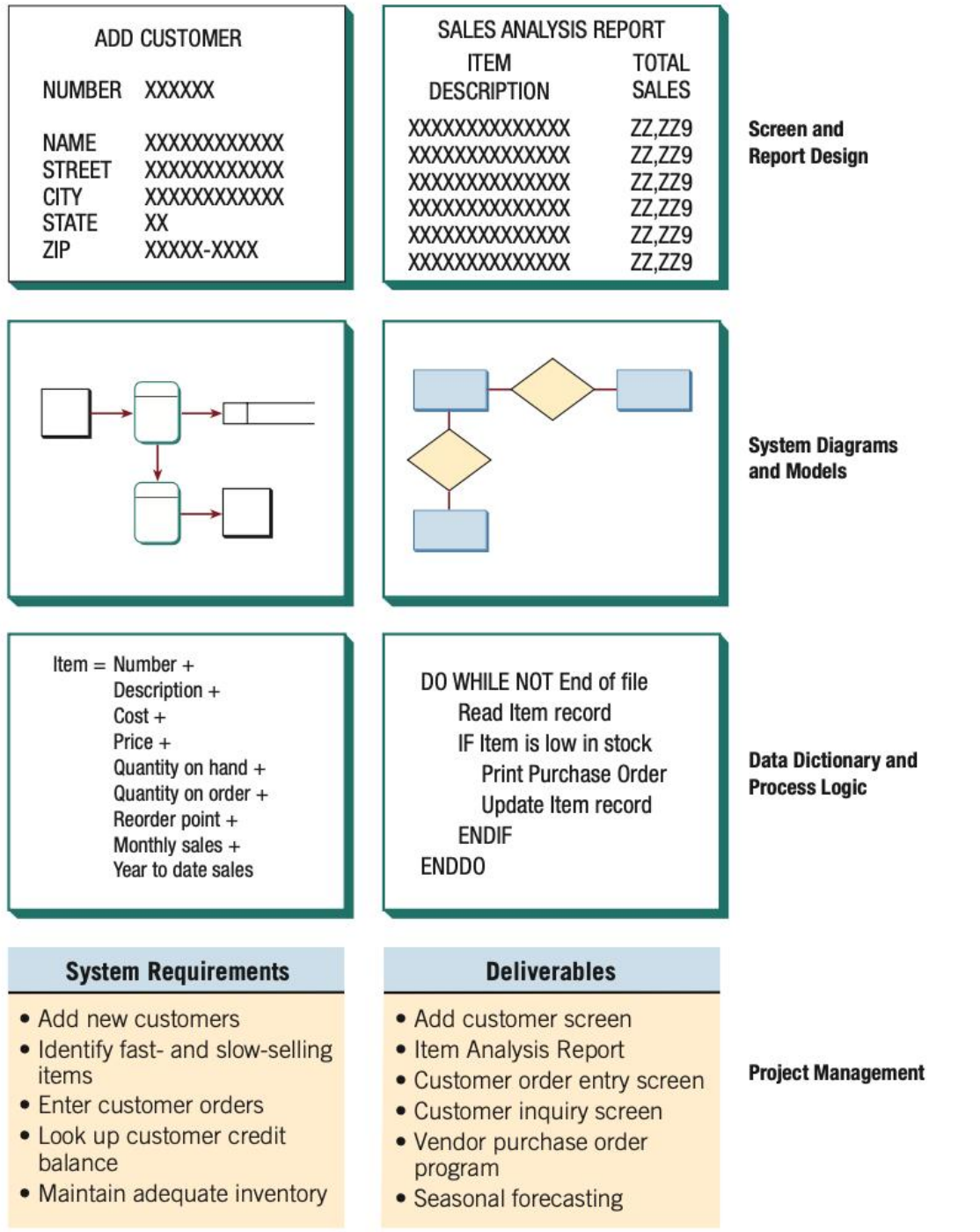
Resource Consumption over the System Lifecycle



Eventually maintenance exceeds the cost of a creating a new system.
At that point a new systems study should be untaken.

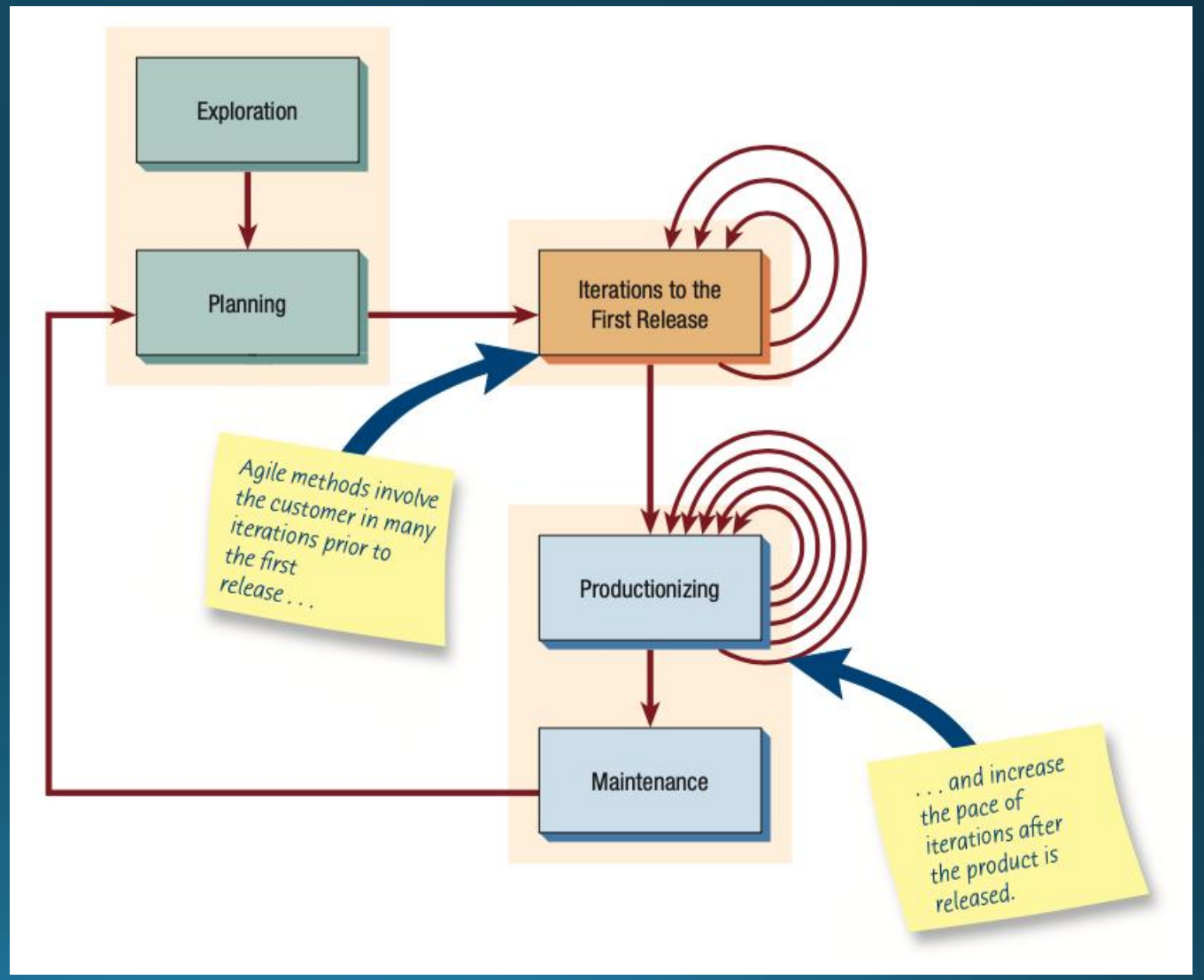
Using CASE Tools

- CASE tools are productivity tools for systems analysts that have been created explicitly to improve their routine work through the use of automated support



The Agile Approach

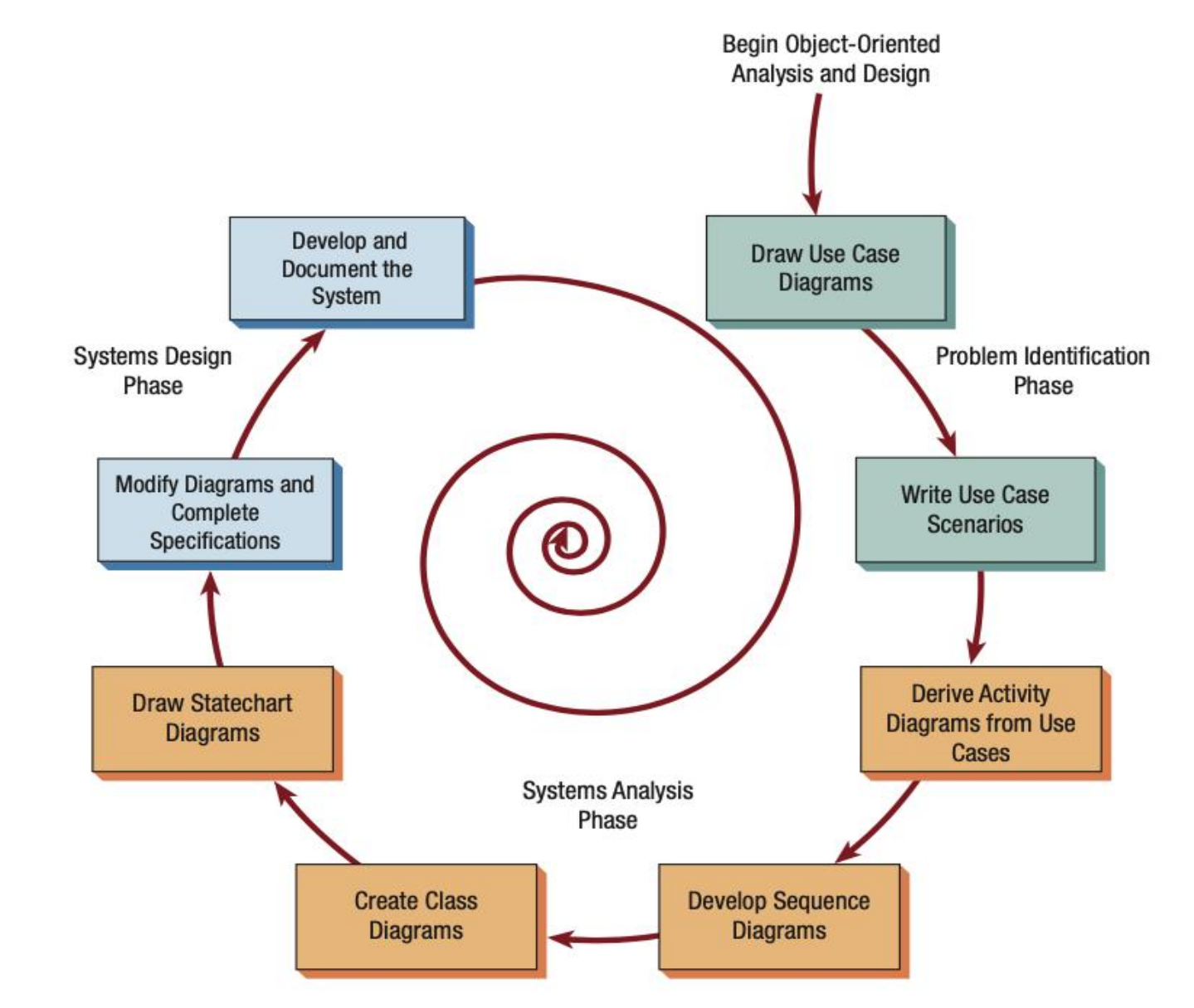
- Based on
  - Values
    - Communication
    - Simplicity
    - Feedback
    - Courage
  - Principles
  - Core Practices
- Five Stages
  - Exploration
  - Planning
  - Iterations to the first release
  - Productionizing
  - Maintenance



Object-Oriented Systems Analysis and Design

- Alternate approach to the structured approach of the SDLC that is intended to facilitate the development of systems that change rapidly in response to dynamic business environments
- Analysis is performed on a small part of the system followed by design and implementation
- The cycle repeats with analysis, design, and implementation of the next part and this repeats until the project is complete
- Examines the objects of a system

The Typical Steps in OOAD with UML



Choosing Which System Development Method to Use

Choose	When
The Systems Development Life Cycle (SDLC) Approach	• systems have been developed and documented using SDLC • it is important to document each step of the way • upper-level management feels more comfortable or safe using SDLC • there are adequate resources and time to complete the full SDLC • communication of how new systems work is important
Agile Methodologies	• there is a project champion of agile methods in the organization • applications need to be developed quickly in response to a dynamic environment • a rescue takes place (the system failed and there is no time to figure out what went wrong) • the customer is satisfied with incremental improvements • executives and analysts agree with the principles of agile methodologies
Object-Oriented Methodologies	• the problems modeled lend themselves to classes • an organization supports the UML learning • systems can be added gradually, one subsystem at a time • reuse of previously written software is a possibility • it is acceptable to tackle the difficult problems first

Open Source Software

- Four Types of open source communities
  - Ad hoc
  - Standardized
  - Organized
  - Commercial
- Dimensions that differentiate open source communities
  - General structure
  - Environment
  - Goals
  - Methods
  - User community
  - Licensing

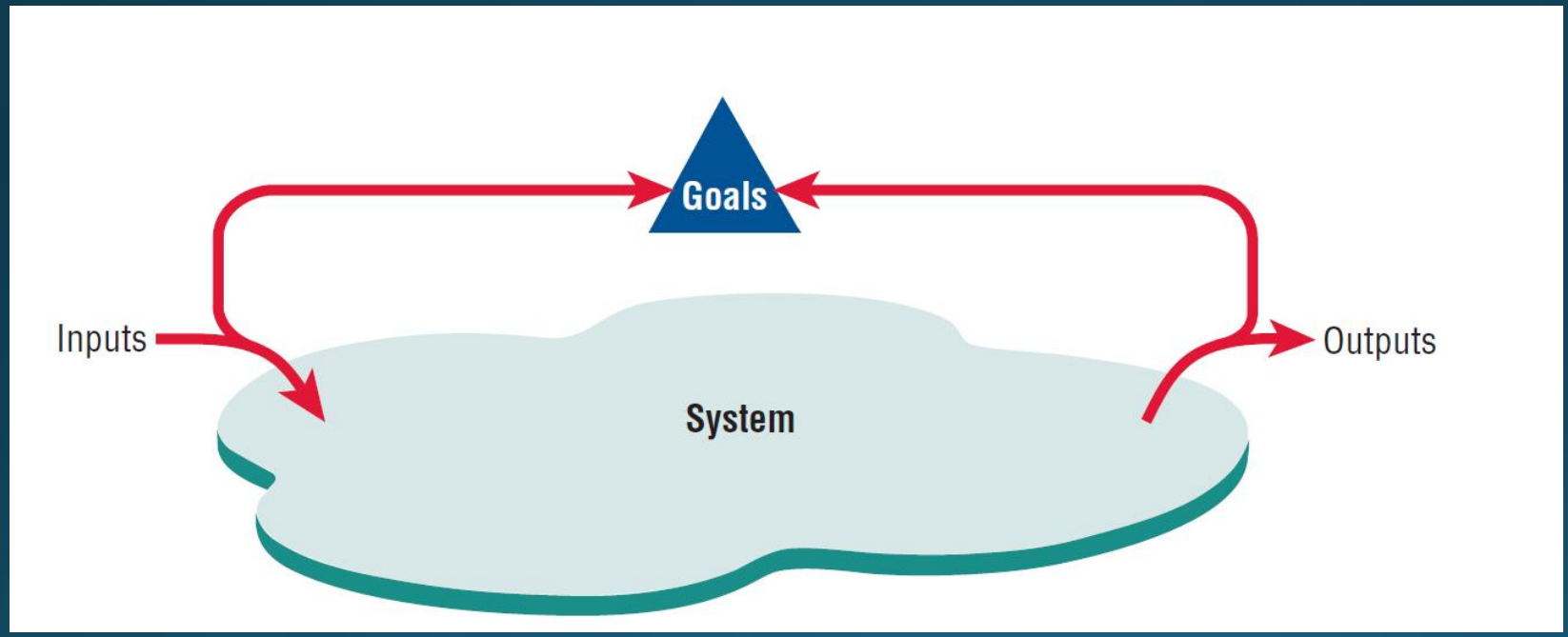
Participate in Open Source Communities

- Rapidity with which new software can be developed and tested
- Faster to have a committed group of experts develop, test, and debug code
- This fosters creativity
- Have many good minds work with innovative applications
- Curiosity about software benefits
- Achieve collective design
  - Incorporate open source software design into:
    - Proprietary products
    - Processes
    - Knowledge
    - IT artifacts

Organizations as Systems

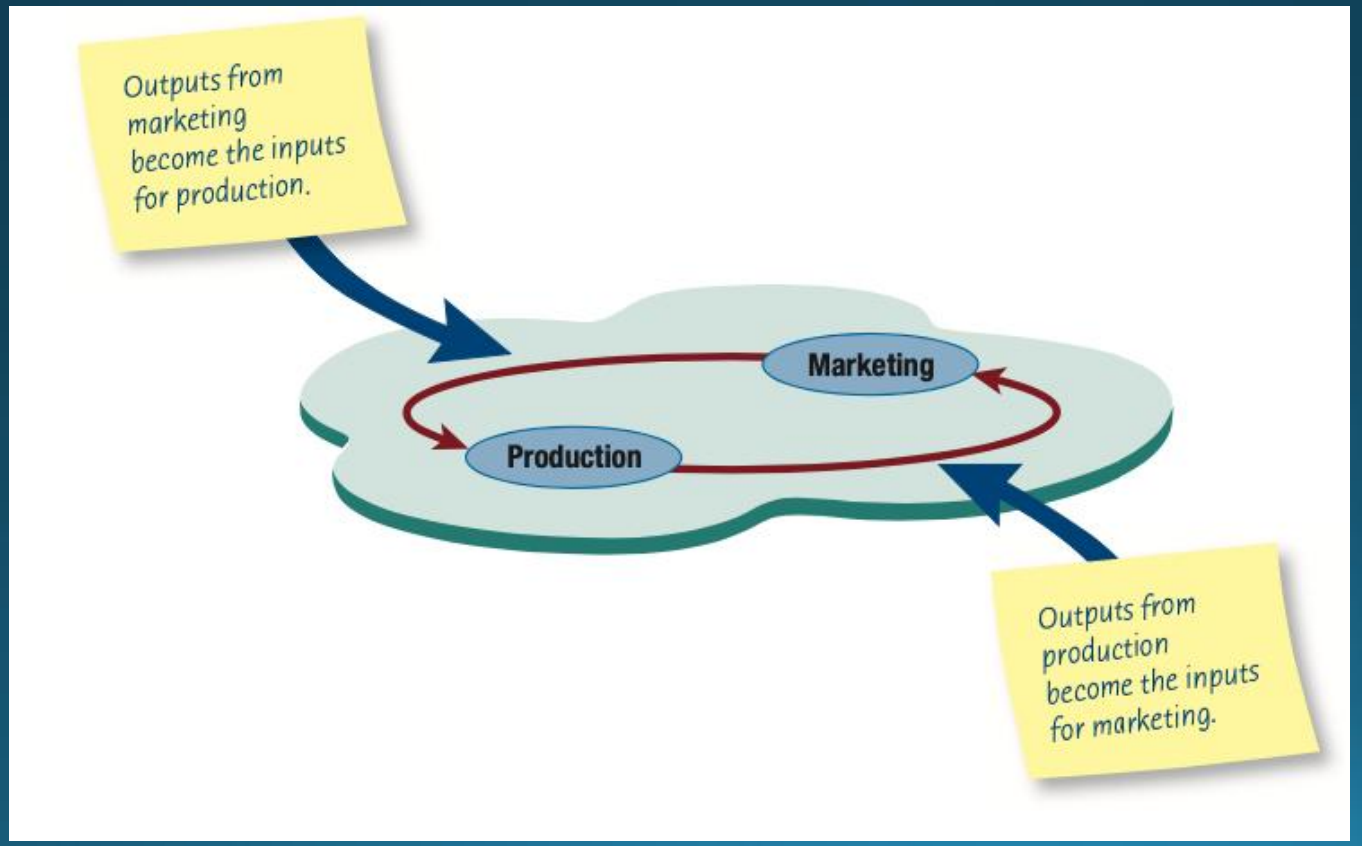
- Conceptualized as systems designed to accomplish predetermined goals and objectives
- Composed of smaller, interrelated systems serving specialized functions
- Specialized functions are reintegrated to form an effective organizational whole
- All systems and subsystems are interrelated and interdependent
- All systems process inputs from their environments
- All systems are contained by boundaries separating them from their environments
- System feedback for planning and control
- An ideal system self-corrects or self-regulates itself.

System Outputs Serve as Feedback that Compares Performance with Goals

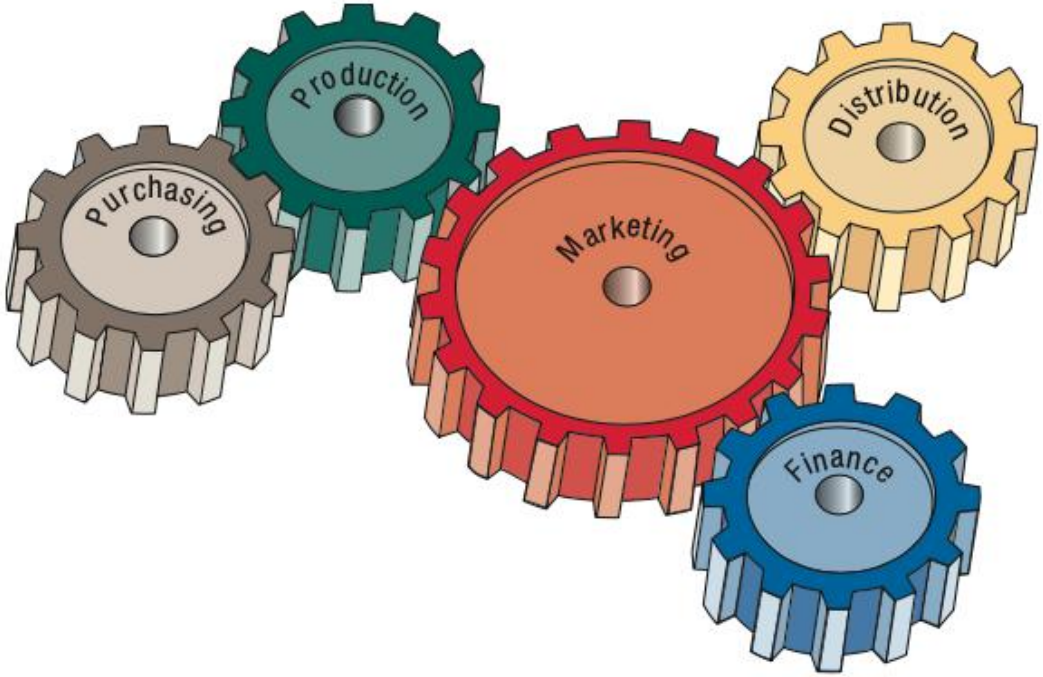


Taking a Systems Perspective

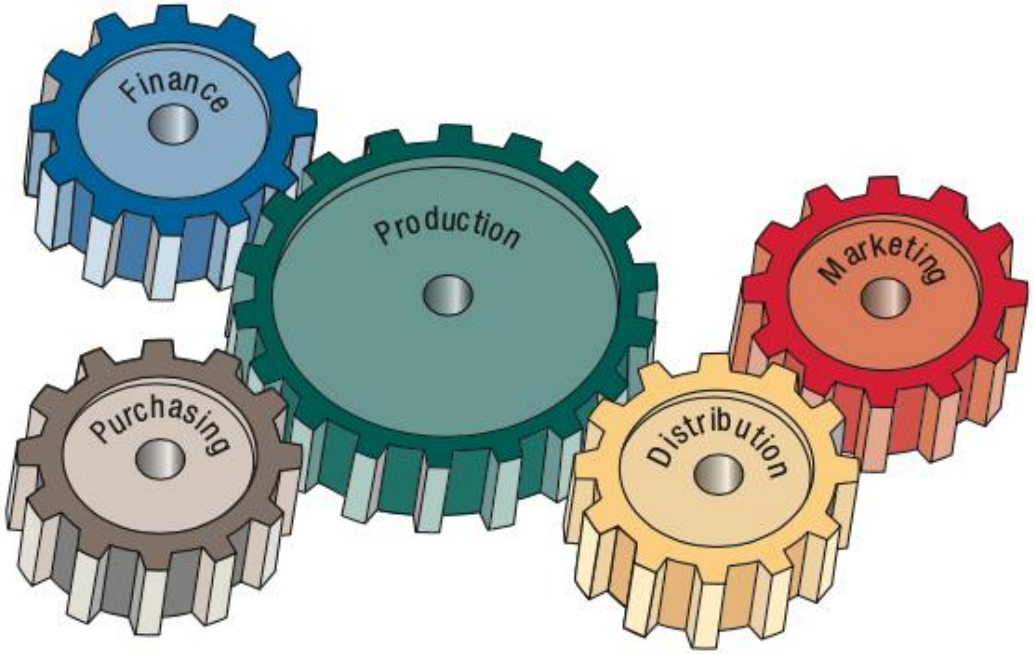
- Allows system analyst to understand businesses before they begin their tasks
- It is important that members of subsystems realize that they are interrelated with other subsystems
- Problems occur when each manager thinks that his/her department is the most important
- Bigger problems may occur when that manager rises through the ranks



Perpective of Functional Managers

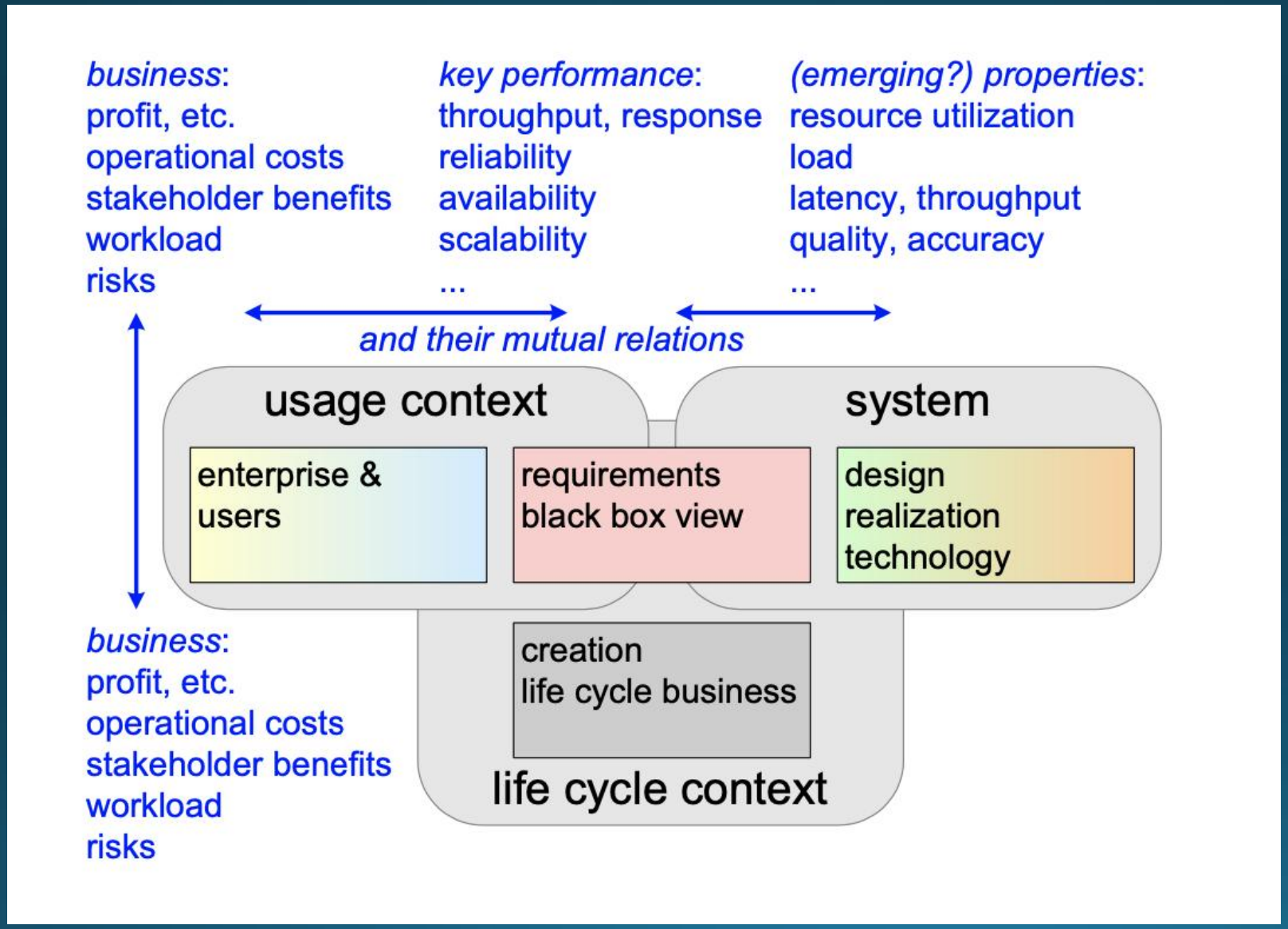


How a Marketing Manager May View the Organization



How a Production Manager May See the Organization

What to Model? A System Vision



References: https://www.gaudisite.nl/SystemModelingAndAnalysisBook.pdf

Visual Modeling: Depicting Systems Graphically

- Context-level data flow diagrams
- Entity-relationship model
- UML
- SysML
  - https://sysml.org/
  - https://github.com/Systems-Modeling/SysML-v2-Release
- ArchiMate
  - https://pubs.opengroup.org/architecture/archimate3-doc/toc.html
  - https://www.archimatetool.com/
- The C4 Model: Conext, Container, Components, and Code
  - https://c4model.com/
- BPMN
  - https://www.bpmn.org/
- Context Map
  - https://contextmapper.org/

- Powerful online diagraming

https://www.lucidchart.com/pages

https://www.processon.com/

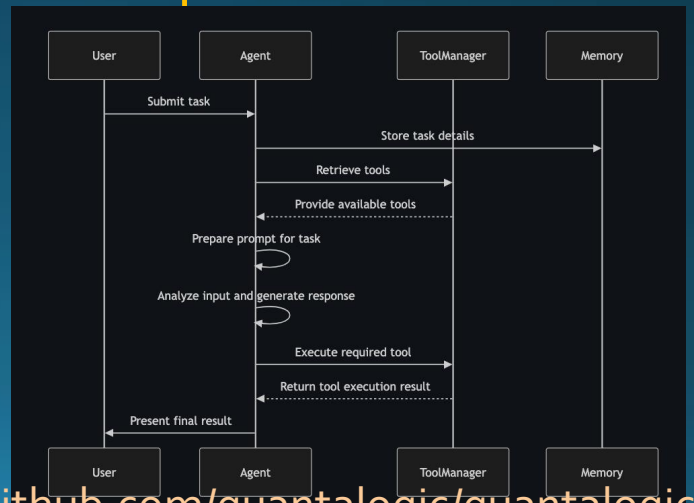
...

- End to end automation tools

https://zapier.com/

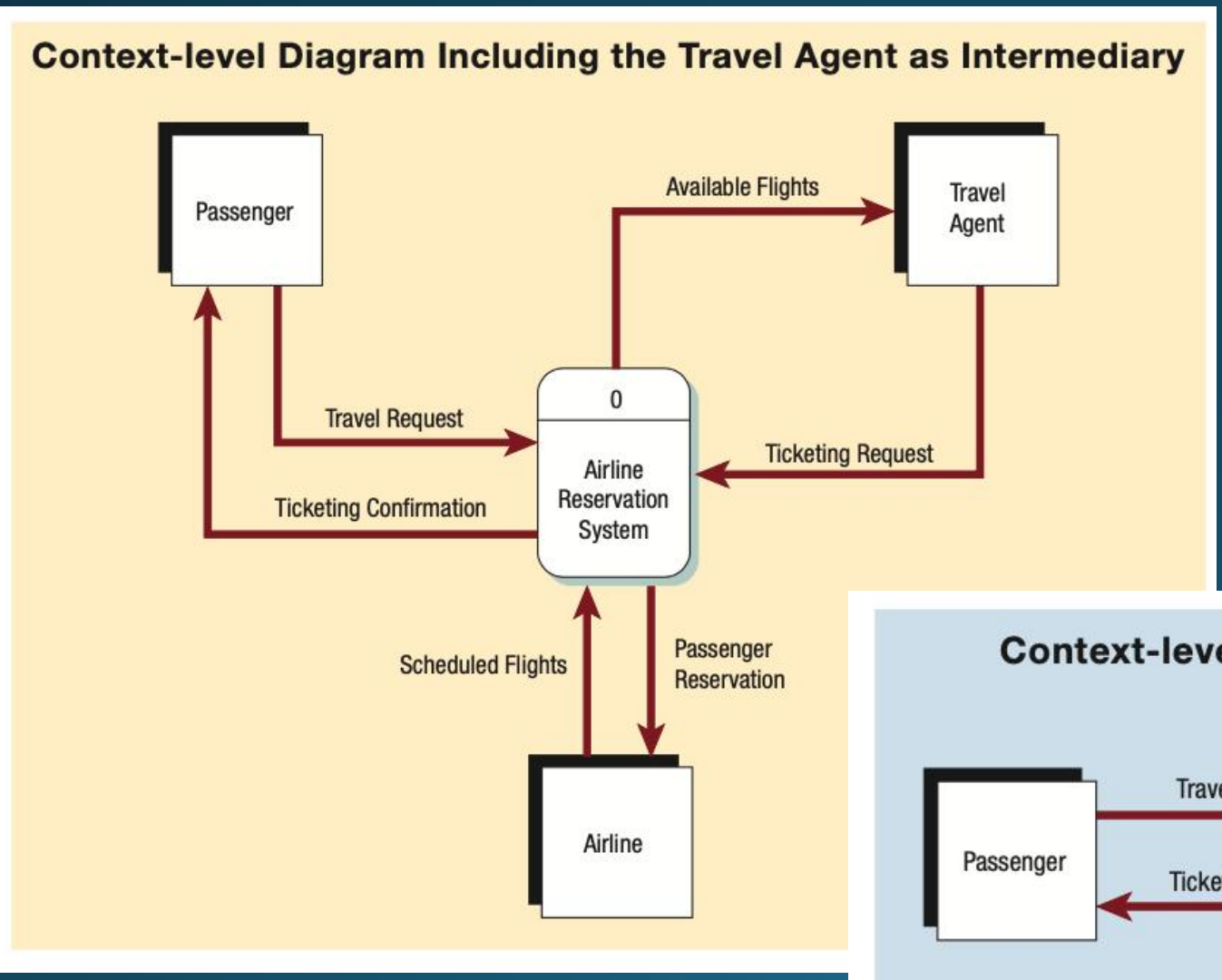
- Modeling is everywhere…

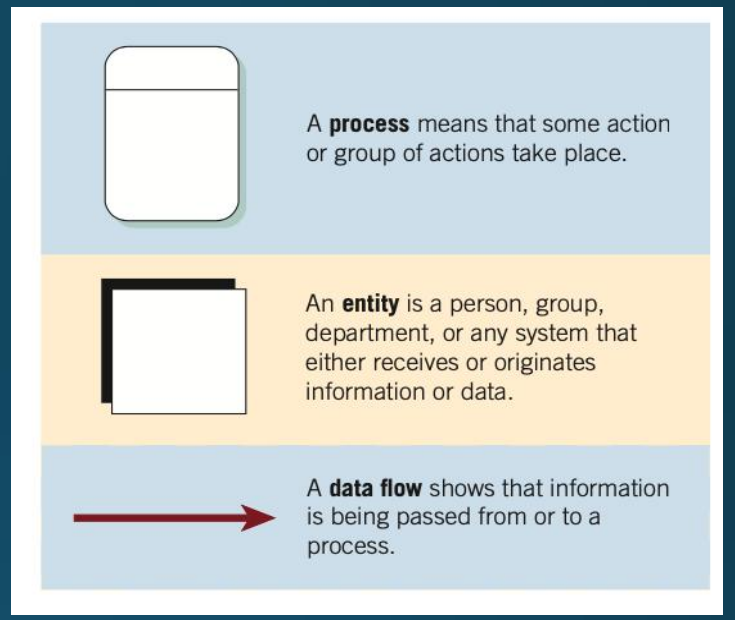
  for example:

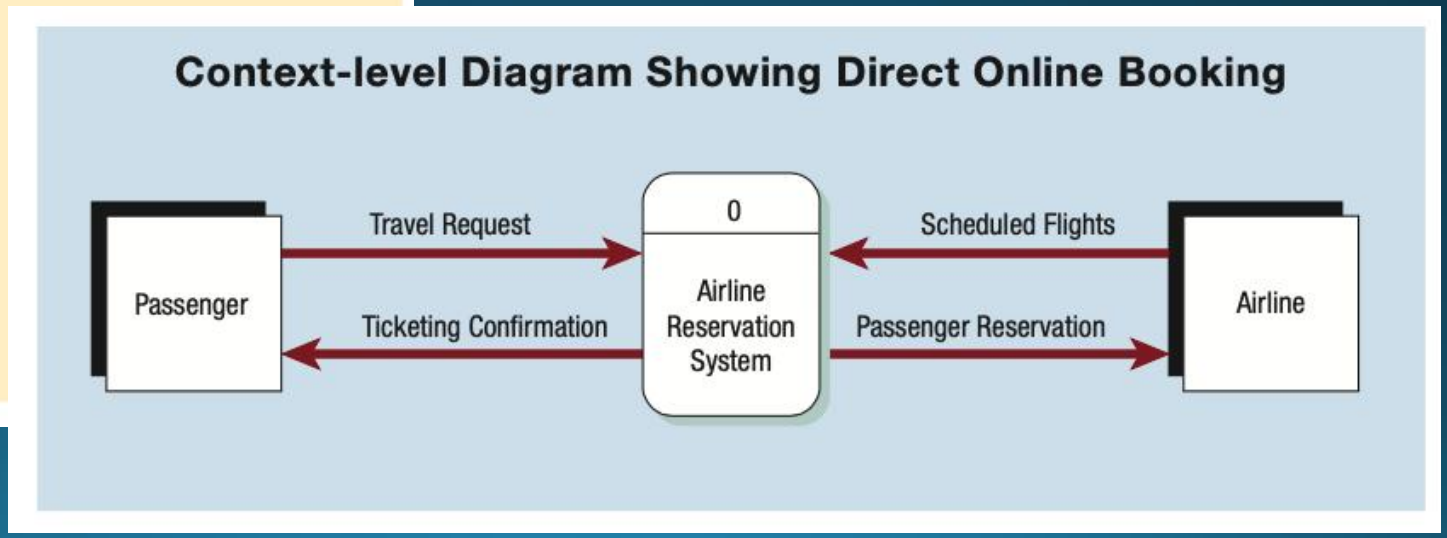


https://github.com/quantalogic/quantalogic

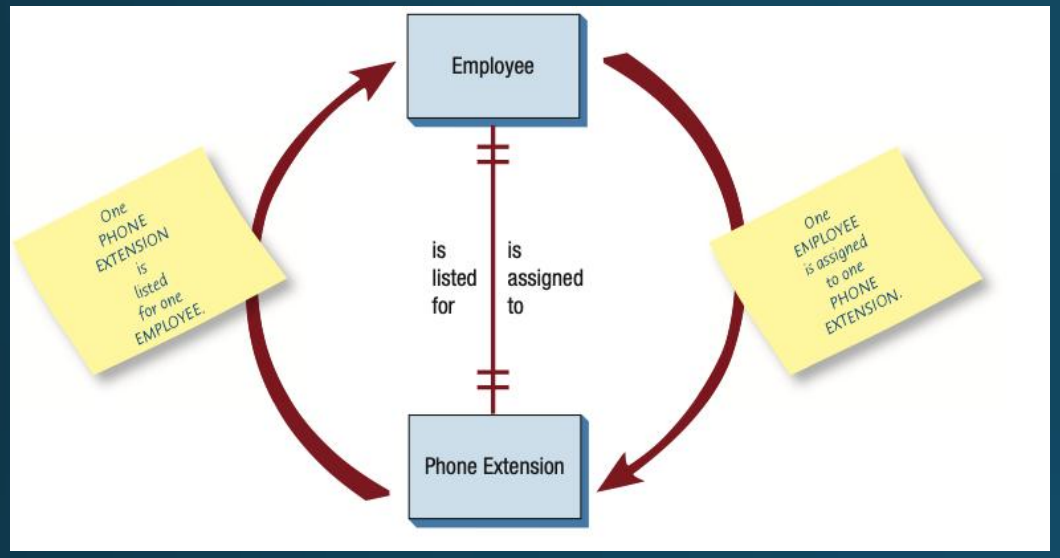
Examples: Context-level data flow diagram

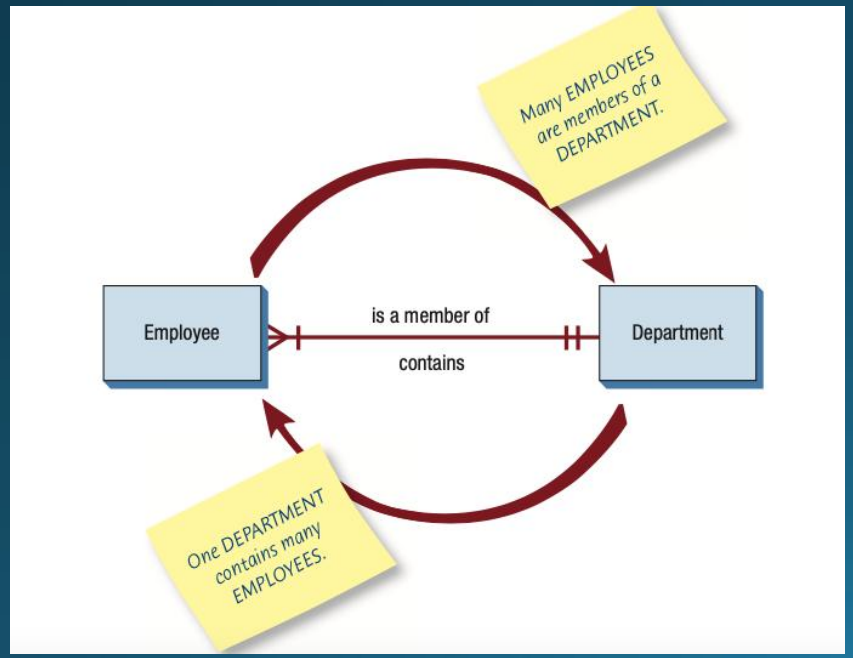


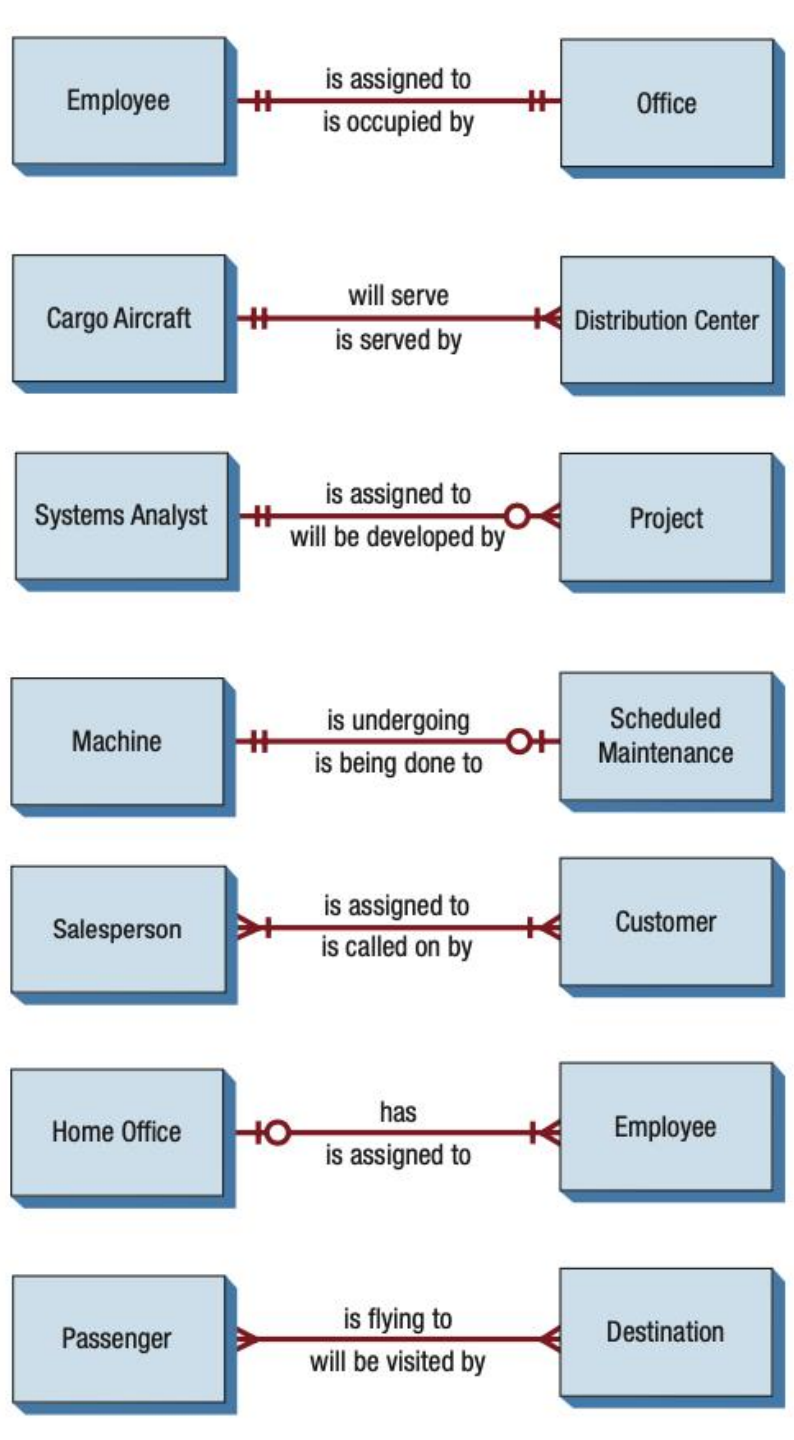




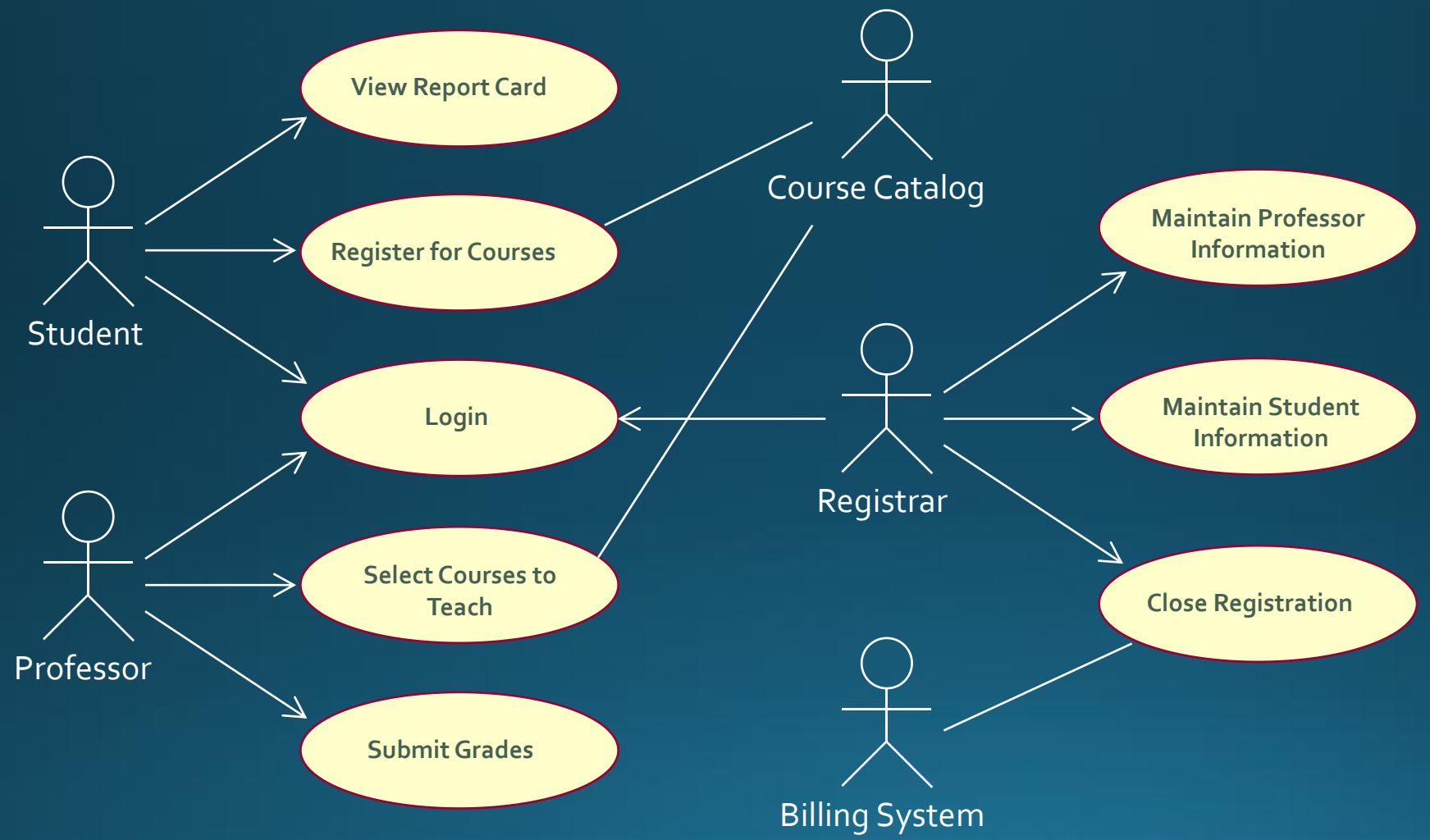
Examples: E-R Diagram





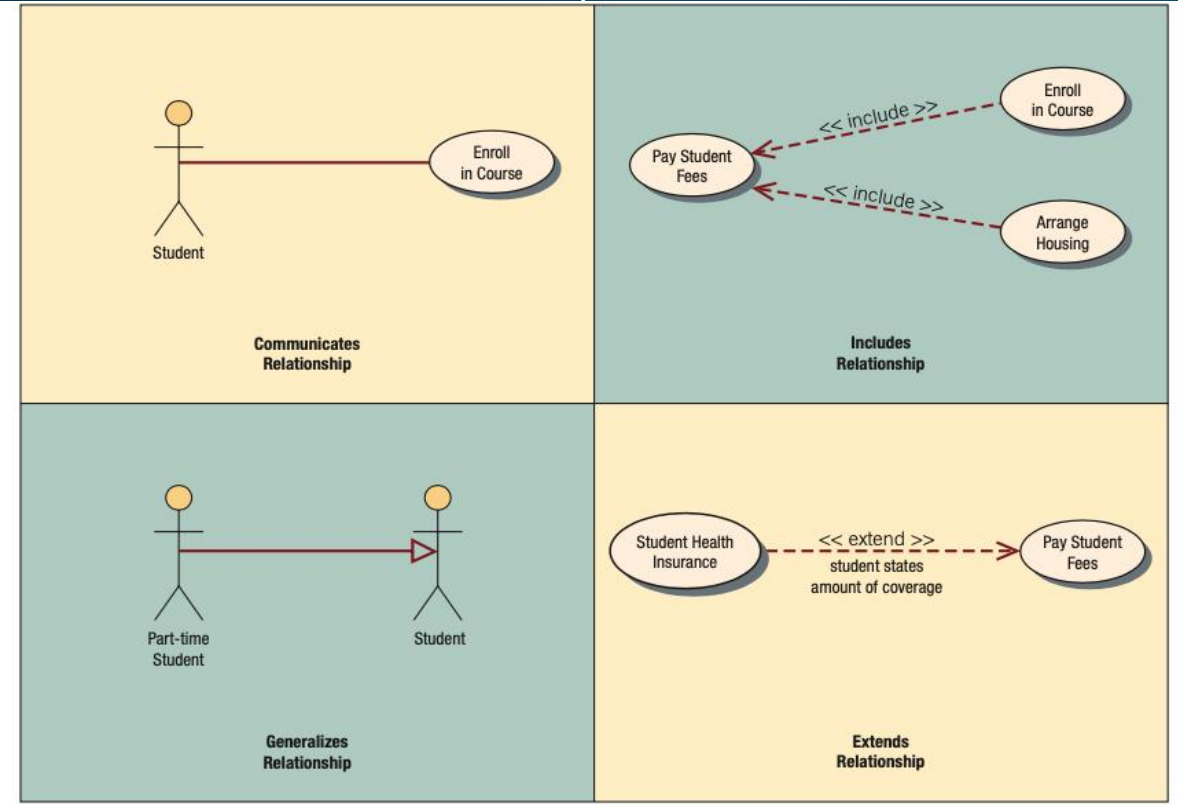


Example: Use Case Diagram



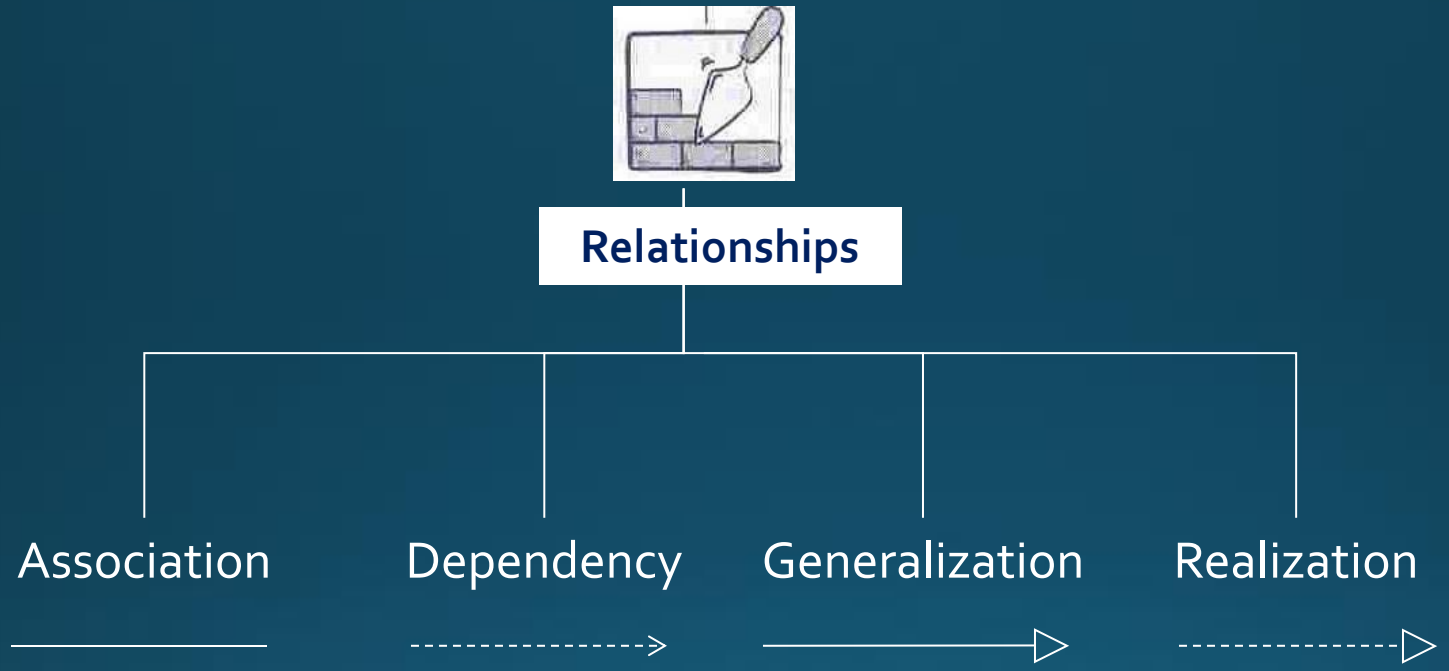
Four Types of Behavioral Relationships in Use Case Diagram

Examples



Relationship	Symbol	Meaning
Communicates	———	An actor is connected to a use case using a line with no arrowheads.
Includes	<<< include >>	A use case contains a behavior that is common to more than one other use case. The arrow points to the common use case.
Extends	<< extend >> --->	A different use case handles exceptions from the basic use case. The arrow points from the extended to the basic use case.
Generalizes	———▷	One UML "thing" is more general than another "thing." The arrow points to the general "thing."

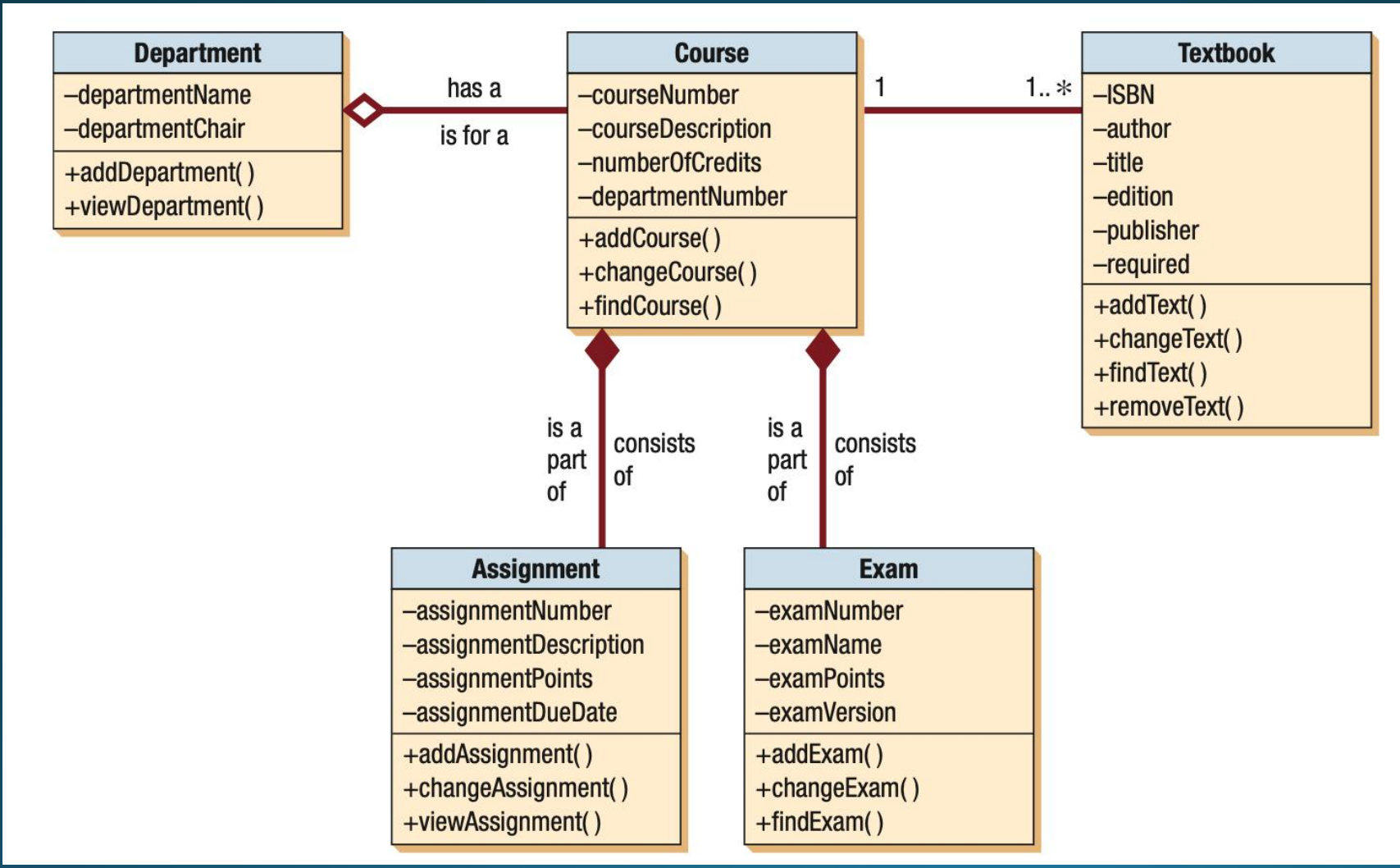
UML Relationships



UML Relationships and their graphical representation

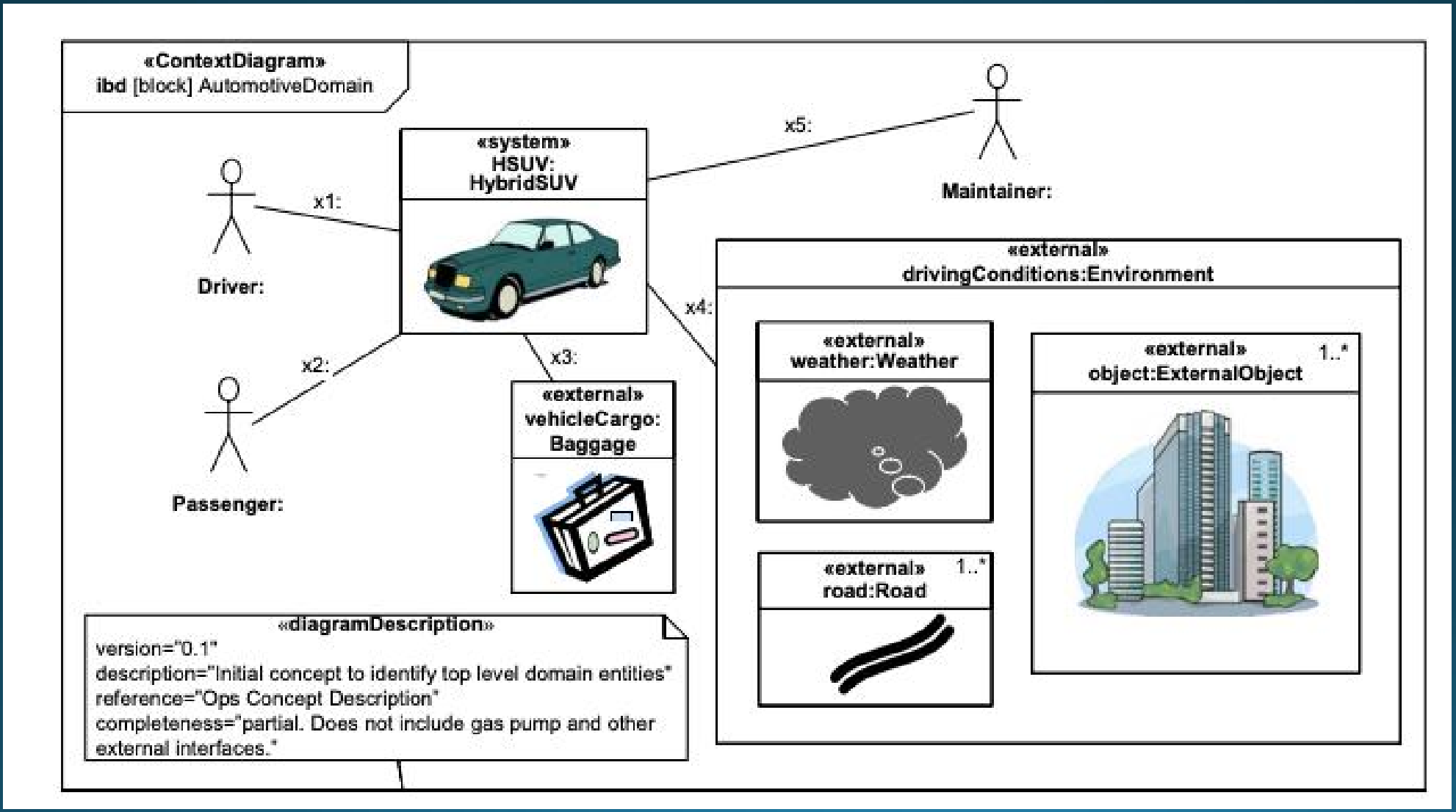
- More examples will be available in later lectures.

Example: UML Class Diagram



Example: SysML

Internal Block Diagram - Setting Context

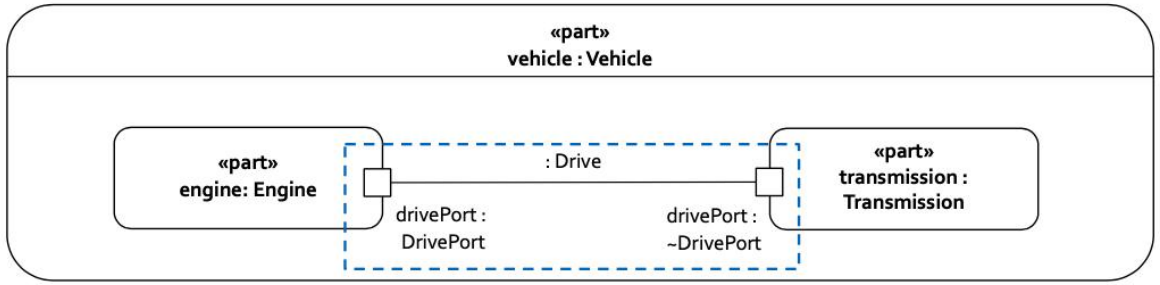


Example: SysML V2

```
package VehicleLogicalConfiguration{
    package PartsTree{
        part vehicleLogical:VehicleLogical{
            part torqueGenerator:TorqueGenerator{
                action generateTorque;
            }
            part electricalGenerator:ElectricalGenerator{
                action generateElectricity;
            }
            part steeringSystem:SteeringSubsystem;
            part brakingSubsystem:BrakingSubsystem;
        }
    }
}
```

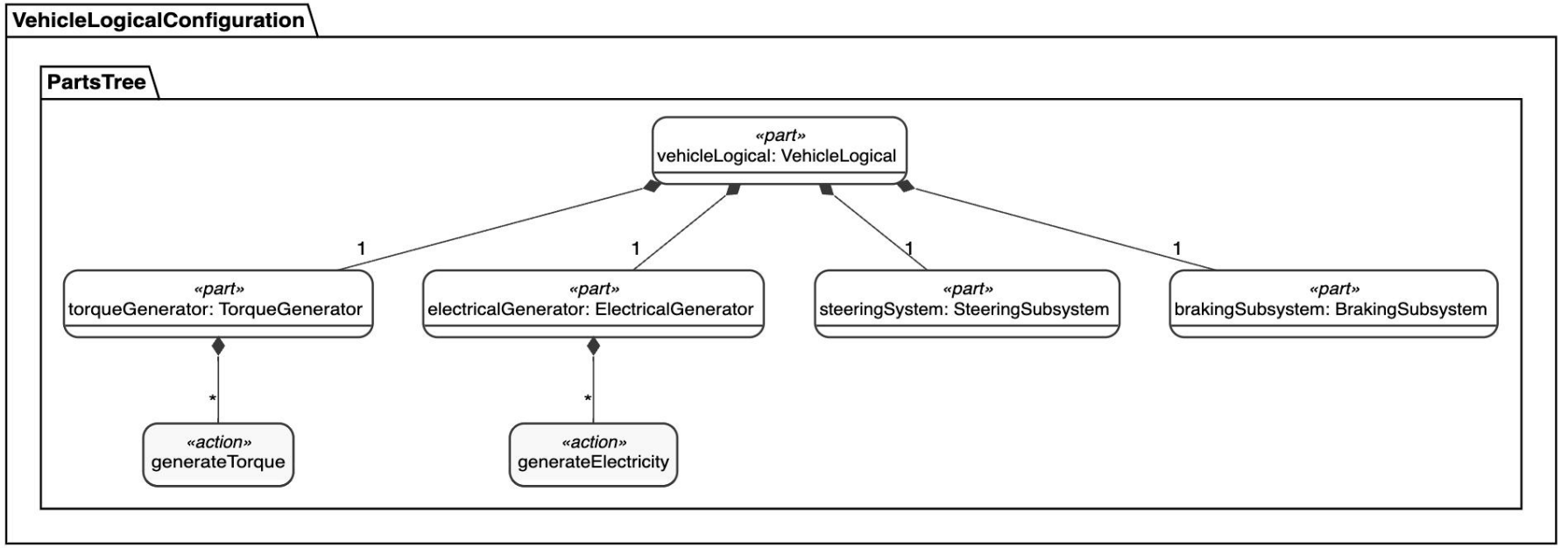
```
interface def Drive {
    end enginePort : DrivePort;
    end transmissionPort : ~DrivePort;
}

part vehicle : Vehicle {
        part engine : Engine { port drivePort : DrivePort; }
        part transmission : Transmission { port drivePort : ~DrivePort; }
        interface : Drive
            connect engine.drivePort to transmission.drivePort;
}
```

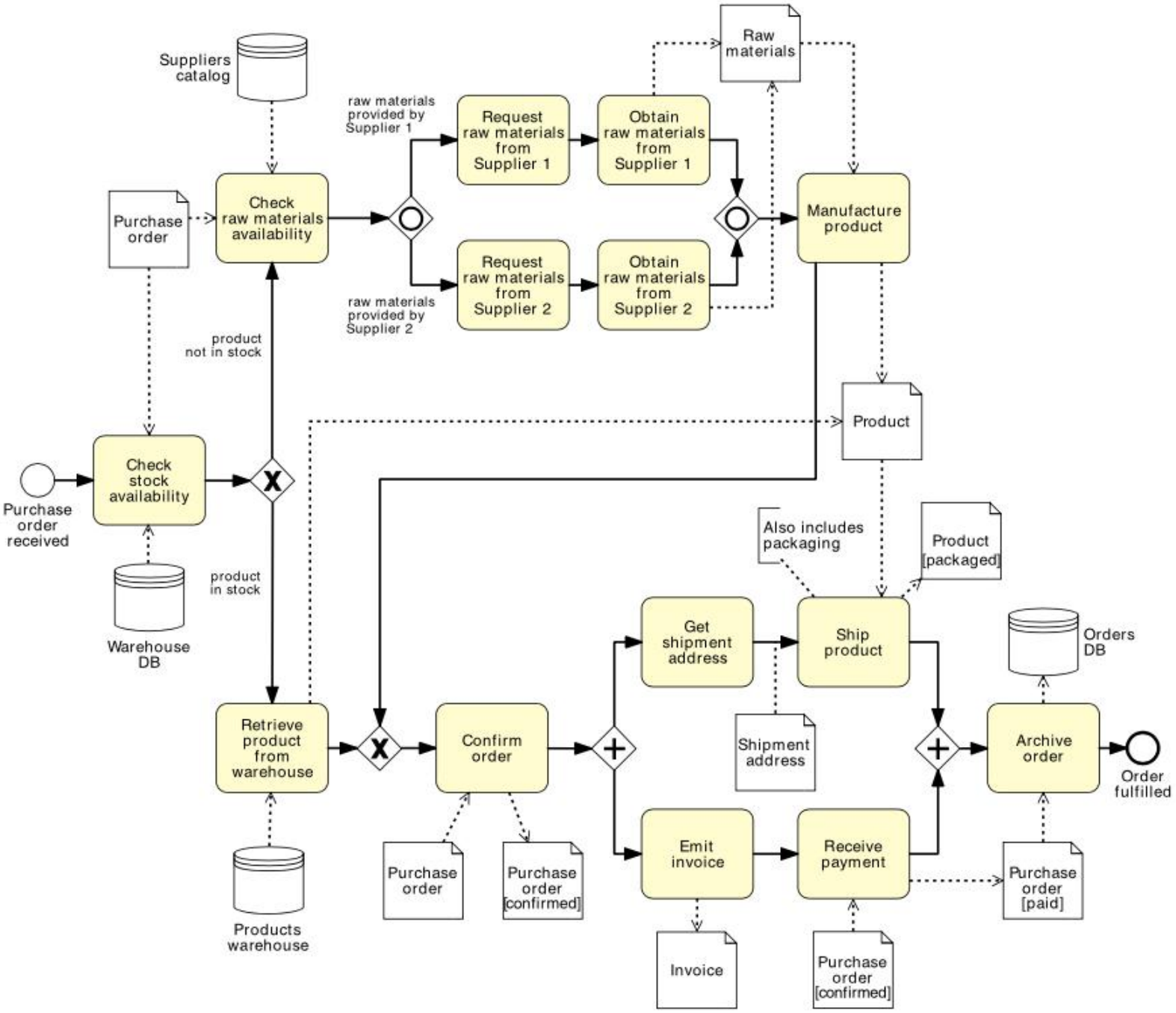


Tom Sawyer Visualization Prototype

```
%viz SimpleVehicleModel::VehicleLogicalConfiguration
```

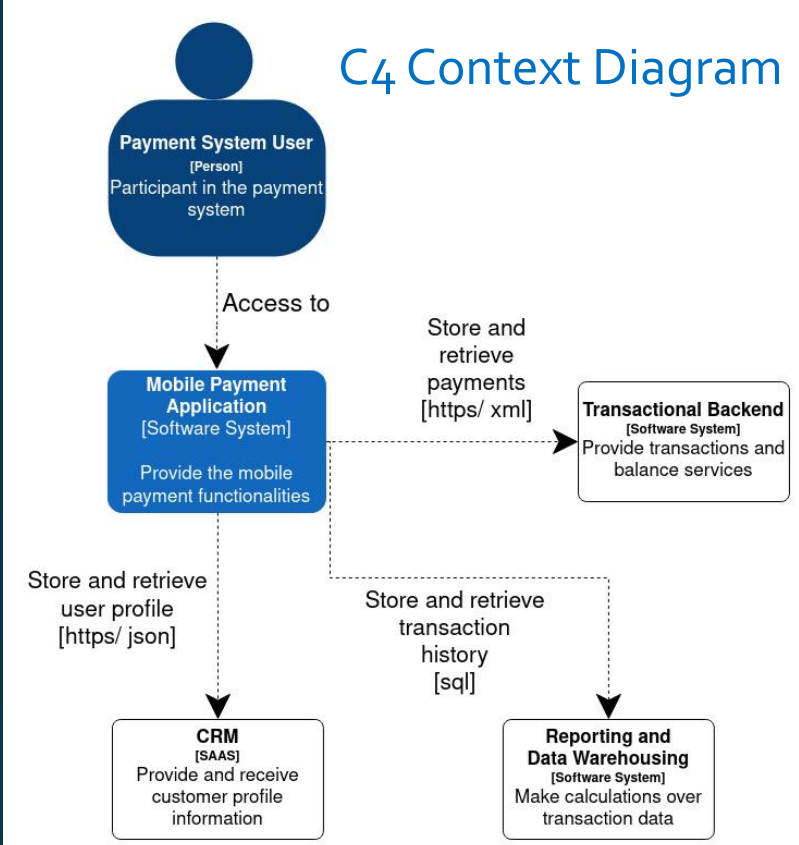


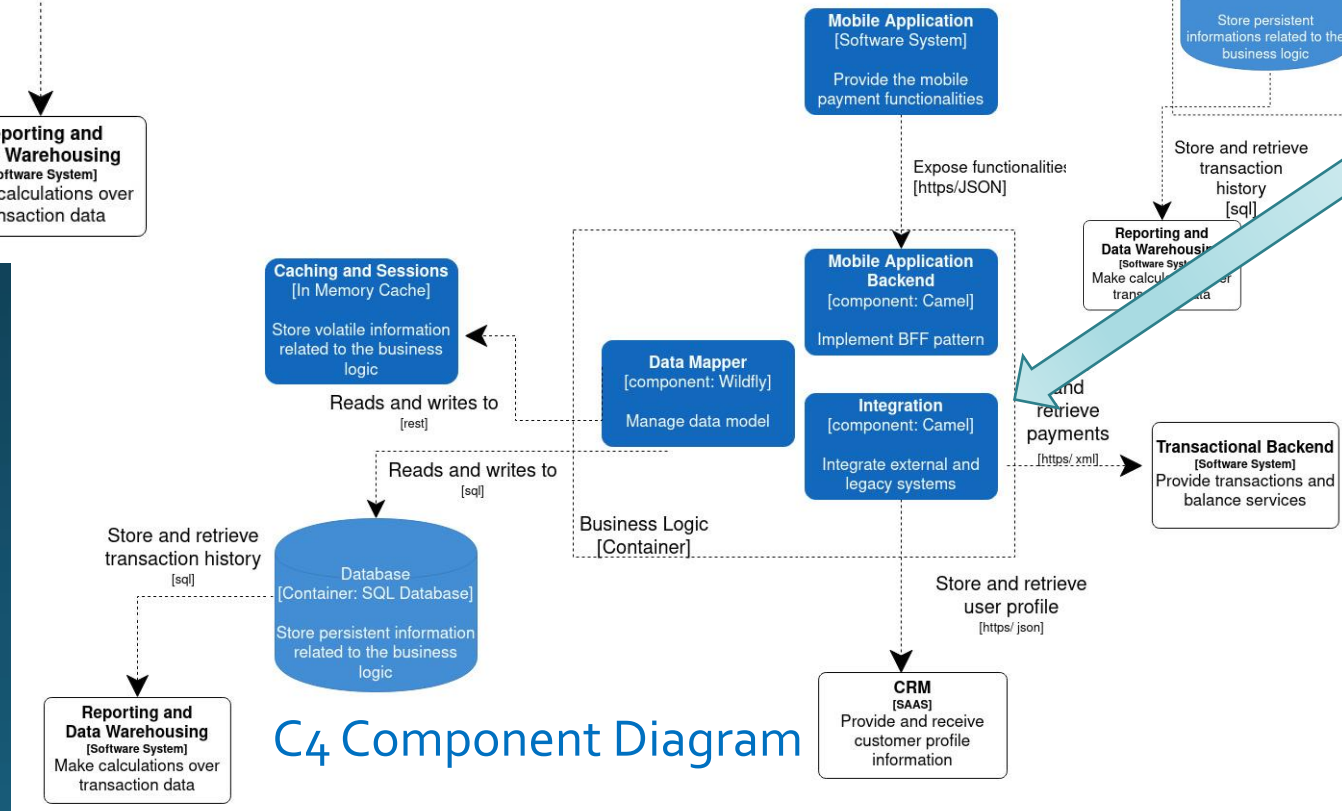
Example: BPMN

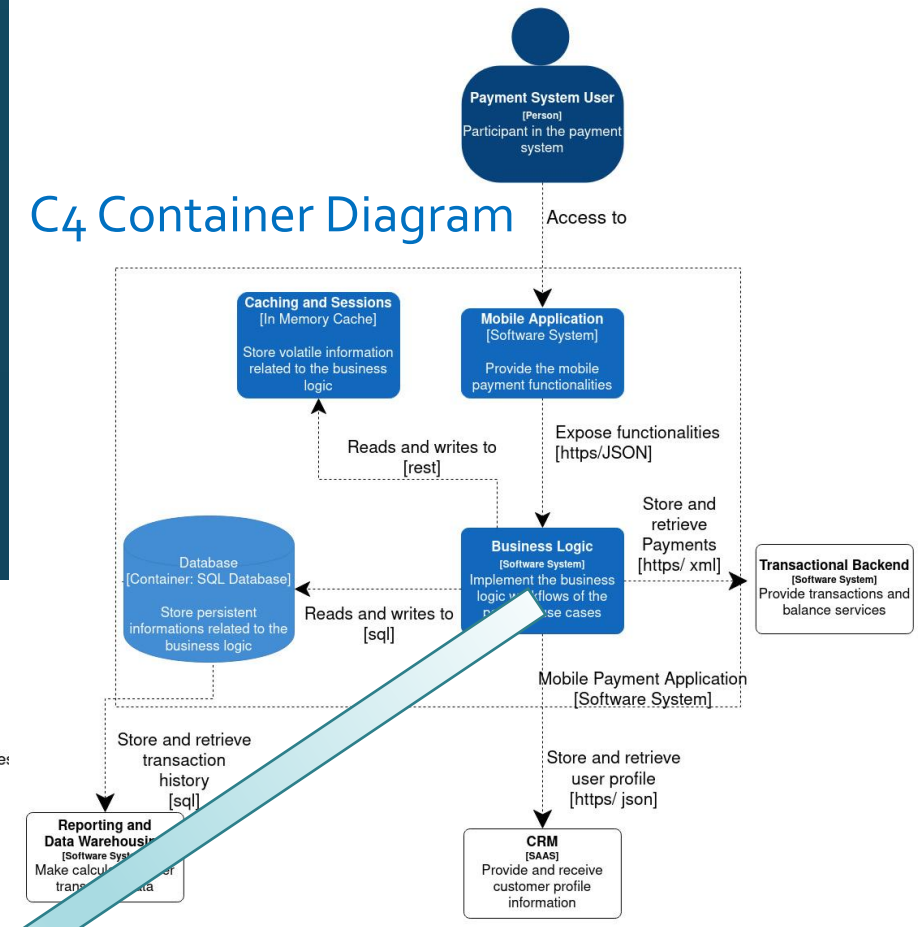


Examples:

C4 diagrams for mobile payment

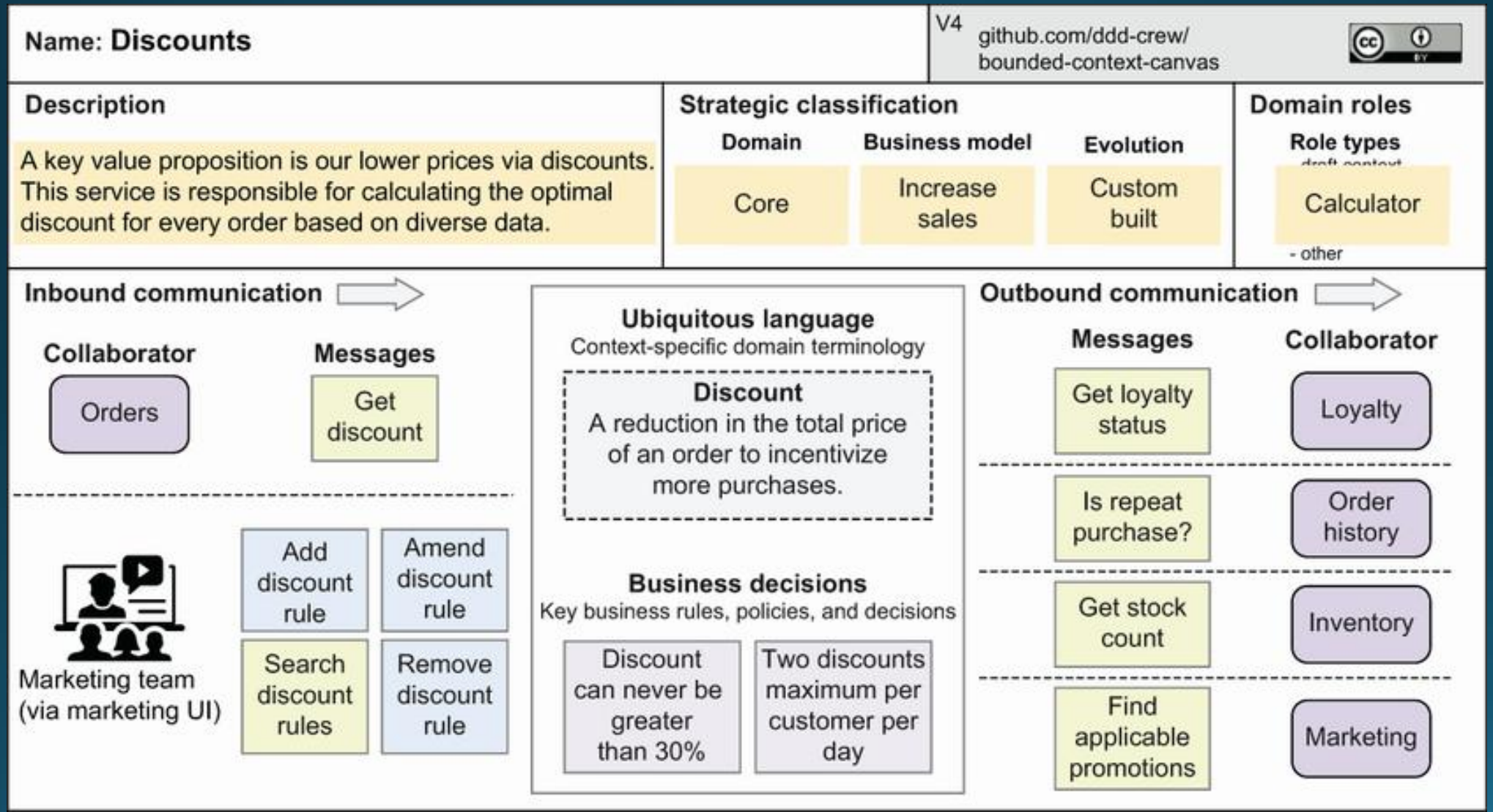






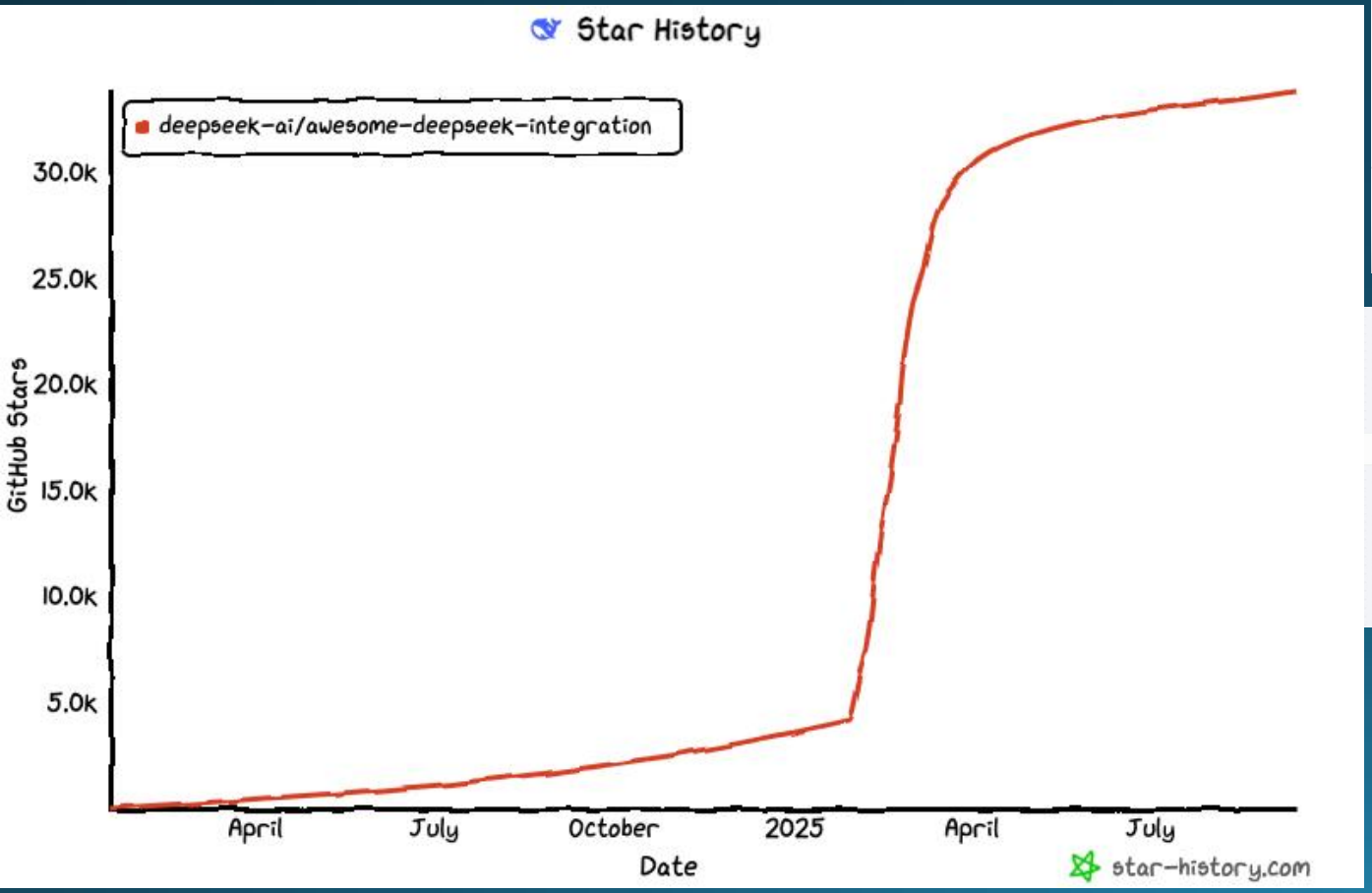
New Ways to Visualize the Requirements and the Design Example: Bounded Context Canvas

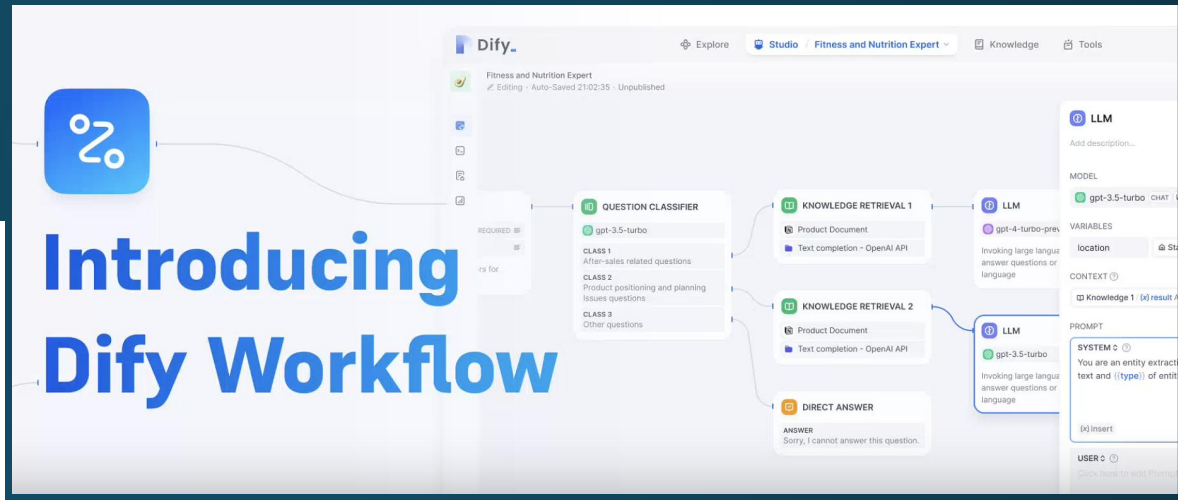
https://github.com/ddd-crew/bounded-context-canvas



Discussion:
The AI+ and Agentic AI Solutions

https://github.com/deepseek-ai/awesome-deepseek-integration

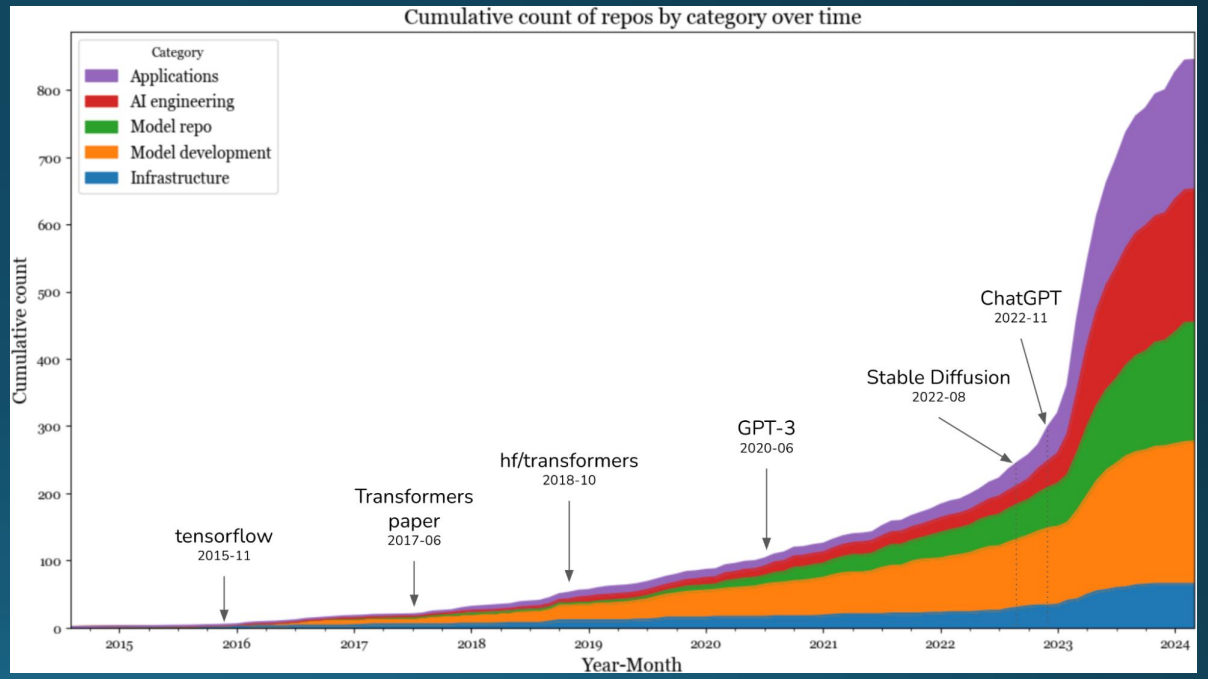




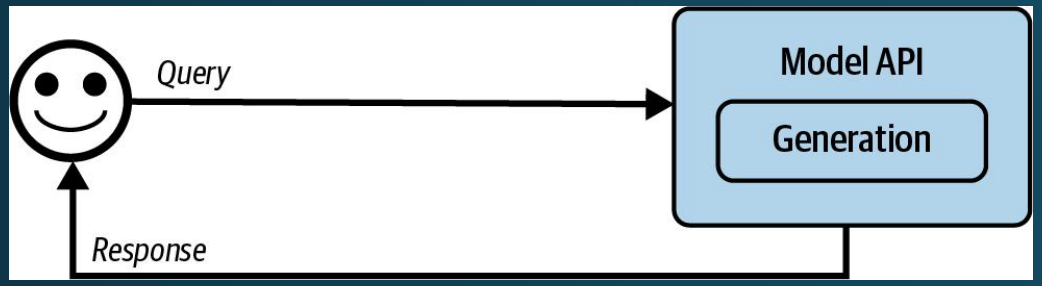
https://dify.ai/
https://github.com/langgenius/dify

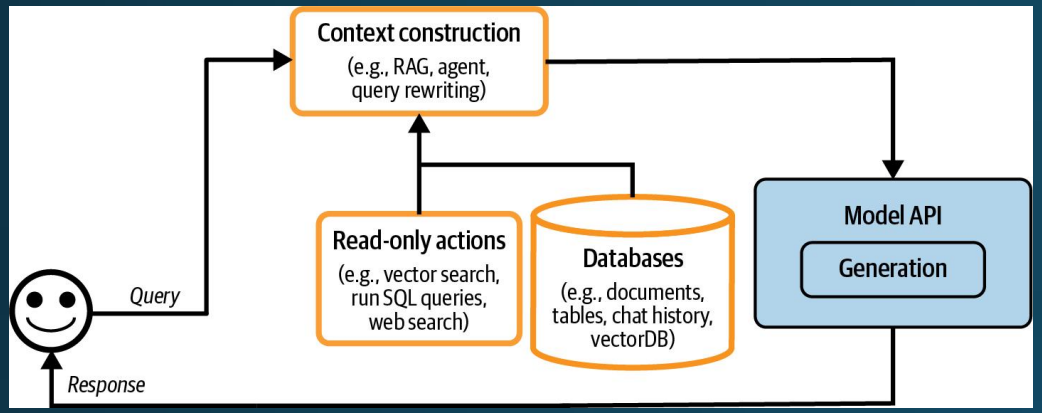
Emerging of AI Engineering

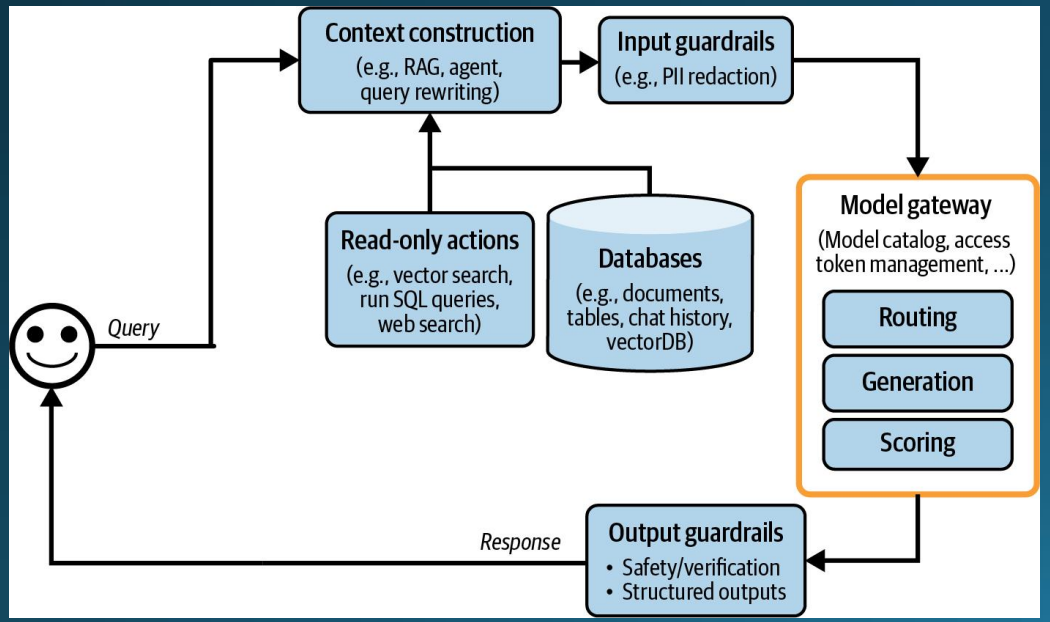
Application development	AI interface Prompt engineering Context construction Evaluation
Model development	Inference optimization Dataset engineering Modeling & training Evaluation
Infrastructure	Compute management Data management Serving Monitoring

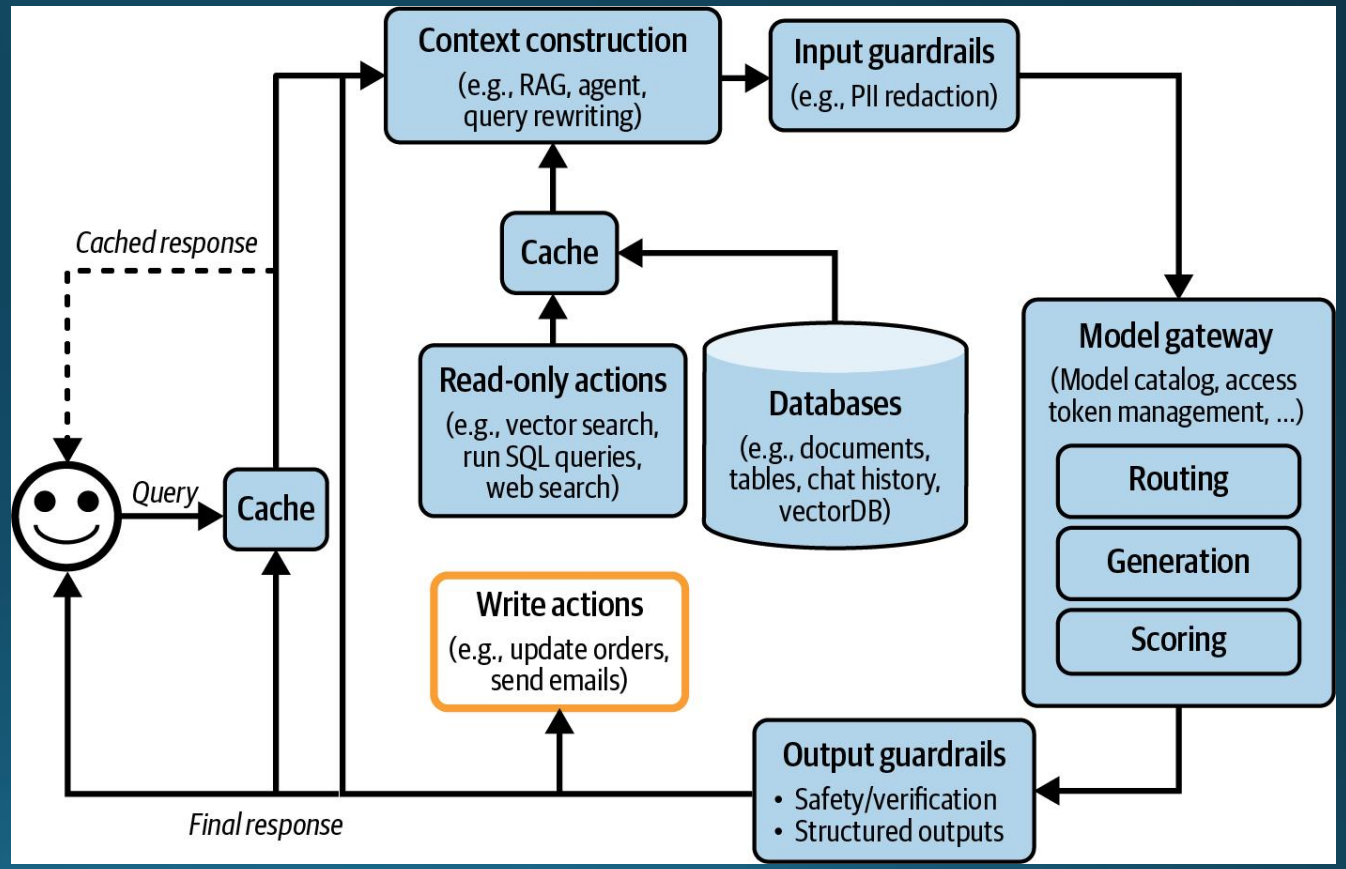


Sample:
AI Engineering Architecture









Discussion: The Future?

Table 1: Comparison between developer working routine in 2024 and 2030.

AI Assistance	AI Limitations in 2024	AI Solutions in 2030
Mental Health	AI is not able to improve developers' mental health	AI can recommend the right moment for breaks and personalized activities to maintain their well-being
Fault Detection	AI is very limited on automatically finding bugs and providing bug fixes for complex software systems	AI can automatically detect bugs and vulnerability, even recommending how to handle them
Code Optimization	AI can recommend only simple code suggestions	AI can optimize the code, monitoring the code written by the developer and suggesting alternatives
Smart Team Interactions	AI is not able to handle or suggest interactions between colleagues or teams in the same company	AI is able to support developers, arranging useful meetings and favoring developers' interactions
Learning New Skills	AI can not suggest relevant resources or recommend novel programming languages features or APIs	AI can find alternatives involving new features and proposes tailored learning path for developers

Ciniselli, M., Puccinelli, N., Qiu, K., & Di Grazia, L. (2024). From Today's Code to Tomorrow's Symphony: The AI Transformation of Developer's Routine by 2030. *ArXiv*. /abs/2405.12731