Carnegie Mellon University
Master of
Software Engineering

17 635 Software Architectures

Introduction

Swarnalatha Ashok
Associate Teaching Professor

Objectives

- To understand the motivation for software architecture
- To understand the definition of software architecture
- To understand common architectural activities

Topics

- Motivation for developing a Software Architecture
- What is Software Architecture
- What are common architectural activities

Topics

- Motivation for developing a Software Architecture
- What is Software Architecture
- What are common architectural activities

Software Product

A Car Hailing application

An application to get you from one place to the other on road

- A commuter should be able to
  - Hire different types of vehicles such as a small car, a medium sized car, a large car, a van, bus or a bike
  - Specify the pickup and destination of the ride
  - Track the current ride on the application
  - Pay for the ride using different payment modes such as cash, credit card, debit card, google pay, apple pay etc.
  - . . .

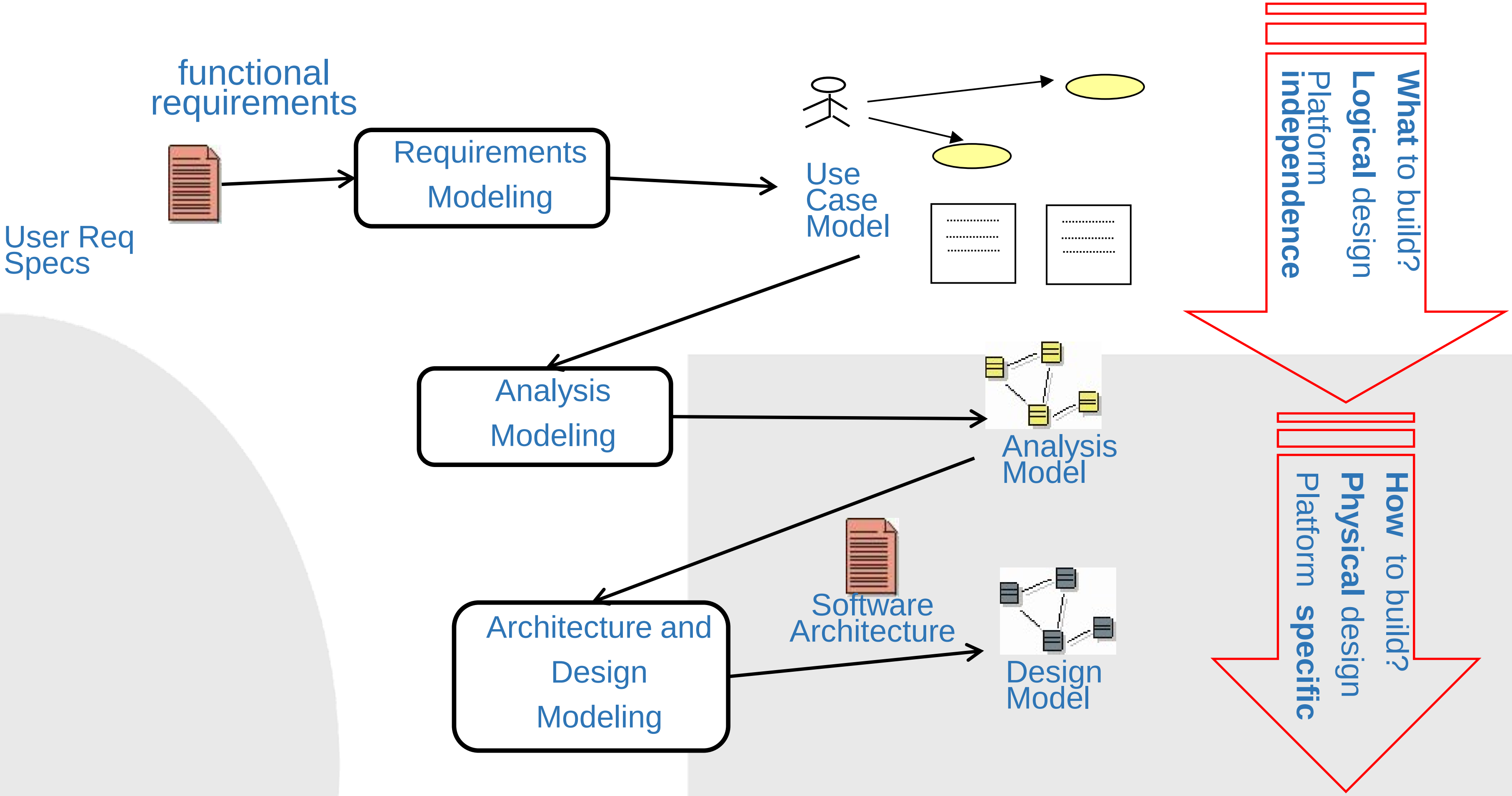
Software Product

A Car Hailing application

An application to get you from one place to the other on road

- A driver should be able to
  - Register his vehicle and himself as an authorized driver
  - Hire a vehicle from Ride’O and register himself as an authorized driver
  - Turn on and off a ride
  - Accept a ride and Cancel a ride
  - . . .

Software Development Process - activities



functional requirements

User Req Specs

Requirements Modeling

Use Case Model

Analysis Modeling

Analysis Model

Architecture and Design Modeling

Software Architecture

Design Model

What to build?
Logical design
Platform
independence

How to build?
Physical design
Platform **specific**

Object Oriented techniques

- Use case modeling (requirements model)
- Domain objects diagram (analysis model)
- Class diagram (design model)
- Sequence diagram (design model)
- State transition diagram (design model)
- Etc.

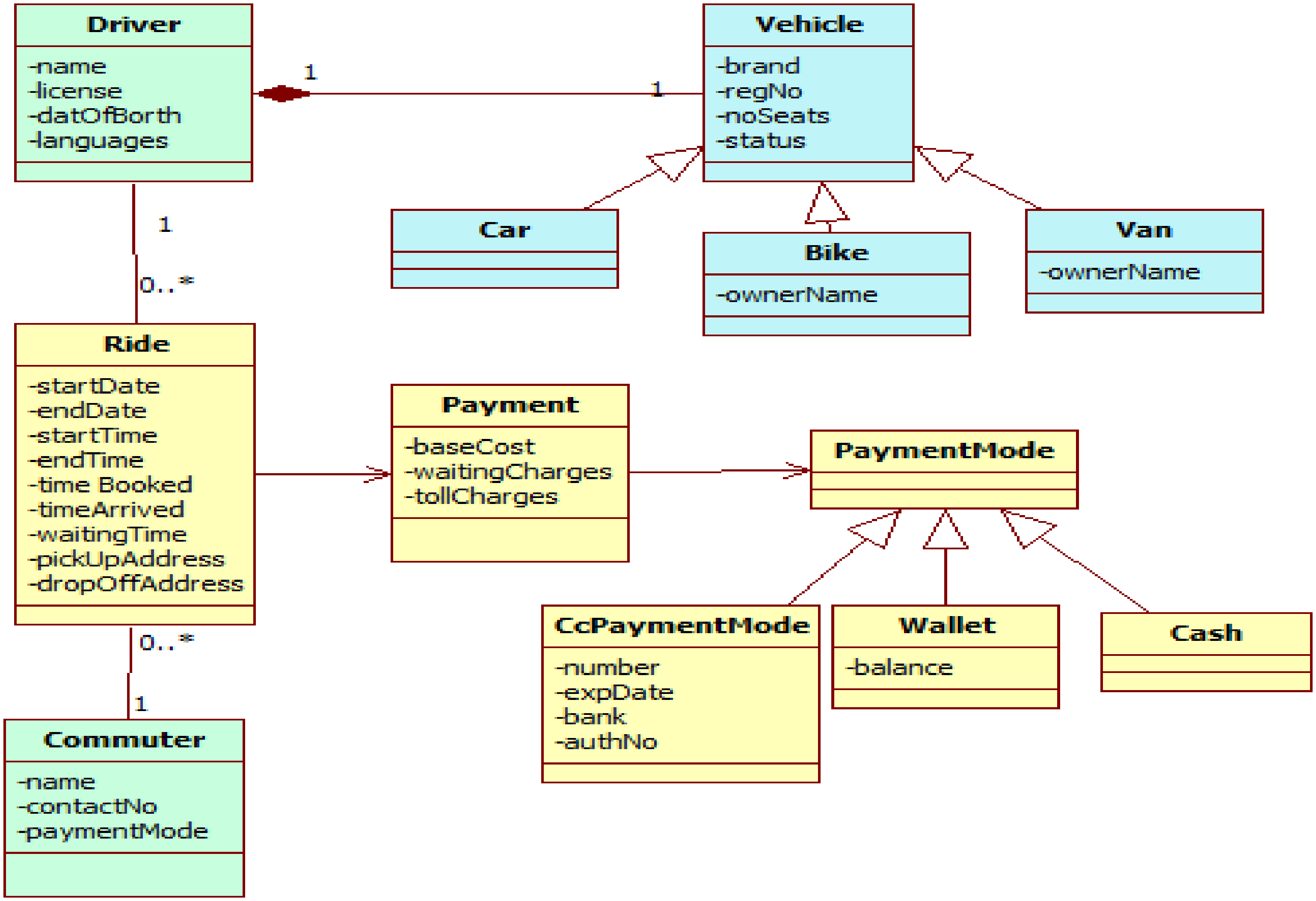
Use Cases

- A use case describes the **interaction** between an **actor** and the **system**
- An example for hailing a car:
  - **System** displays the screen for entering hailing information
  - **Commuter** provides the ‘from’ and ‘to’ locations
  - **System** shows the options for the vehicles and cost
  - **Commuter** selects the vehicle
  - **System** searches for available vehicles and displays the estimated time of arrival
  - …

Use Cases → Domain Model

- Domain model captures the **most essential classes of the domain** that are **relevant** to the application.
- A standard technique is to look at nouns in the requirements/use cases
- Nouns are candidates for domain objects e.g.:
  - Commuter, Driver, Car
  - Route etc.

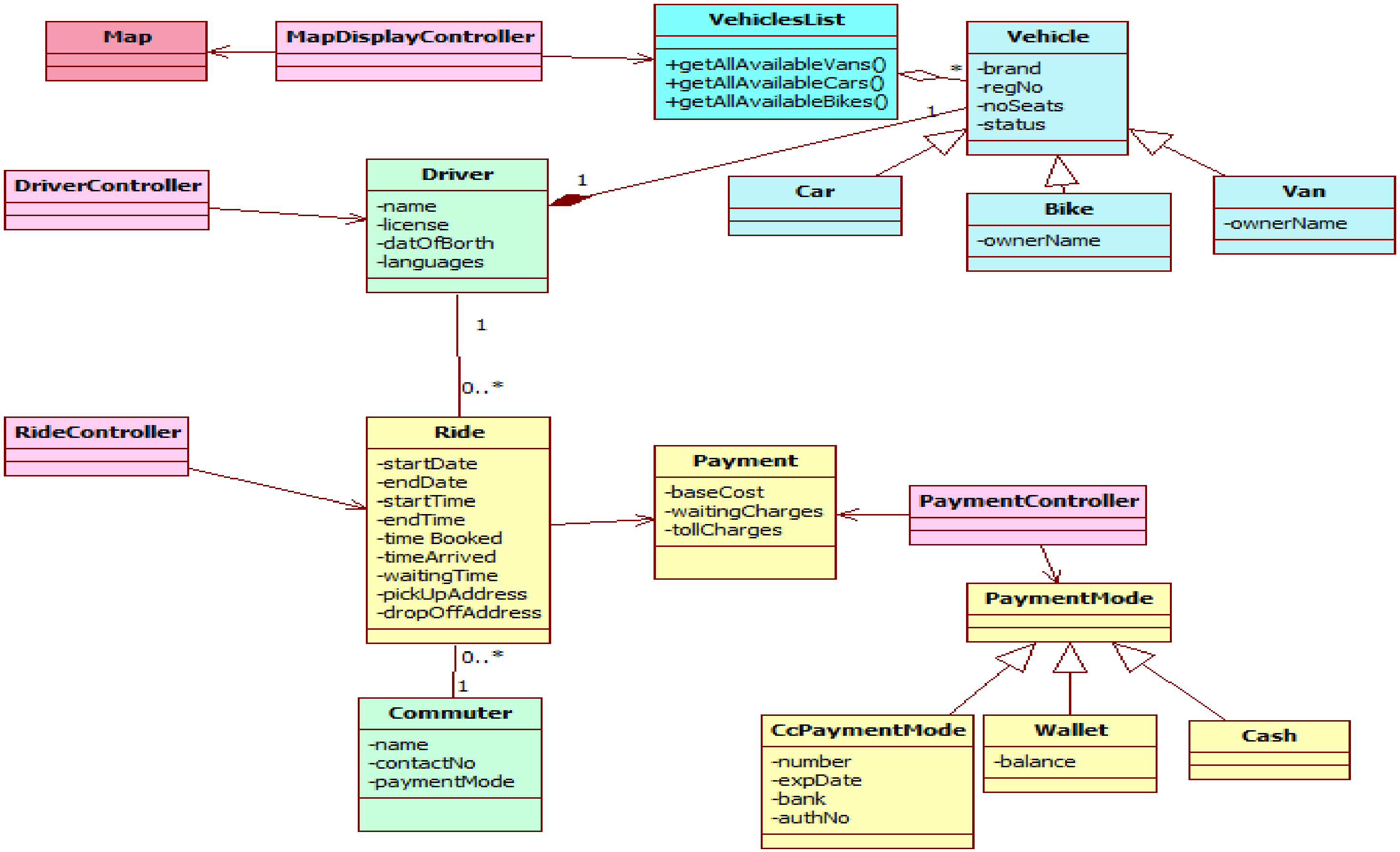
Use Cases → Domain Model



Analysis Model: Class diagram

- Analysis model uses the domain model
- Purpose is **to understand the use cases**
- Focus is on the **problem space** and not the solution space (implementation details)
- Expressed in terms of class diagrams and collaboration diagrams (or sequence diagram)

Analysis Model: class diagram



Map

MapDisplayController

VehiclesList
+getAllAvailableVans()
+getAllAvailableCars()
+getAllAvailableBikes()

Vehicle
-brand
-regNo
-noSeats
-status

Car

Bike
-ownerName

Van
-ownerName

DriverController

Driver
-name
-license
-datOfBorth
-languages

RideController

Ride
-startDate
-endDate
-startTime
-endTime
-time Booked
-timeArrived
-waitingTime
-pickUpAddress
-dropOffAddress

Payment
-baseCost
-waitingCharges
-tollCharges

PaymentController

PaymentMode

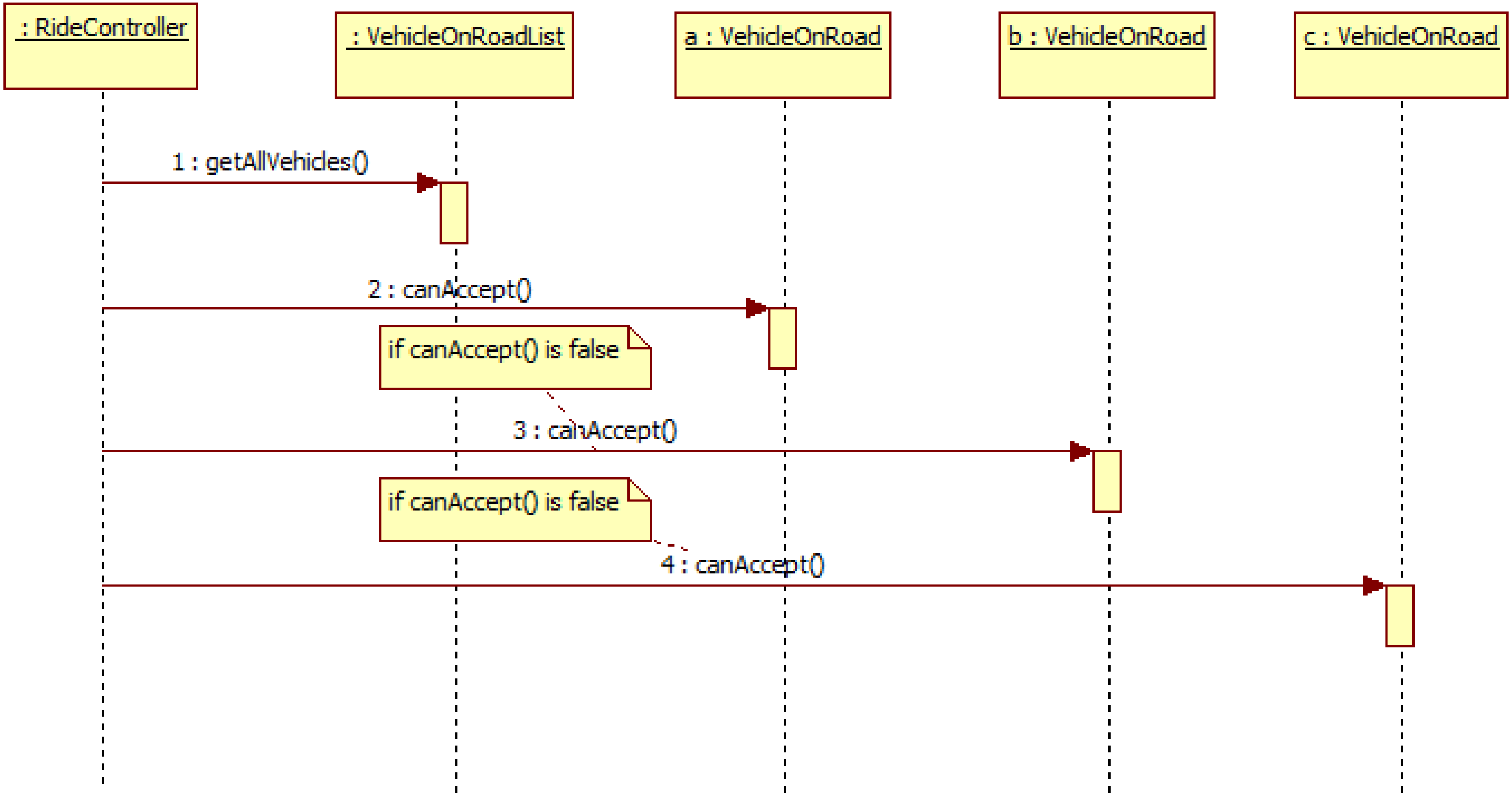
CcPaymentMode
-number
-expDate
-bank
-authNo

Wallet
-balance

Cash

Commuter
-name
-contactNo
-paymentMode

Analysis Model: Sequence diagram
Driver accepts a ride use case



Design Model

- Design model **refines** the analysis model with details of **specific implementation details**
- Module structures, algorithms
- Choice of frameworks, storage technology etc.

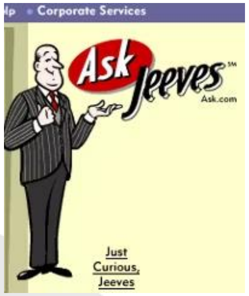
Alignment w/ Business Needs?

- We can now verify that the functionality can been implemented
- We also understand what part of the system is responsible for what part of the capability
- Is this sufficient?

Is it enough to ensure that the functional requirements are implementable?

- Have the business needs been achieved?
- Let’s look at an example or two …

Search Engine example



others

What made Google the leading Search Engine?

Search Engine example

What made Google the leading Search Engine?

- Performance - how quickly it returns the results
- Accuracy - page ranking and relevance
- User experience
- Data-driven decision-making
- Google's commitment to innovation – did AI make a difference?

Google depends on Ad revenue for their income

- As Google’s **loading time decreasing** from 0.9 seconds to 0.4 seconds ad **revenue increases** by 20%
- Search results in a **particular shade of blue** yields a 16 times **greater click through rate** than the next best color at Google

eCommerce example

What makes Amazon the leading eCommerce platform?

- Convenience
  - Amazon makes it easy to shop for a **wide variety of products** from the comfort of your own home. **Integration with a variety of sellers**.
- Competitive prices
  - Amazon often offers the best prices on products, especially when you factor in discounts and promotions. **Integration with sellers**. **Just in time orders** (no inventory).
- Speedy and reliable delivery
  - **Tracking facility**. **Integration with logistic partners**.

eCommerce example

- Amazon (retail side) relies on a **large volume of sales** (they have small margins)
- As the bulk of their sales are through an **online system**, that system needs to support the business model
- Factors other than functionality impact sales
  - For every 100 ms decrease in latency Amazon's sales increase by 1%

Amazon outages reasons: https://awsmaniac.com/aws-outages/

→Mostly due to **backup failures, network errors** not handled etc. (architectural issues on availability)

Phone company example

Lack of focus on **usability**

Lack of **modifiability** – closely tied to Symbian OS

No data driven decisions

Etc.

What can be derived from business alignment?

There are factors other than the functional requirements that must influence the software:

- Performance
- Availability
- User Experience
- Modifiability
- Other –ilities

a.k.a Quality Attributes or Systemic Properties

- Other factors like data-driven decisions, innovation etc.

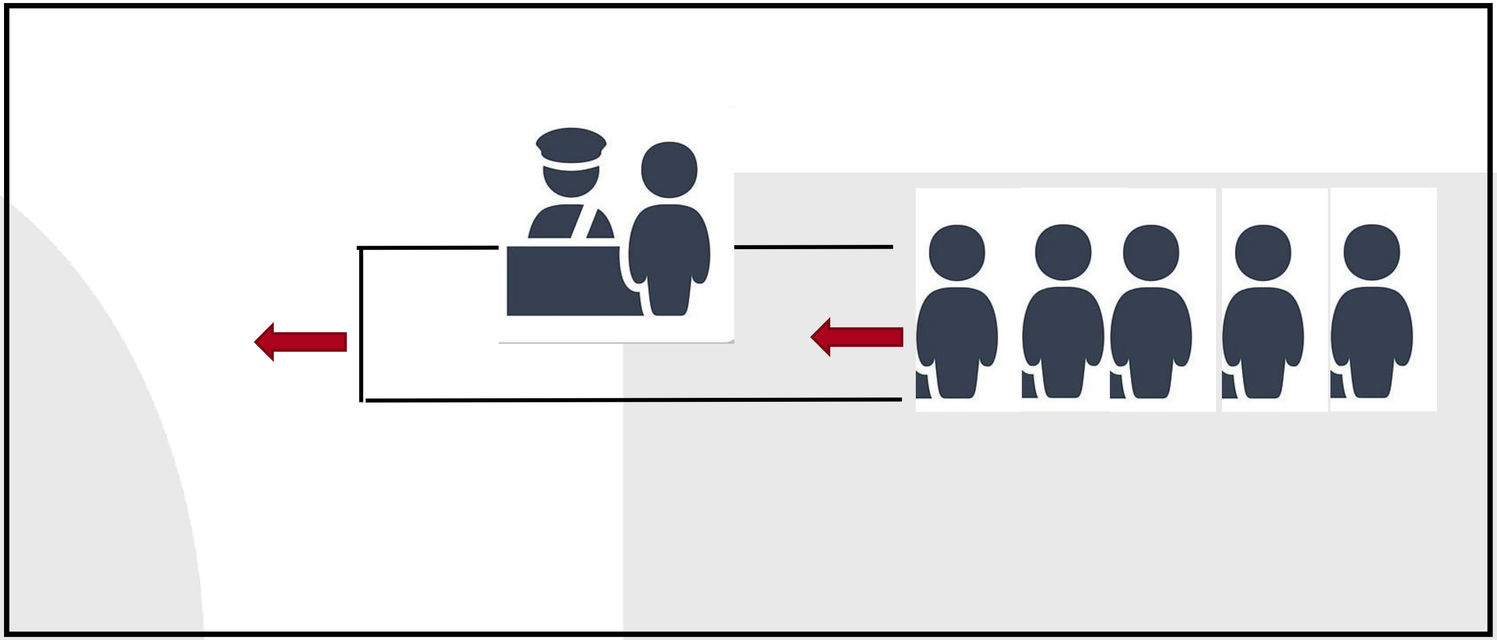
Structure Impacts Quality Attributes

- The gross ***structure**** of the system dictates what systemic properties (quality attributes) will be supported or inhibited
  - It does not, however, typically limit the functionality that can be implemented
- Let’s look at a non-software example to illustrate
  - Immigration processing at the airport

* Primarily decomposition of the code modules, for now (more later)

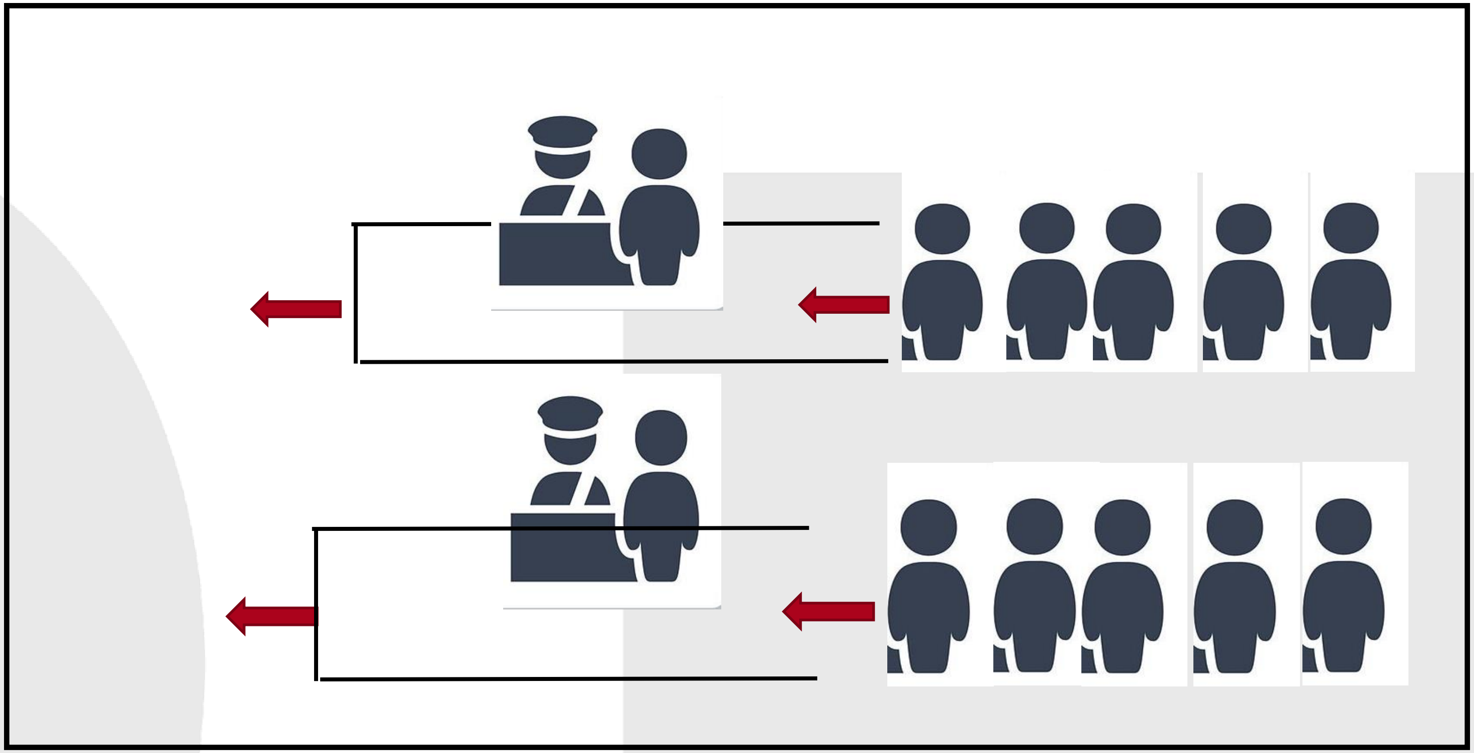
Airport immigration queues

- What could we do to get through more people in the queue in a period (throughput)?



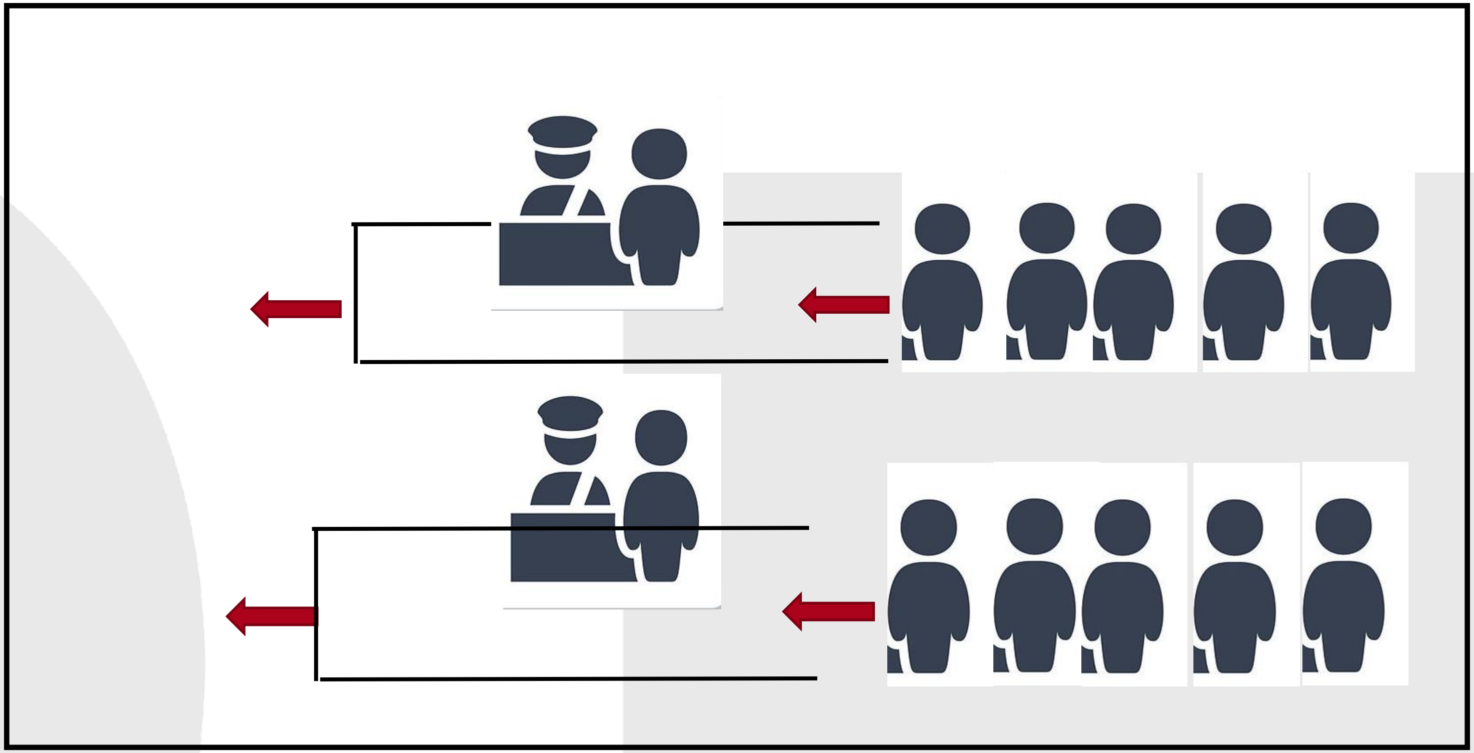
Airport immigration queues

- What could we do to reduce the time required to process your documents?



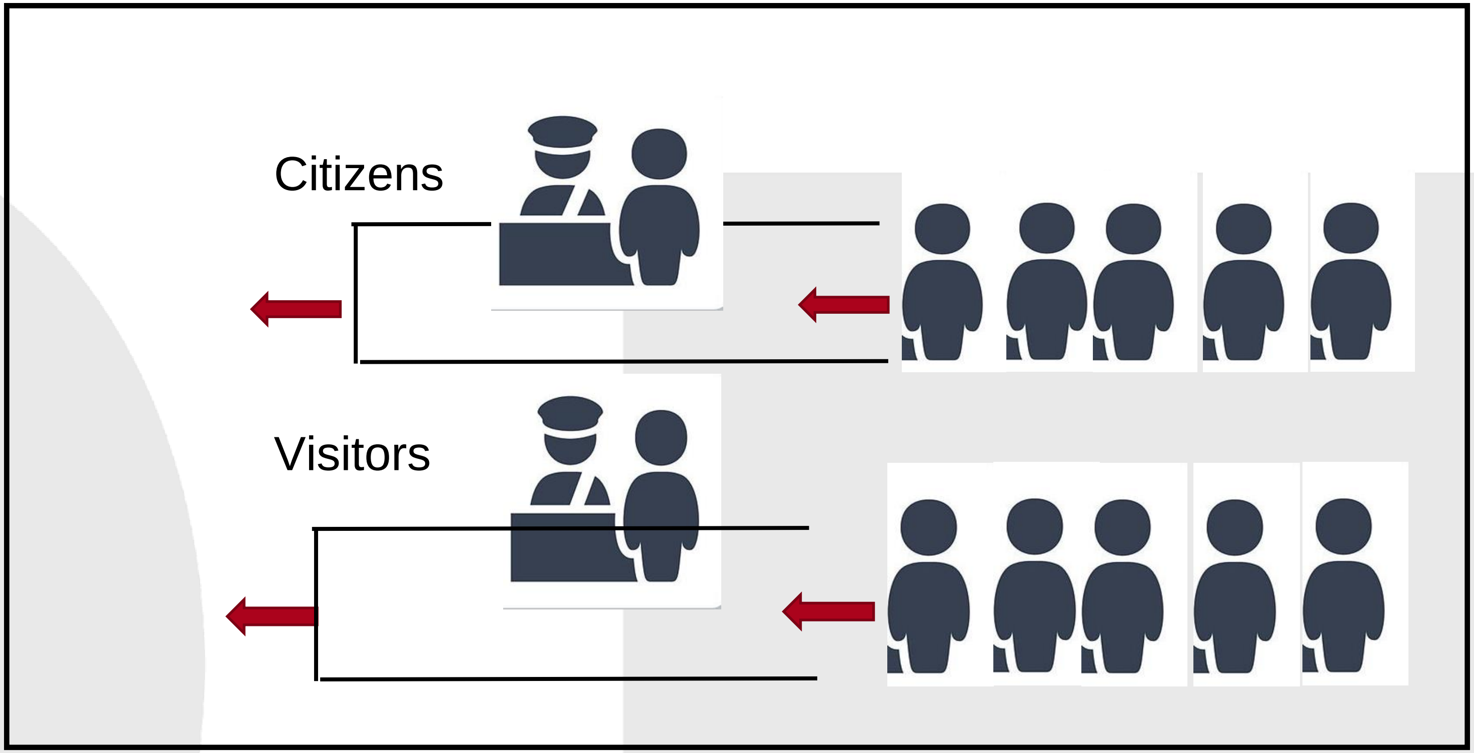
Airport immigration queues

- What could we do to reduce the time required to process your documents (latency)?



Airport immigration queues

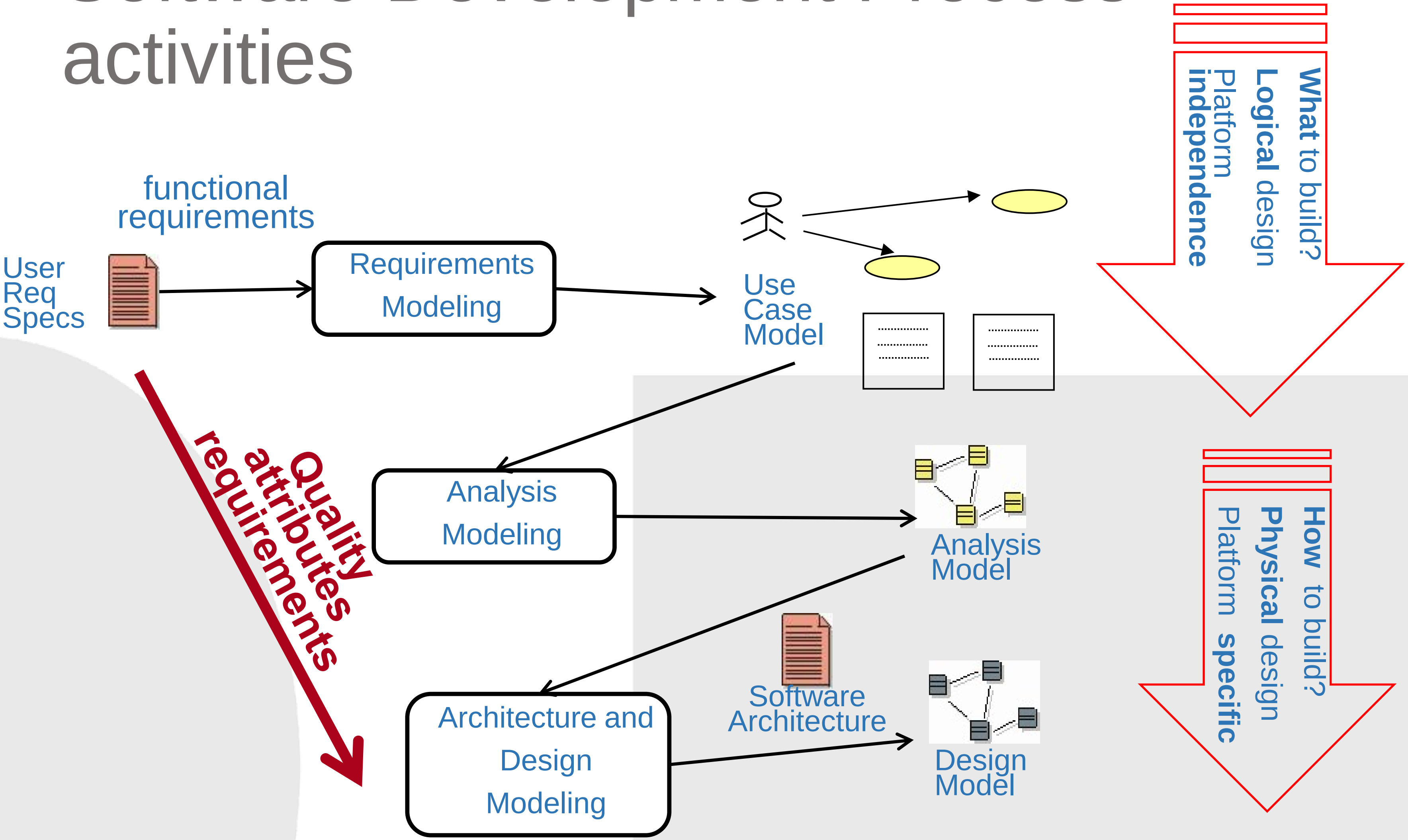
- What could we do to reduce the time required to process your documents?



Software Is Similar

- Structural decisions such as:
  - Do we have **concurrent units** of execution?
  - What **functionality** is **assigned** to which **processes**?
  - Where and how is **state managed**?
  - How is **functionality allocated** to code **modules**?
  - …
- Impact properties such as:
  - Latency (Response time)
  - Throughput (number of requests processed in a fixed time)
  - Fault tolerance
  - Modifiability
  - …

Software Development Process - activities



Topics

- Motivation for developing a Software Architecture
- What is Software Architecture
- What are common architectural activities

Software Architecture

- The **design decisions that impact systemic properties** are what we'll call "Architectural Decisions"
  - This is a working definition
- A more "formal" definition of software architecture is:

> "The software architecture of a system is the **set of structures needed to reason about the system**. These structures comprise software elements, relations among them, and properties of both."*

* Software Architecture in Practice 4th edition

Software States

Software exists in multiple states

- Static: prior to compilation and execution
  - Code modules
  - Sub systems
  - Layers
  - Etc.

Code within these units are tightly coupled

- Dynamic: running system
  - Processes
  - Threads
  - Etc.

They interact using shared data and coordination mechanisms

Why do you think it’s important to understand and think about these structures?

Elements of Software Architecture

- **Structure** of the software entities – modules, submodules, subsystems, processes, threads etc.
- Handling **cross-cutting concerns**
- **Inter-entity communication** – synchronous, asynchronous
- **Concurrent/parallel** or sequential execution
- **Deployment model** – which box/where will they reside?
- **Data sharing and transport models**
- Etc.

Software Architecture – others

- Architectural Patterns
- Reference architectures, Frameworks
- Prototyping and experiments for handling architectural risks and unknowns
- Validating the architecture
- Etc.

Laws of Software Architecture

- LAW 1:
  Everything in Software Architecture is a **trade-off**
  - Can you give some examples?
- LAW 2:
  Why is more important than how
  - How is that going to add value to the business?

Fundamentals of Software Architectures – Mark Richards & Neal Ford

Topics

- Motivation for developing a Software Architecture
- What is Software Architecture
- What are common architectural activities

Common architectural activities

- Gathering and analyzing architectural requirements (quality attributes and architectural constraints)
- Developing static, dynamic and deployment perspectives of software architectures
- Technical risk mitigations
  - Evaluating choices of platforms, frameworks, software components, algorithms etc.
  - Feasibility study, conducting experiments, prototyping
- Documenting architectural decisions and software architecture

Summary

- Business context and systemic properties supported by the system are related
- Systemic properties and set of “fundamental architectural decisions” are related
  - Contributes to the structure of the software
  - Contributes to the static, dynamic and deployment perspectives of the software
- Definition, laws and elements of software architecture

References

- Bass et al. “Software Architecture in Practice” 4th edition chapter 2