Understanding the Problem Space

17-423/723 Software System Design

Recitation 1
Jan 16, 2026

Recap: Problem vs. Solution Space

Problem vs. Solution Space

Problem space (aka **domain** or **world**)

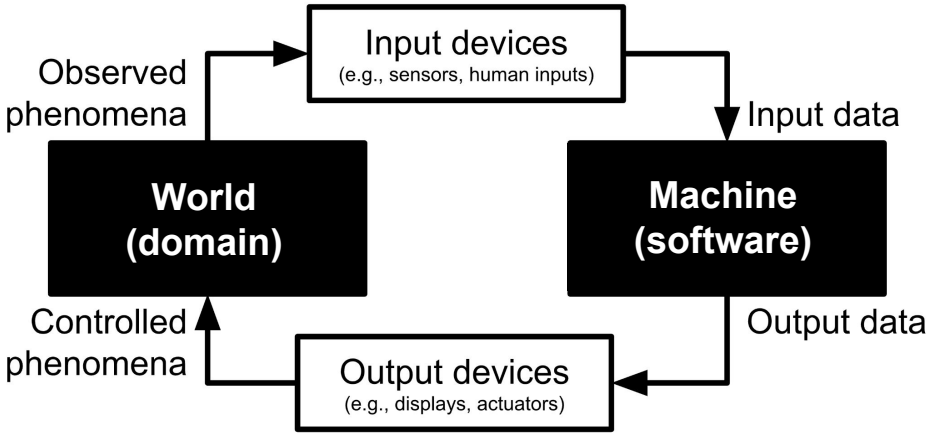
Physical entities in the real world, their behaviors & relationships

Part of the world that software may influence, but cannot directly control

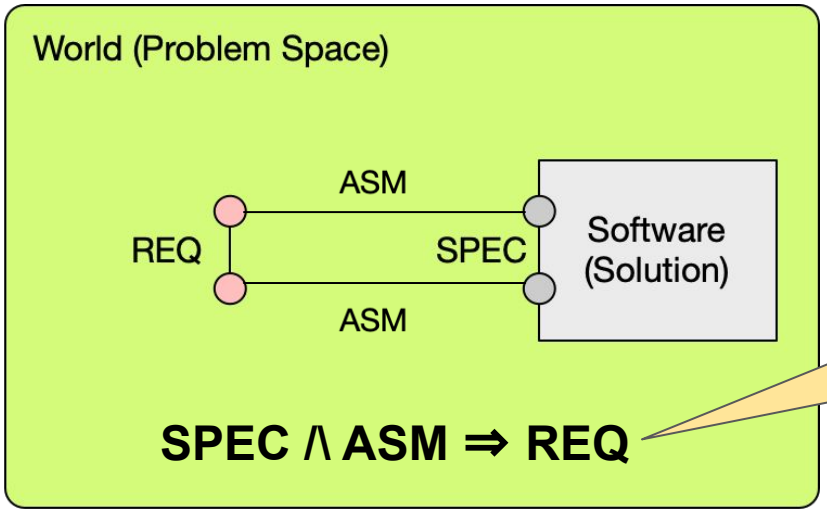
Solution space (aka **machine**)

A product (i.e., software) to be developed to solve the customers' problem

A combination of software components that you have creative control over



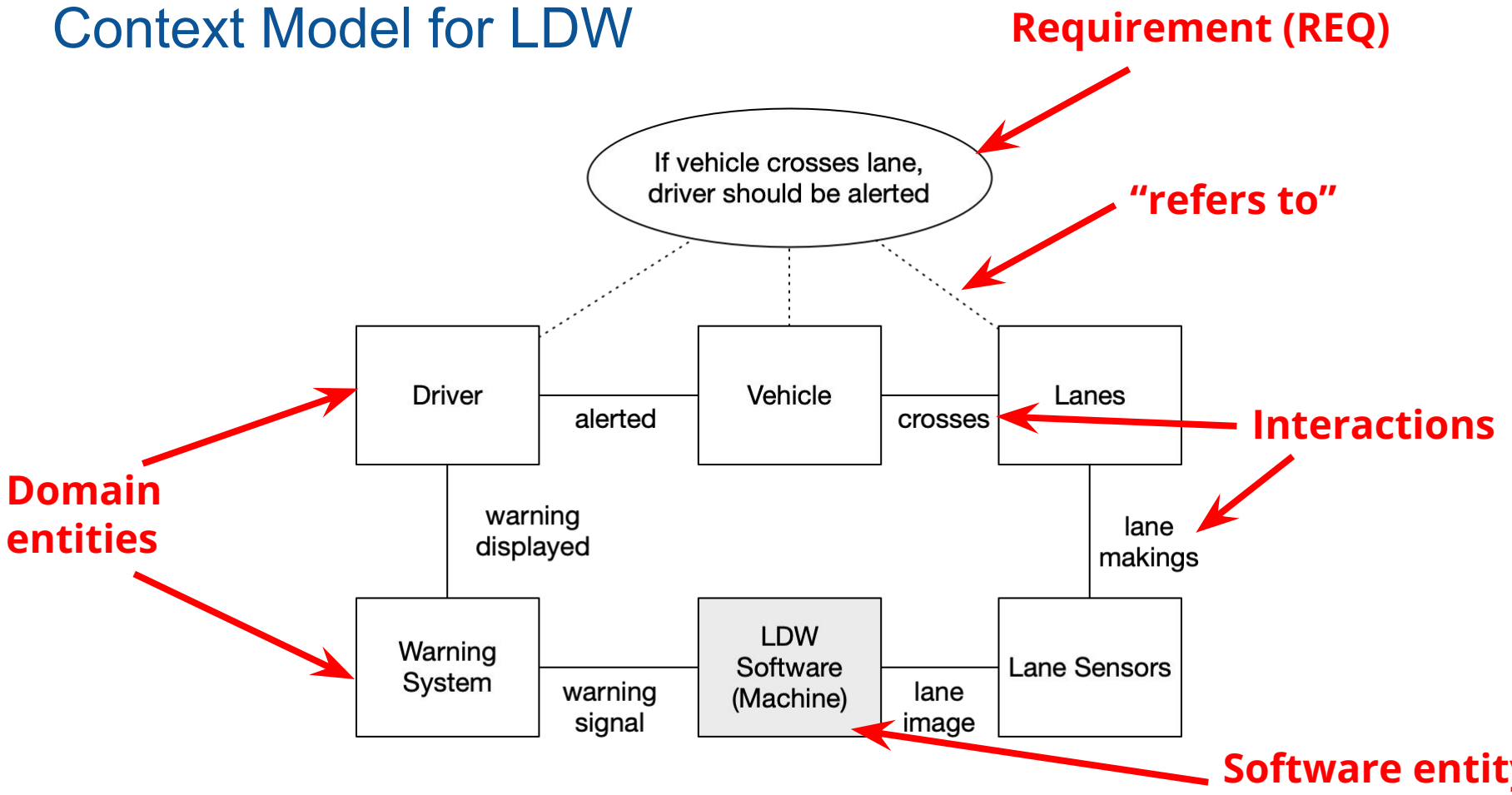
Satisfaction Argument



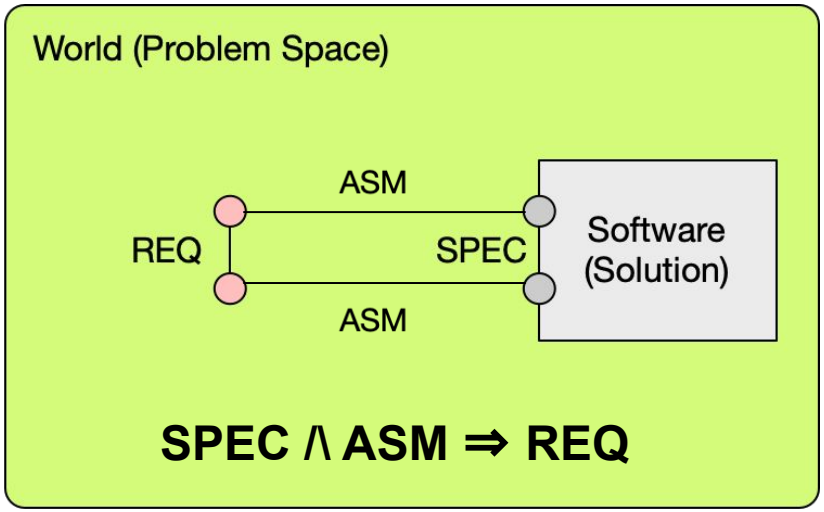
"If my software is implemented correctly (SPEC) and the world behaves as assumed (ASM), then the system achieves its requirement (REQ)"

Requirement (REQ): What the system must achieve, in terms of desired effects on the world
Specification (SPEC): What software must implement, expressed over the shared interface
Domain assumptions (ASM): What's assumed about the behavior/properties of the world; bridges the gap between REQ and SPEC

Context Model for LDW



Analyzing the context model



Apply adversarial thinking!
- What are possible ways to break the argument “SPEC /\ ASM => REQ”?
- What are possible ways to violate an assumption?

- **Missing or invalid assumptions (ASM)**
- Missing or inconsistent requirement (REQ)
- Incorrect/violated specification (SPEC)
- Inconsistent spec and assumptions (SPEC /\ ASM implies false)

A recipe for building a context model

1. State a requirement to be achieved by the system (REQ)
2. Identify entities that are referenced by the requirement
3. Identify other entities that interact with those entities in the real world
4. Connect domain entities to the software component
5. Design the specification (SPEC) on the software component that is needed to satisfy REQ
6. Identify domain assumptions (ASM) that are needed along with SPEC to satisfy REQ
7. Check whether any of the assumptions may be violated in practice
8. If so, relax ASM to reflect possible violations and design a new SPEC to ensure that SPEC /\ ASM => REQ

Case Study: Ambulance Dispatching System

Ambulance Dispatching System: Traditional Workflow

- Dispatcher receives an emergency 911 call and determines the location and severity of the incident
- Dispatcher looks up the list of nearby ambulances on a computer
- Dispatcher contacts and dispatches one of the available ambulances to the incident location
- Ambulance crew arrives at the location and treats the patient and/or transports them to a hospital

New Automated Dispatching System

- **Automatic Dispatch Software (ADS)**: The 911 operator enters the details of the incident into new software. The software decides which ambulance to allocate for the incident.
- **Automated Ambulance Localisation,** installed inside each ambulance: A GPS-based system provides the current location of the ambulance.
- **Mobile Data Terminals (MDT)**, installed inside each ambulance: The ambulance crew uses the terminal to communicate their status to the Automatic Dispatch Software (whether they are available, when they arrive at the incident scene, when they hand over the patient to a hospital, etc.,)
- **System requirement (REQ)**: Ensure the arrival of an ambulance at an incident location within 15 minutes from the 911 call.

Breakout Activity

- **Task 1:** Develop a context model for the new ambulance dispatching system. Identify the list of domain assumptions (ASM) and software specifications (SPEC) that are needed to satisfy the requirement (REQ).
- **Task 2:** Share and describe your context model to another breakout group. Looking at the other team's context model, identify assumptions that may be violated in practice.
- **Task 3:** Based on the feedback from the other team, discuss how you would modify the specifications (SPEC) to deal with the violated assumptions (ASM).

Case Study: London Ambulance System (LAS)

London Ambulance System (LAS) Project

- One of the largest ambulance systems in the world; > 2500 calls a day
- In early 1990s, a government project to convert from manual to automated dispatching system, to meet increasing demands
- **Target requirement**: For 95% of incidents, an ambulance must arrive at the incident location within 14 min
- Considered unrealistically ambitious, but government invites competing bids from software contractors
- **Project costs**: Two attempts over five years
  - First attempt failure: 7.5M pounds
  - Second attempt: 1.5M pounds, developed over 6 months

London Ambulance System (LAS) Project: Failures

- In the first few days, system works OK
- As the load increases over time, arrival delay also increases
  - > 80% of ambulances take longer than 15 min to respond
  - Some arrival takes as long as 11 hours
  - Media reports of 30+ deaths resulting from delayed arrivals
- Eventually, a complete system failure
  - Impossible to revert back to an older, manual system (already dismantled)
  - 911 operators improvise to deal with failures on an ad-hoc basis
- Chief executive of LAS resigns

LAS: Why did it fail?

- Many factors, including:
  - Poor project management
  - Accepting low-quality bid from contractors to save cost
  - Lack of proper software engineering practices
  - Lack of quality assurance and system audit
  - Lack of staff training
  - …and many others
- But also: **Poor understanding of domain assumptions!**
- For more detailed reports on the failures:
  - “Disaster in London: The LAS Case study”, Darren Dalcher (1999)
  - “Understanding Failure: The London Ambulance Service Disaster”, John Dobson (2007)

Domain Assumptions in LAS

- (ASM) The GPS gives the correct ambulance location.
  - Ambulances occasionally give inaccurate or no location information
- (ASM) When an ambulance is allocated to an incident, the ambulance crew drives the ambulance to the incident location.
  - At an EMS station, ambulance crew sometimes decide to take a different ambulance to the incident (not the one dispatched by the software)
- (ASM) When the ambulance arrives at the incident location, the ambulance crew signals arrival on their Mobile Data Terminal.
  - Ambulance crew often busy/occupied with patient treatment; slow in updating the status or forget entirely

Discussion: Gen AI for Domain Understanding

- **Q. Can generative AI be used to identify domain entities and assumptions? What are potential benefits and risks?**

Summary

- Domain assumptions are just as critical in achieving requirements
  - If you ignore/misunderstand these, your system may fail or do poorly (no matter how perfect your software is)
- Identify and document these assumptions as early as possible
- Some of the assumptions may be violated in practice
- The specification of the software should be designed with these assumptions & their possible violations in mind