

# Problem vs. Solution Space

17-423/723 Software System Design

Lecture 2  
Jan 14, 2026

# Looking Ahead

## **Next 2~3 weeks: Foundational techniques and tools for design**

Domain & design modeling, quality attributes & trade-offs, generating design ideas, design review, design processes

## **Second half of the course: Designing for quality attributes**

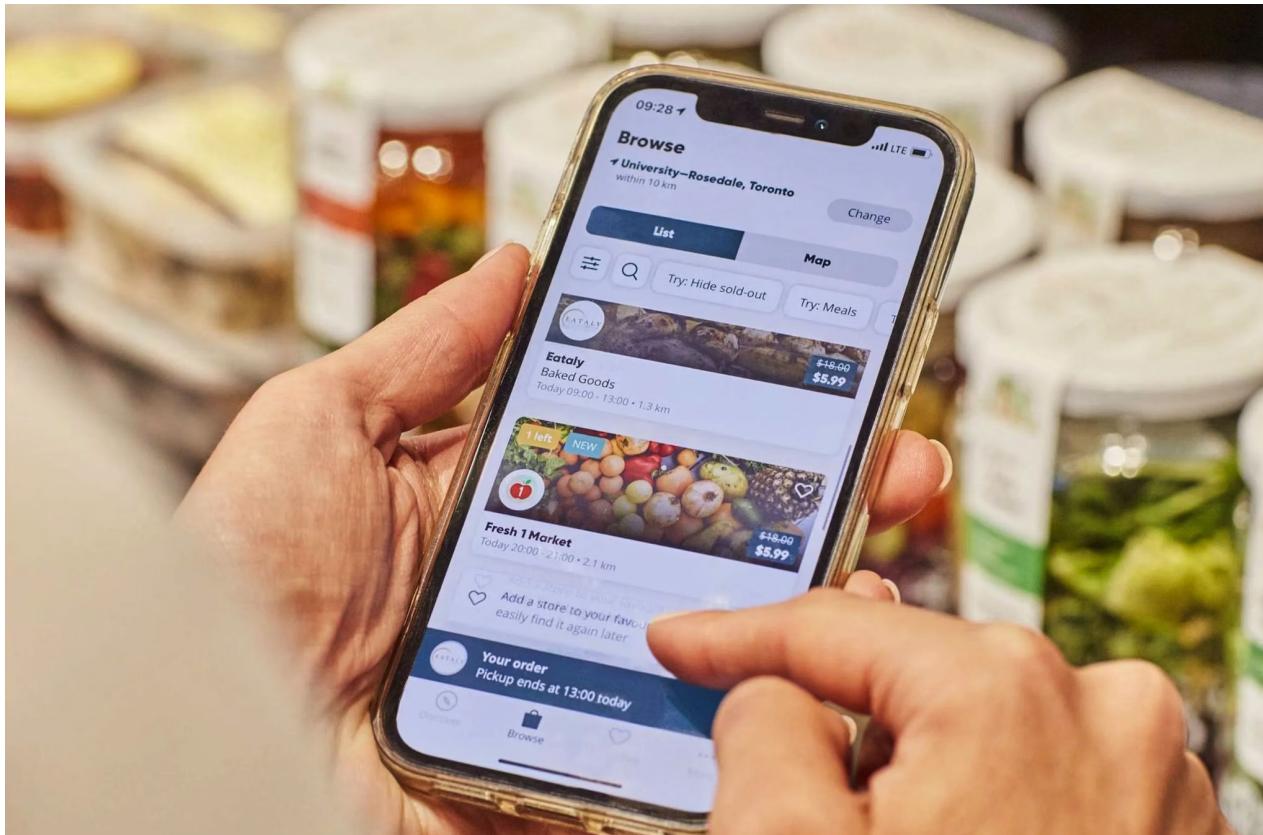
Design for change, interoperability, reuse, scalability, robustness, security, AI,...

# Combating Food Waste



~40% of all food supply in the US wasted (valued \$218 billion)

# Project Idea: Food Rescue & Distribution App



# Project Milestones

M1: Initial system design

M2: Prototype implementation & deployment

M3: Iterative design for feature extension

M4: Service integration **<- cross-team collaboration**

M5: Iterative design for additional qualities

M6: Final project presentation

**Fill out team formation survey by next lecture (Wednesday, January 21)**

# Today's Learning Goals

- Recognize the distinction between the problem and solution space
- Describe requirements, domain assumptions, and specifications for a system
- Specify a context model to communicate key elements of a domain
- Describe two perspectives on design: Design as problem solving vs. design as problem setting

# Why do software systems fail?

PIENSE

ΣΙΝΑΩΜΙΣ

THINK

सोचिए

ΣΚΕΨΟΥ

DENKE

PENSER



\$300,000

Who is Stoker?  
(FOR ONE WELCOME OUR  
NEW COMPUTER OVERLORDS)

\$ 1,000

\$1,000,000

Who is Bram  
Stoker?

\$ 17,973

\$200,000

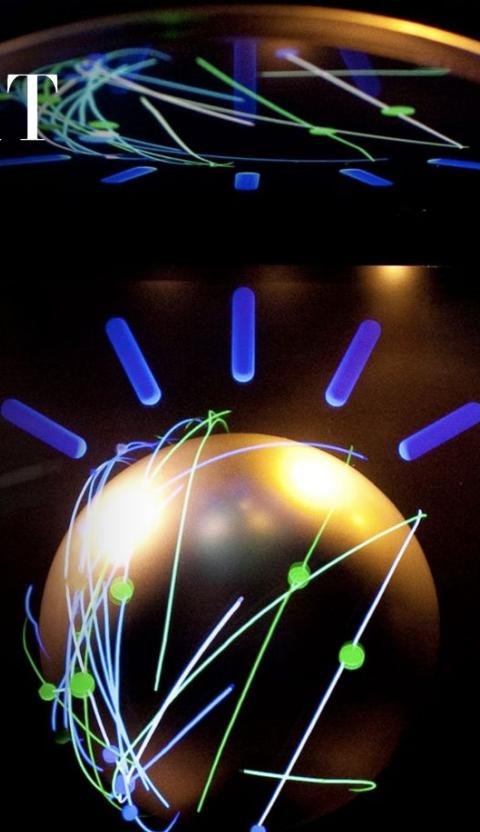
WHO IS  
BRAM STOKER?

\$5600

THE HUMAN UPGRADE

# WATSON'S NEXT FEAT? TAKING ON CANCER

IBM's computer brain is training alongside doctors to do what they can't



# How IBM's Watson Went From the Future of Health Care to Sold Off for Parts

BY LIZZIE O'LEARY JAN 31, 2022 • 9:00 AM

“We got concerns from them that the recommendations that it was giving were just not relevant...it would suggest a particular kind of treatment that wasn’t available in the locality in which it was making the recommendation, or the recommendation did not at all square with the treatment protocols that were in use at the local institution...”



11月21日 湖北 广水

# TESLA AUTOPILOT LEADS COPS ON HIGH SPEED CHASE WHILE DRIVER COMPLETELY PASSED OUT

"...police reportedly found a "steering wheel weight" in the vehicle's footwell. Such a device is used to trick Tesla's safety systems into thinking a driver's hands remain on a wheel so that the autopilot remains active..."

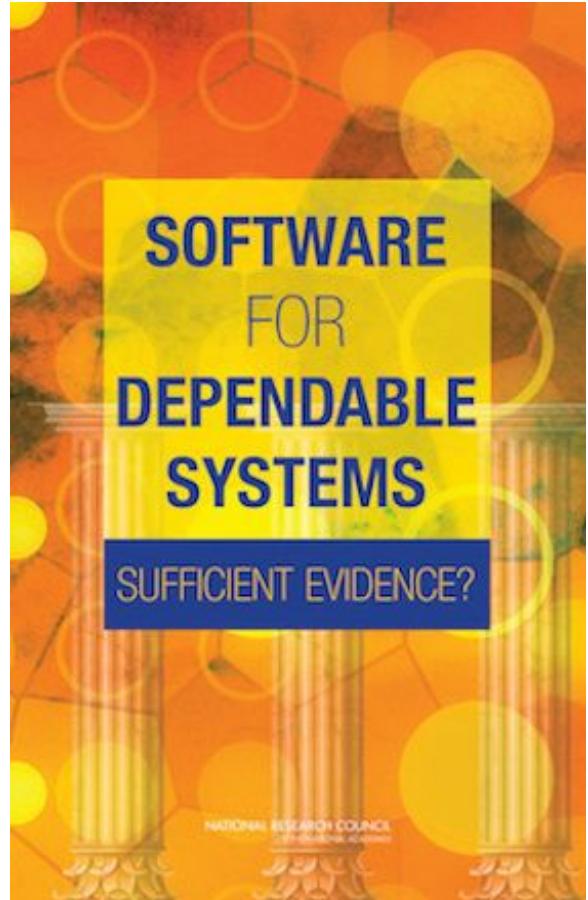
# Major Cause of Software Failures

An investigation of software-related failures by the US National Research Council

Studied failures in a wide range of domains such as aeronautics, electricity grids, automotive, healthcare, financial services

Bugs in code account only for **3%** of fatal software accidents

Most failures due to **poor understanding of system requirements or usability issues**



# Problem vs. Solution Space

# Problem vs. Solution Space

## Problem space (aka domain or world)

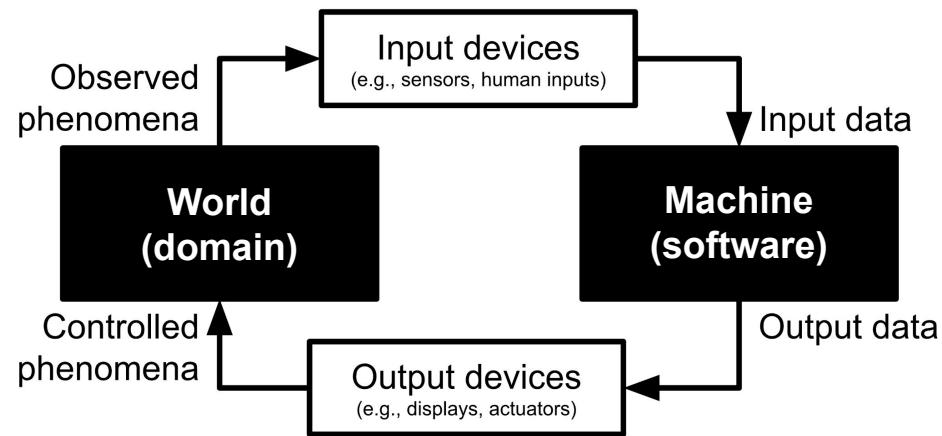
Physical entities in the real world, their behaviors & relationships

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

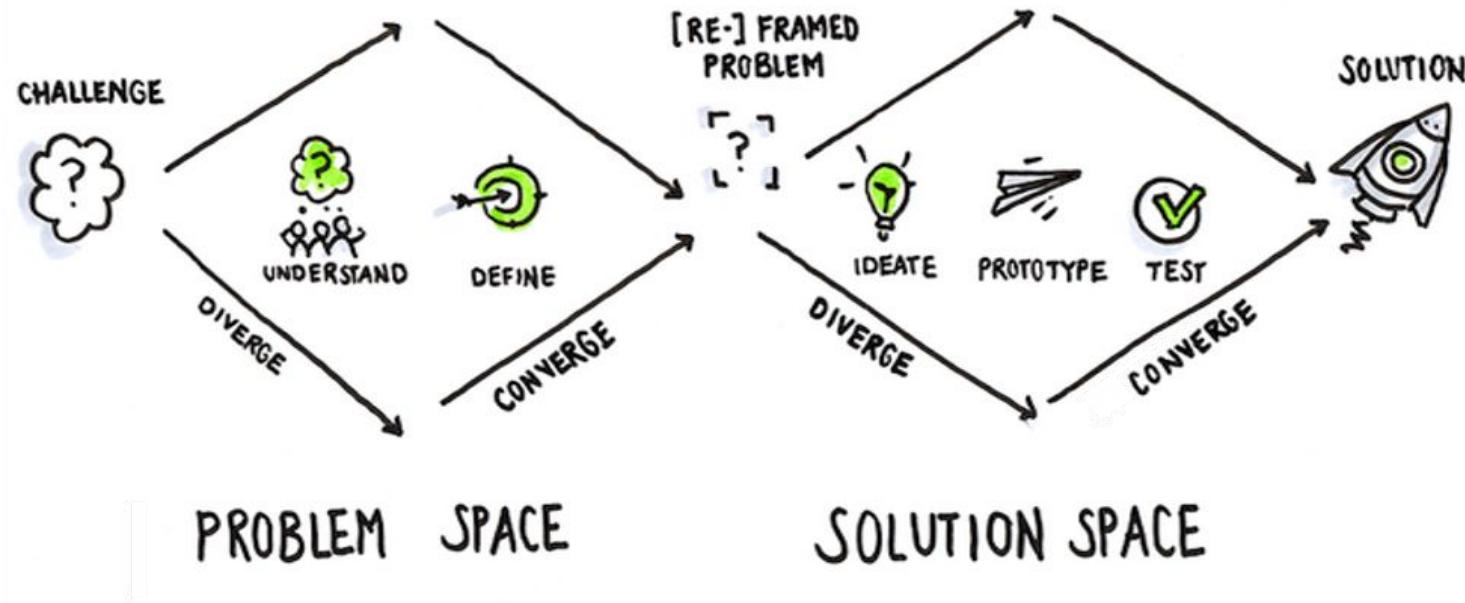
## Solution space (aka machine)

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

A combination of software components that **you have creative control over**



# Understanding problem space is a critical part of design!



# Example: Lane Departure Warning (LDW) System



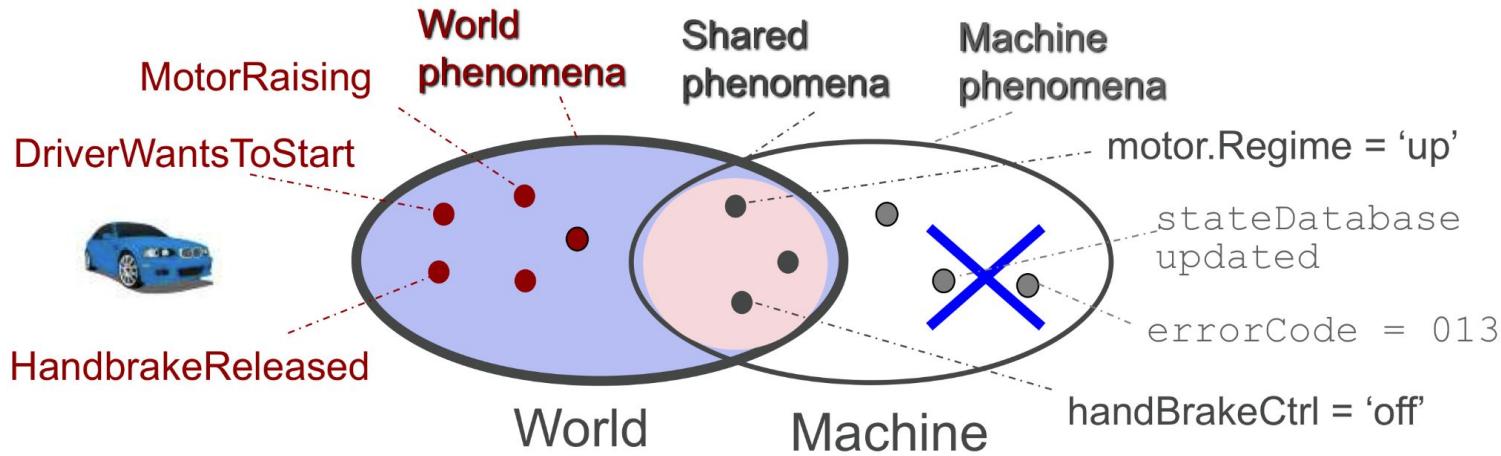
**System (LDW) Requirement:** Alert the driver by displaying a warning when the car is about to go over the lane

# Example: Lane Departure Warning (LDW) System



Q. What are different types of entities in the world (problem space)?

# Shared Phenomena



**Shared phenomena:** Interface between the world & software

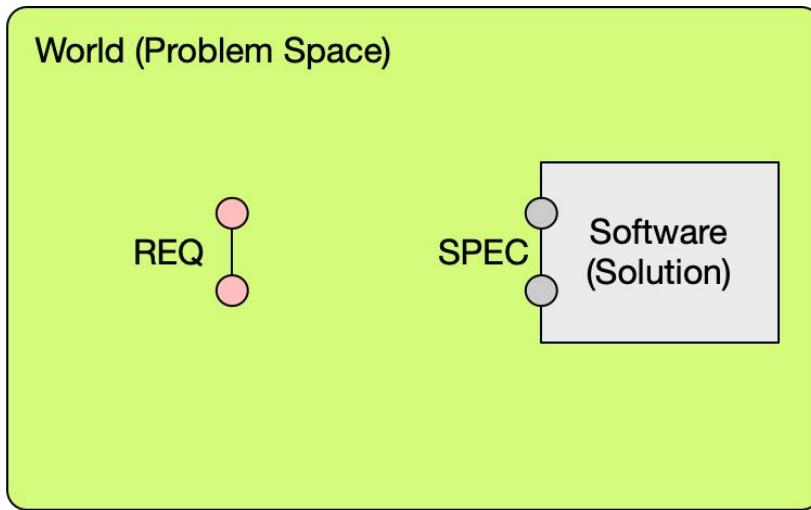
**Input:** Information generated through Lidar, camera, pressure sensors, GPS

**Output:** Signals generated & sent to the departure warning system

Software can influence the world **only through the shared interface**

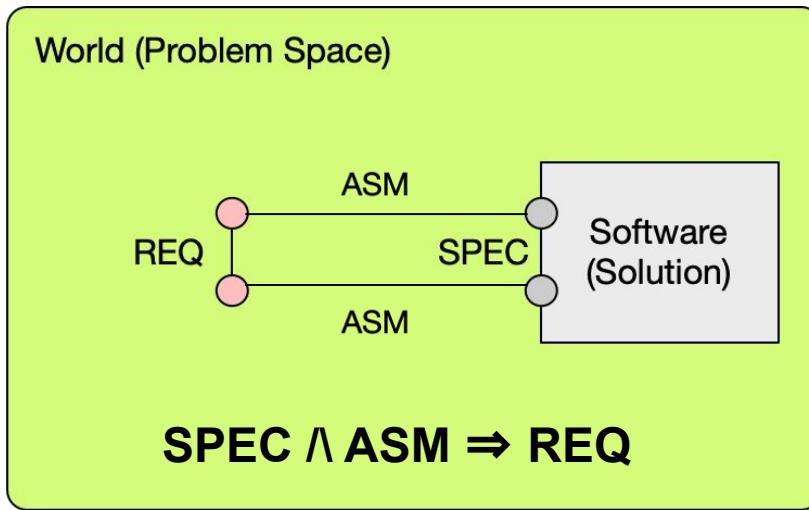
Beyond this interface, we can **only assume** how the entities in the world will behave

# Requirement, Assumptions, and Specification



**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

# Requirement, Assumptions, and Specification

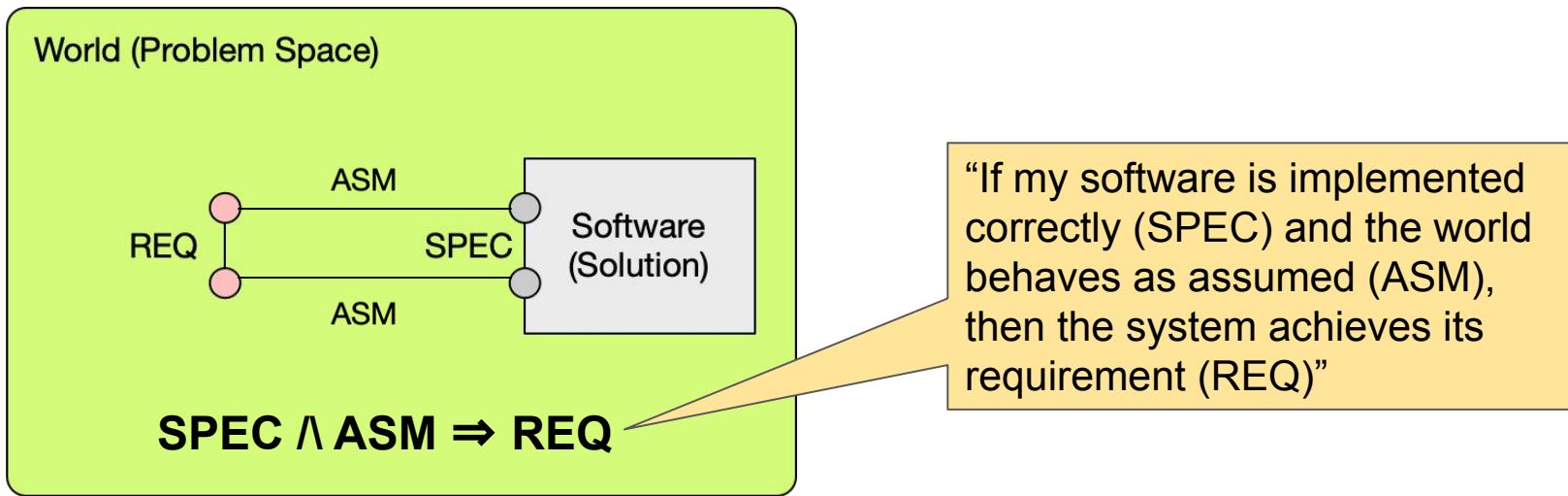


**Requirement (REQ):** What the system must achieve, in terms of desired effects on the world

**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

# Satisfaction Argument



**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

# Example: Lane Departure Warning (LDW) System



**Domain entities:** Lane markings, lane sensors, vehicle, driver

- Q.** What are assumptions (ASM) that are necessary to ensure the system requirement (REQ)?
- Q.** What is the responsibility of the LDW software (SPEC)?

# Why do we care about domain assumptions?

Missing or incorrect assumptions are a common cause of system failures

Assumptions constraint the space of possible solutions (i.e., software)

- Determine what's actually viable as a product
- Determine what the responsibility of software should be

Identifying domain entities & assumptions is one of the first steps in any design process!

## Assumptions: Other Examples

**Q. Examples from your own experience?**

## Assumptions: Other Examples

“The nurse always enters the correct amount of medicine”

“Both of the engines on a plane will not fail at the same time”

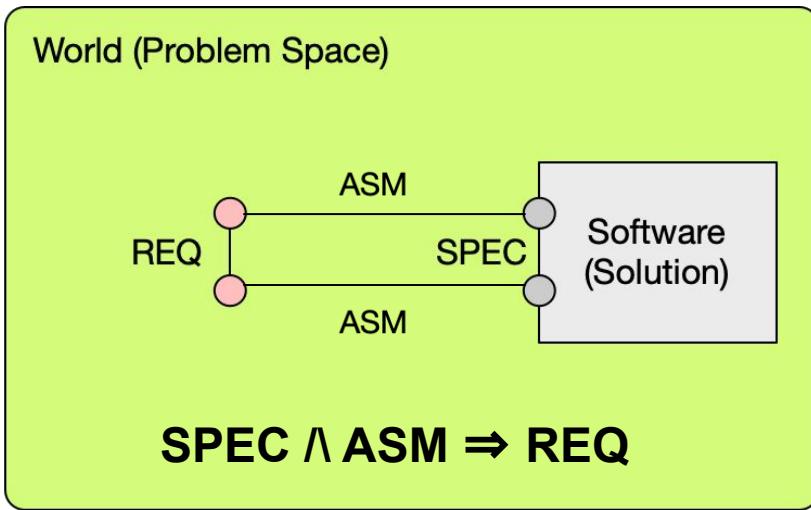
“Users will have access to a mobile device with GPS enabled”

“The battery will last for the next 300 hours”

“The user will never store the password in a plaintext on Github”

“The driver will be fully attentive and always have their hands on the steering wheel”

# What could go wrong?



- Missing or invalid assumptions (ASM)
- Missing or inconsistent requirement (REQ)
- Incorrect/violated specification - for example, due to a bug in code (SPEC)
- Inconsistent spec and assumptions (SPEC  $\wedge$  ASM implies false)

# Assumptions are often violated!



VASILIKI DOLAS VIA STORYFUL

**TESLA DRIVER CAUGHT SLEEPING AT THE WHEEL**

TEMECULA



#abc7eyewitness

## Assumption violations are a common cause of failures



Tesla is sued by drivers over alleged false Autopilot, Full Self-Driving claims

[Tesla]... has also said both technologies "require active driver supervision," with a "fully attentive" driver whose hands are on the wheel, "and do not make the vehicle autonomous."

# Another Example: Lufthansa 2904 Runway Crash



RT enabled



On ground

## Reverse thrust (RT):

Decelerates plane during landing

When plane is on the ground, the RT system should be activated

Conversely, if the plane is in the air, it is unsafe to enable RT!

## Safety requirement (REQ):

RT is enabled if and only if plane is on the ground

**Q. Suppose you want to implement this feature using software. How should the software know if the plane is on the ground?**

One answer: Check whether the wheel is turning

# Another Example: Lufthansa 2904 Runway Crash



RT enabled  $\Leftrightarrow$  On ground

RT enabled  $\Leftrightarrow$  Wheel turning  $\Leftrightarrow$  On ground



**Reverse thrust (RT):** Decelerates plane during landing

**Safety requirement (REQ):**

RT is enabled if and only if plane is on the ground

**What was implemented by software (SPEC):**

RT is enabled if and only if wheel turning

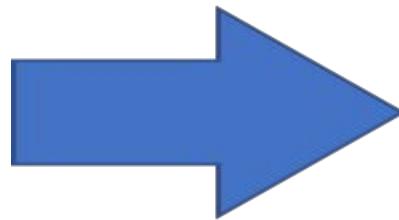
**What was assumed (ASM):**

Wheel is turning if and only if it's on ground

**But sometimes runway gets wet due to rain!**

- Wheel failed to turn fast enough when plane was on ground
- **Assumption (ASM) was violated!**
- Pilot attempted to enable RT, but it was overridden by the software
- Plane went off the runway and crashed

# Discussion: Remote Classroom



As an IT developer for the Pittsburgh Public School, you've been tasked with designing a system for enabling transition to remote education during pandemic.

**Q.** What are different entities and stakeholders in the problem space?

**Q.** What are domain assumptions (ASM) needed to ensure that every student continues to receive education (REQ)?

**Pittsburgh Public Schools Forges Ahead With Online-Only Classes, Even As Thousands Of Students Don't Have Computers Yet**

**Online Classes In Pittsburgh Public Schools And Trinity Area School District Interrupted By Inappropriate Videos And Messages**

**Pittsburgh Public Schools Providing Grab And Go Meals For First Nine Weeks Of School**

# Communicating Domain Knowledge

# Domain Models

Textual/graphical descriptions of the domain (i.e., problem space), depicting its key entities and assumptions

## Why build a domain model?

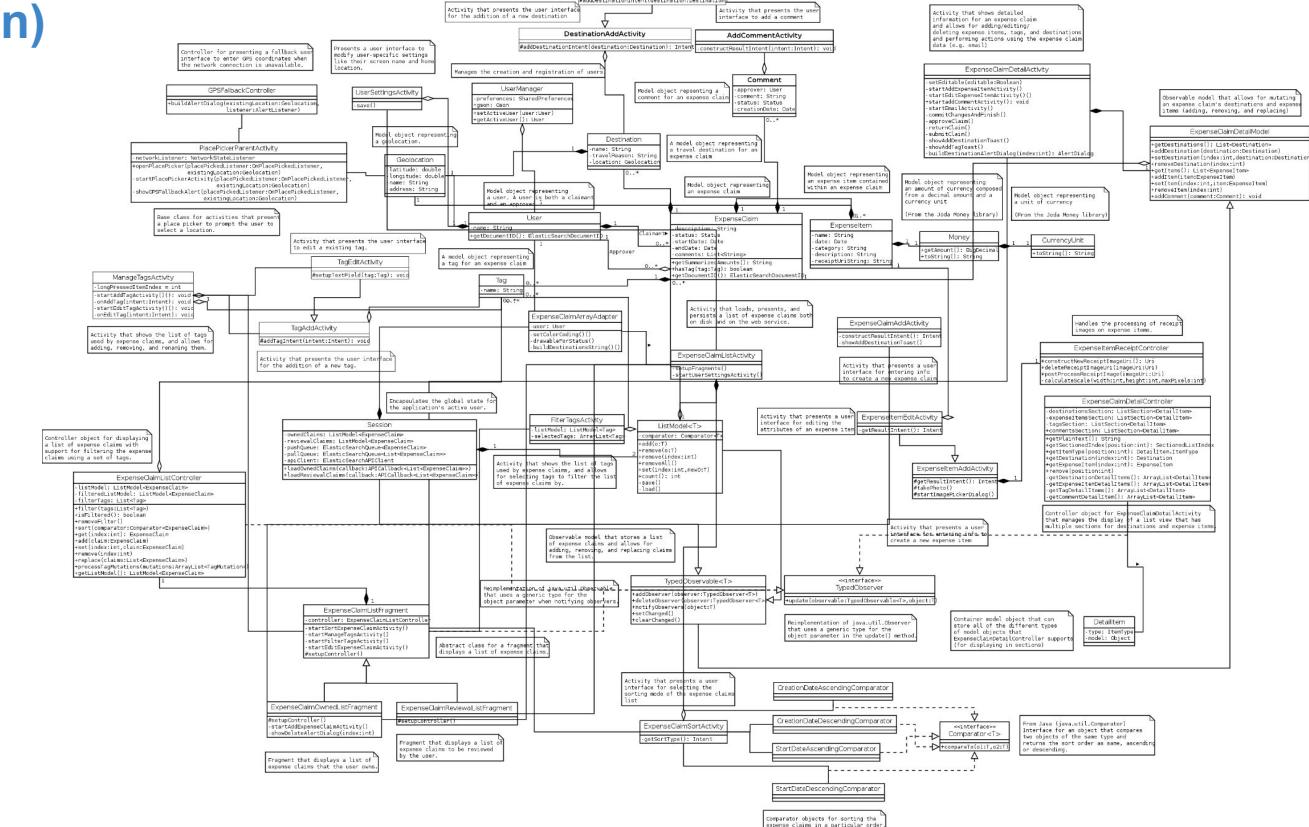
Establish common terminology & assumptions with stakeholders and other teams

Analyze to identify missing entities or missing/incorrect assumptions

Serve as a reference for subsequent design decisions

In this class, we will use a type of domain model called a **context model**

# (Digression)



There are many notations/diagrams/approaches for specifying models (e.g., UML)  
In this class, we will stick to simple, lightweight notations for modeling  
Our goal is to communicate ideas, not to document everything!

# Context Model: Elements

## Entity

Stakeholders, users, physical objects, conceptual objects

One **single, special “machine” entity** is reserved for the software component

## Interaction

A relationship between a pair of entities: Physical interactions, events, information, etc.,

## Requirement (REQ)

A relationship among entities that is desired, but not satisfied yet; a system will be designed & built to make this true

## Assumptions (ASM)

A statement about a property or behavior of an entity, necessary for satisfying a requirement

## Specification (SPEC)

A statement about the responsibility of the machine (i.e., software)

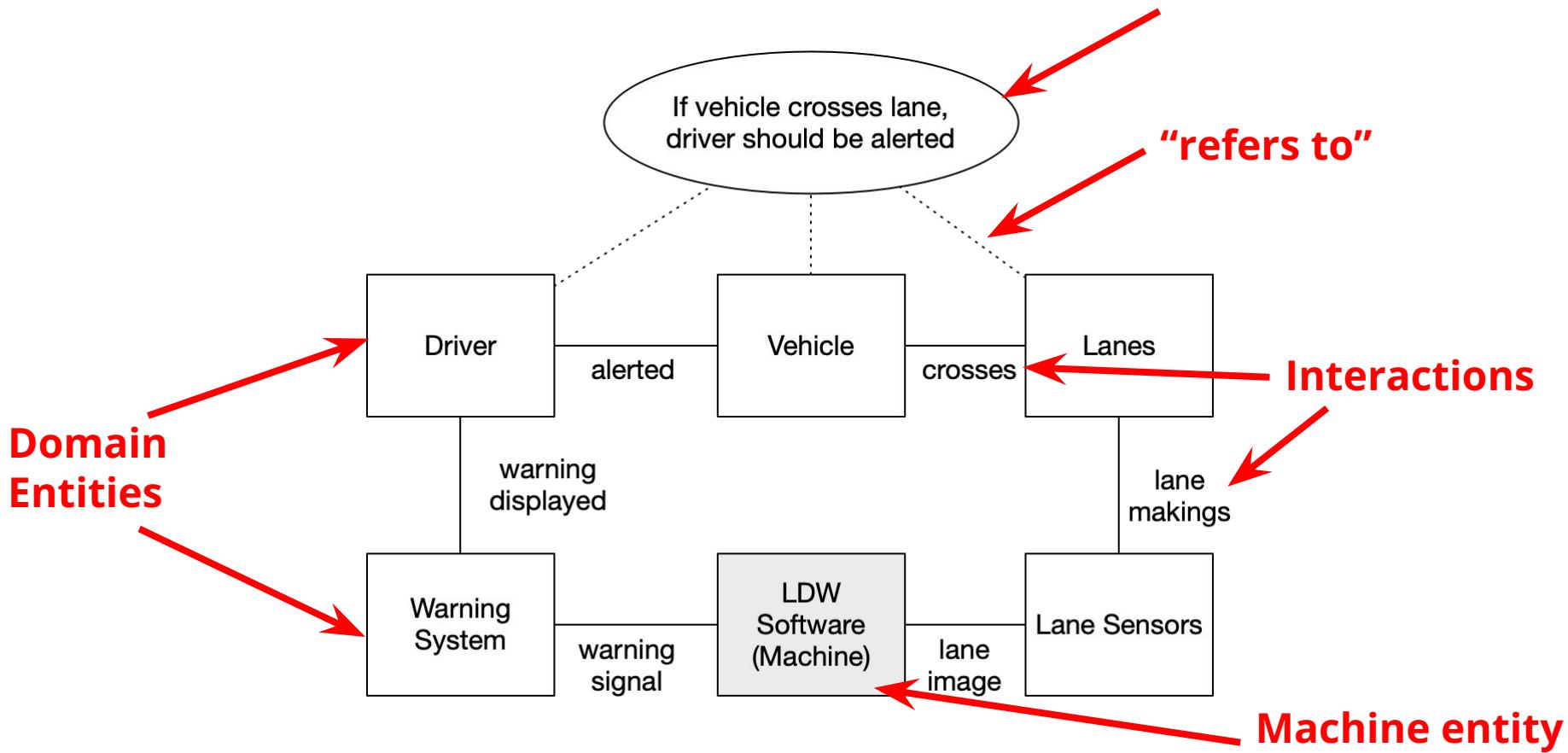
# Example: Lane Departure Warning (LDW) System



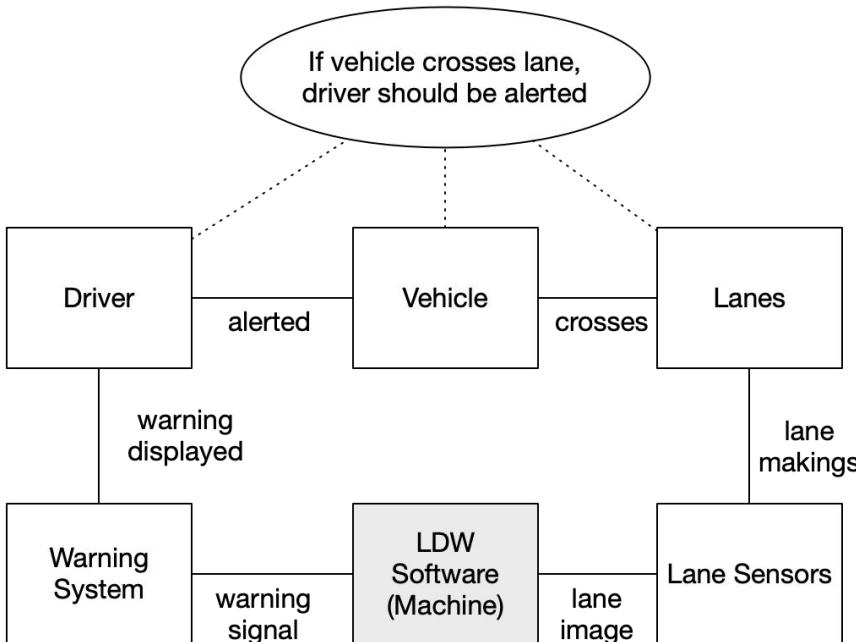
**Requirement (REQ):** If the vehicle is about to cross the lane, alert the driver by displaying a warning

# Context Model for LDW

Requirement (REQ)



# Context Model for LDW



## Assumptions (ASM)

**Lanes:** Lane markings are continuous and clearly visible

**Lane sensors:** Sensors capture the lane markings at a sufficiently high resolution

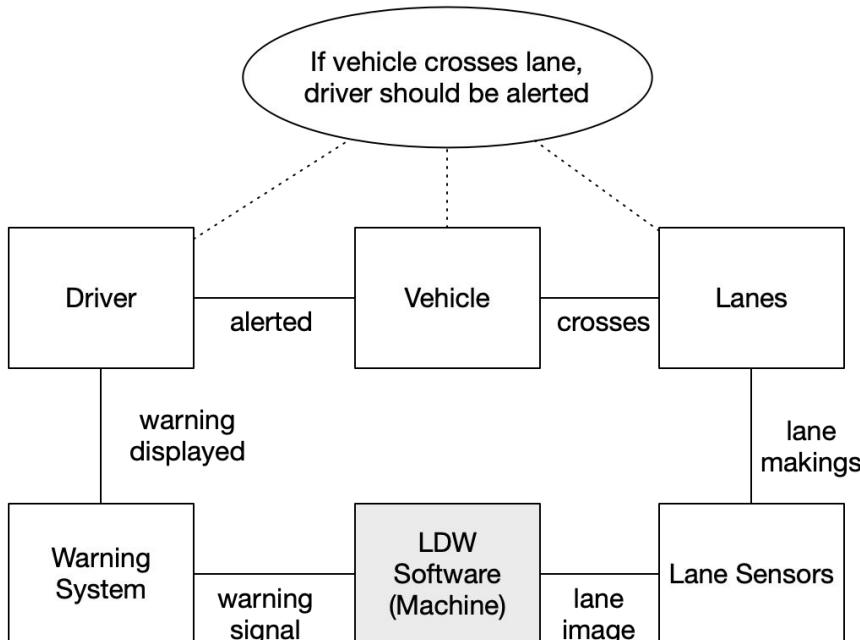
**Driver:** Driver is attentive of the dashboard display

**Warning system:** Given a warning signal, it immediately displays the warning on the dashboard

## Specification (SPEC)

LDW software shall detect lane departure in input images & generate a warning signal if detected

# Context Model for LDW



**Q.** Why isn't the machine (LDW software) entity directly connected to the lanes or drivers?

**Q.** Why isn't the warning system part of the machine (software) entity?

**Q.** Why didn't I include an entity for the steering wheel?

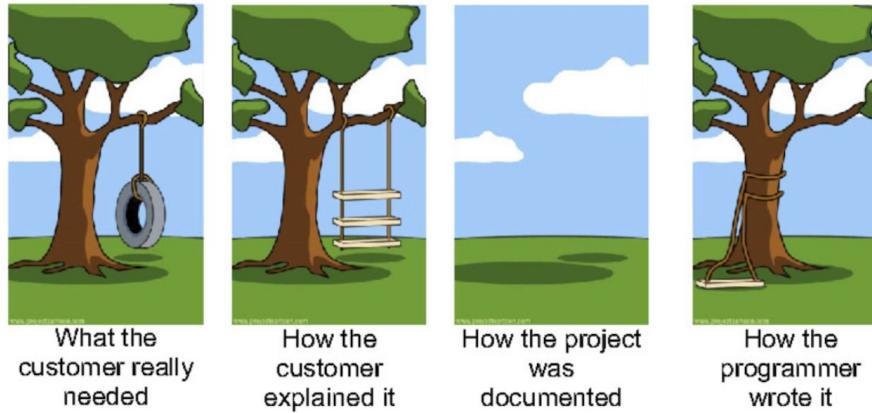
## Machine Entity

One **single, special “machine” entity** is reserved for the software component

**Q. Why don't we further describe this in detail by breaking it into multiple entities?**

**A. Avoid premature design decisions!** At this point, we are still trying to understand the problem space.

# Where do domain assumptions come from?



**Short answer:** From talking to stakeholders, users, and domain experts!

There is an entire subarea of software engineering called **requirements engineering**, which studies how to elicit, specify, and analyze requirements from the problem domain

In this class, we will not go deep into this topic (but we will come back to it from time to time!)

# Analyzing the context model

World (Problem Space)



**SPEC  $\wedge$  ASM  $\Rightarrow$  REQ**

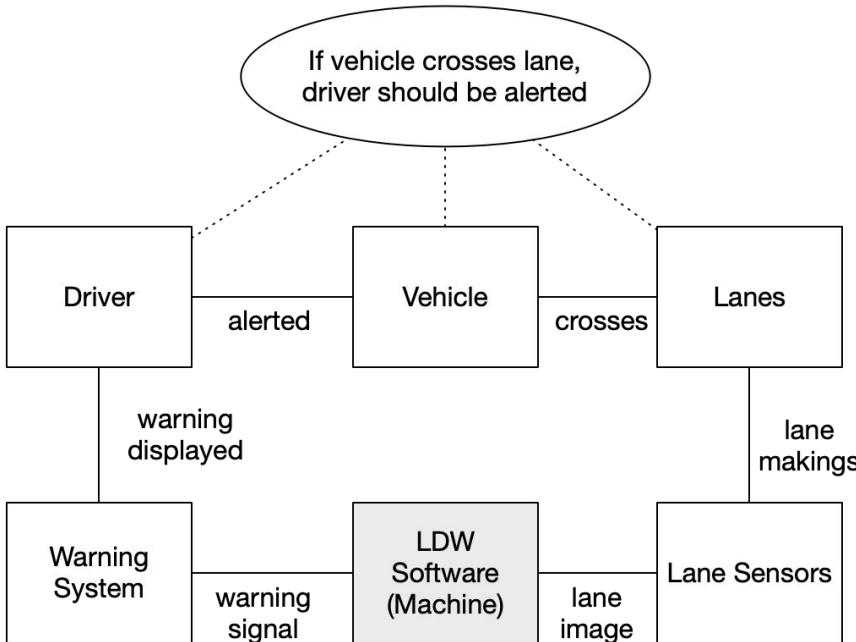
**Apply adversarial thinking!**

- What are possible ways to break the argument “SPEC  $\wedge$  ASM  $\Rightarrow$  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  $\wedge$  ASM implies false)

# Context Model for LDW



## Assumptions (ASM)

**Lanes:** Lane markings are continuous and clearly visible

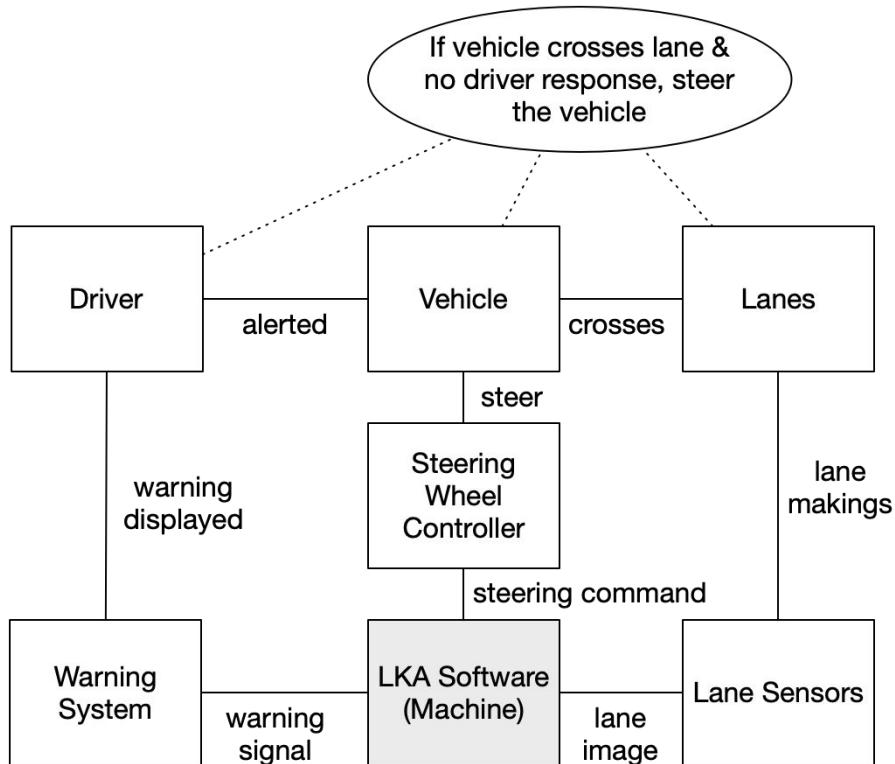
**Lane sensors:** Sensors capture the lane markings at a sufficiently high resolution

**Driver:** Driver is attentive of the dashboard display

**Warning system:** Given a warning signal, it immediately displays the warning on the dashboard

**Q.** Which of these assumptions may not hold in practice? What's a more reasonable assumption? How does this change the responsibility of software?

# Context Model for Lane Keeping Assist (LKA)



## Assumptions (ASM)

**Driver:** Driver might not be fully attentive of the dashboard display

**Steering Wheel Controller:** It steers the vehicle based on the command received

## Specification (SPEC)

LKA software shall detect lane departure in input images & generate a warning signal; if the driver does not respond in time, it shall generate a command to steer the vehicle back into the lane

# A recipe for building a context model

More in  
recitation!

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  $SPEC \wedge ASM \Rightarrow REQ$

# Closing Thoughts

Missing or incorrect assumptions are a common cause of system failures

Assumptions constraint the space of possible solutions and determine the responsibility of software

In most cases, it is impossible to come up with a complete list of domain assumptions (**Q. Why?**)

But the process of identifying and documenting assumptions can help (i) reduce potential risks of failures and (ii) communicate domain knowledge to stakeholders and other team members

# Two Perspectives on Design

# Perspective: Design as Problem Solving

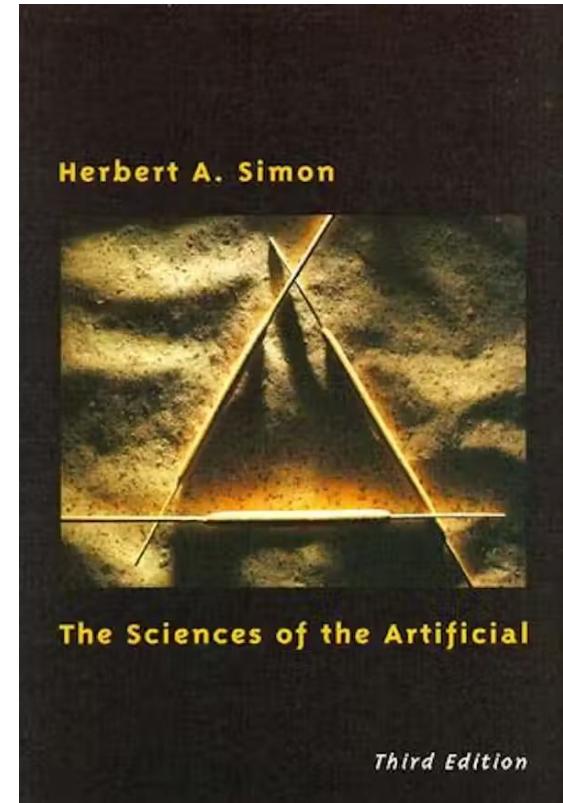
Design is a **systematic, rational** process

A description of a problem space & constraints (i.e., assumptions) is given

Designer makes a sequence of design decisions

Each candidate solution is evaluated until a satisfactory design is found

Simon hinted that one day, this process could be automated by computers



# Perspective: Design as Problem Setting

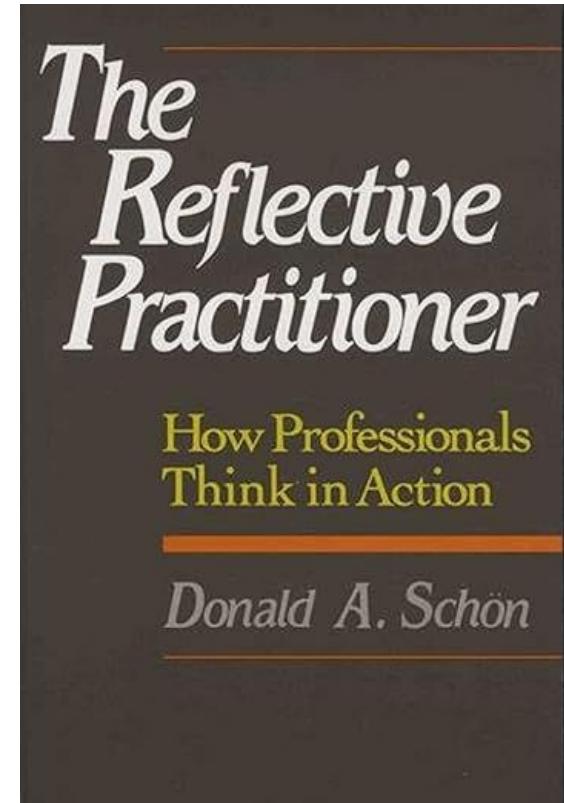
**Design is a conversation between the problem space & the designer**

Simon's model is flawed; designers don't actually work like this in practice

As the designer explores possible solutions, they learn more about the problem itself

Outcome of design is both the product & also an increased understanding of the problem space

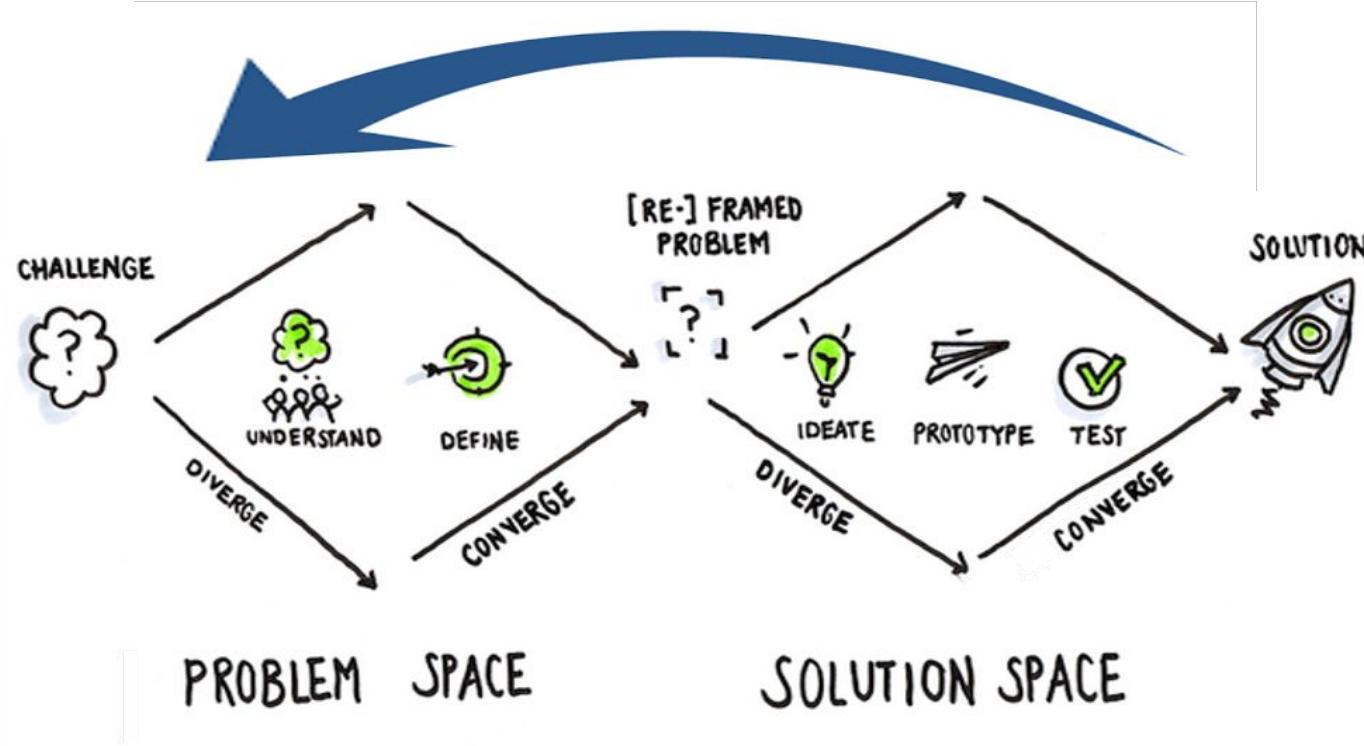
This is unlikely to be fully automatable by computers



## Discussion: Simon vs. Schön

**Q. Which one do you think is the “right” model of design?**

# Design is an iterative, continuous process!



# Summary

- Exit ticket!

# Further Readings on Problem & Requirements Understanding

- *Problem Frames: Analysing and Structuring Software Development Problems.* Michael A. Jackson. Addison-Wesley, 2000.
- *Requirements Engineering: From System Goals to UML Models to Software Specifications.* Axel van Lamsweerde. Wiley, 2009.
- *Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices.* Michael A. Jackson. Addison-Wesley Professional, 1995.
- *Domain-driven Design: Tackling Complexity in the Heart of Software.* Eric Evans. Addison-Wesley Professional, 2003.

# Summary

- Software alone cannot fulfill system requirements
  - They are just one part of the system, and have limited control over the world
- 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 over time as the world evolves
- You are never done with identifying assumptions