

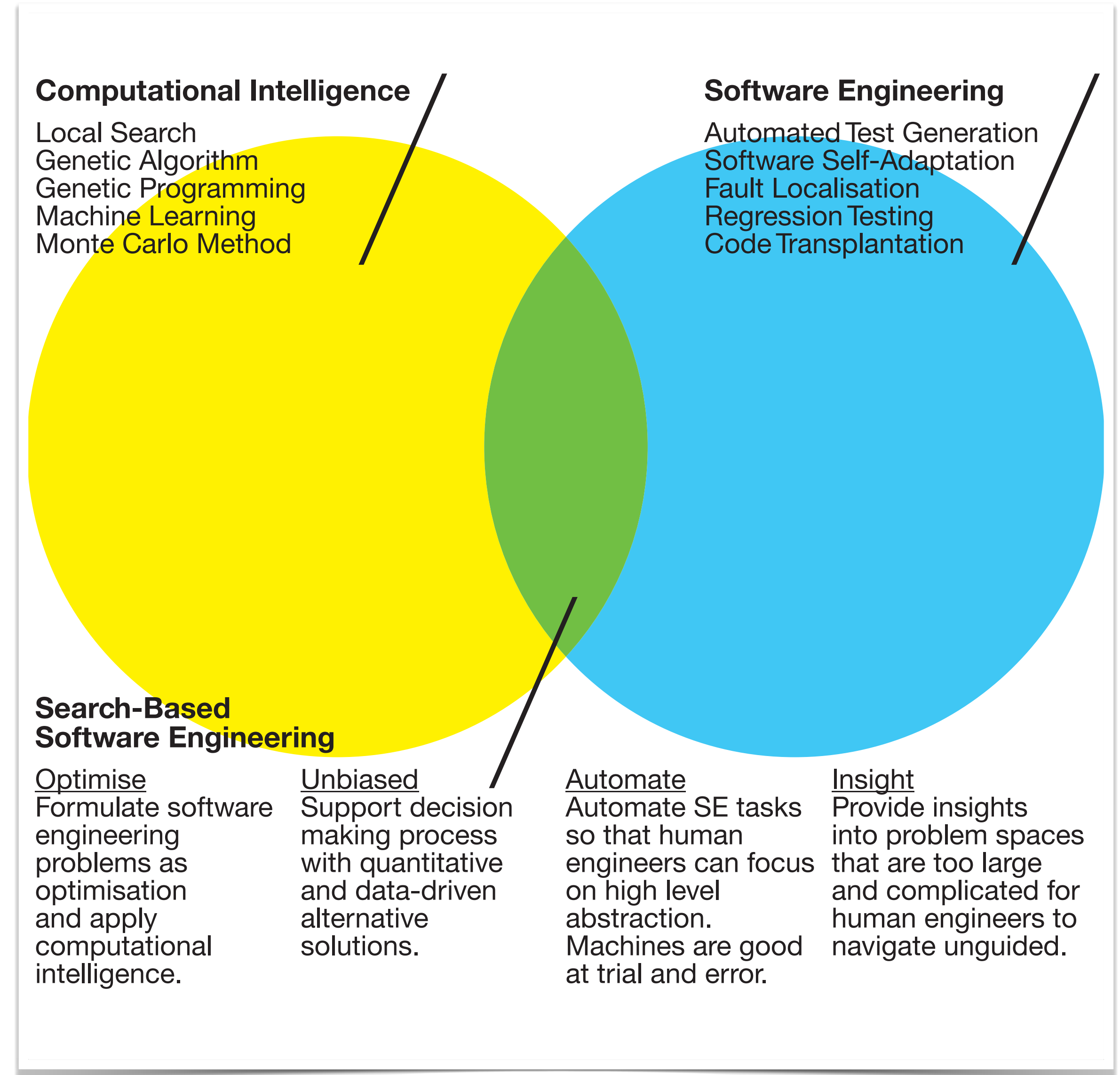
# **Introduction**

**CS454 AI-Based Software Engineering**

**Shin Yoo**

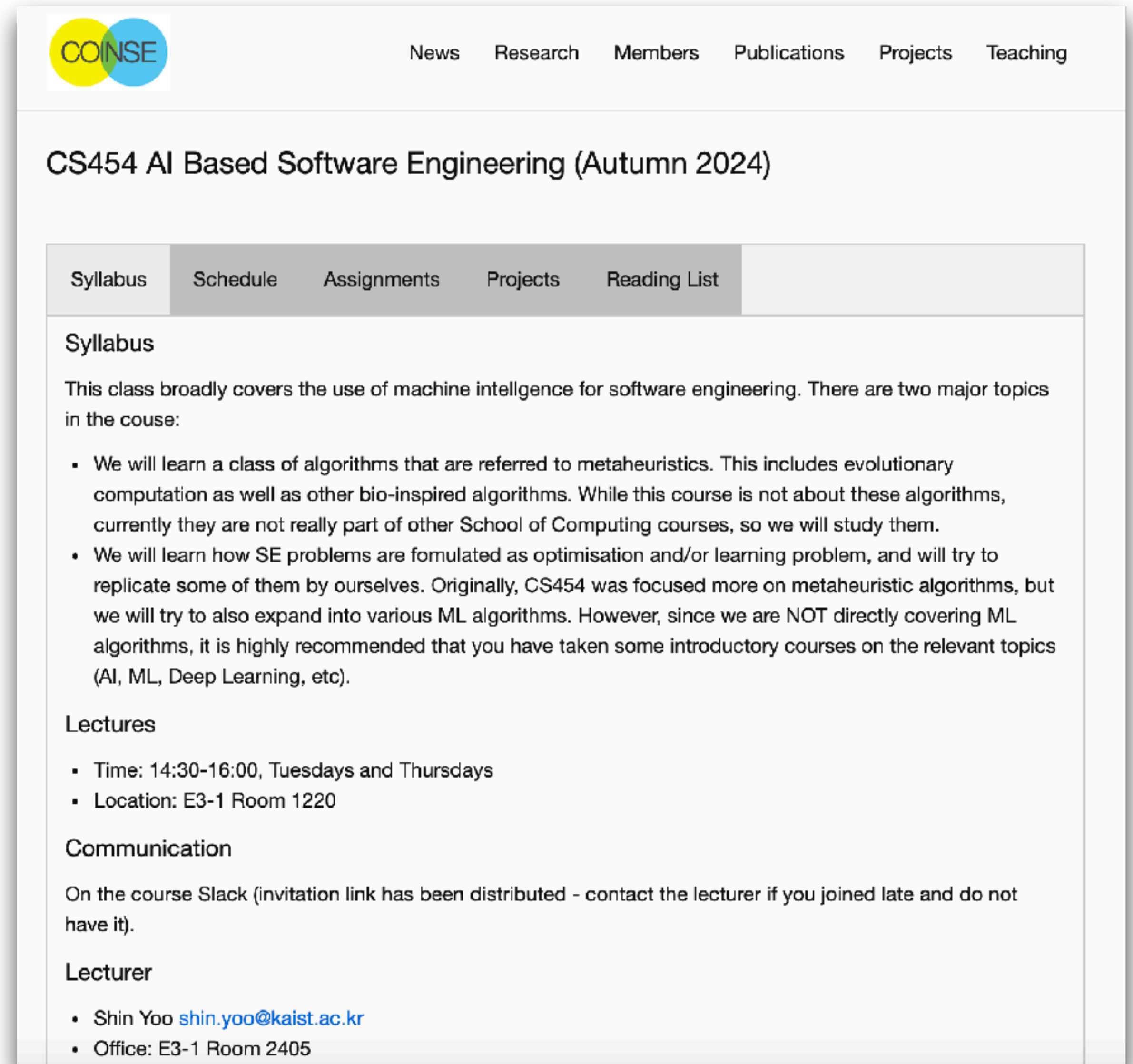
# Me

- Shin Yoo
  - PhD at King's College London, UK
  - Assistant Professor at University College London, UK
- COINSE (Computational Intelligence for Software Engineering) Lab
- Research interest: SBSE, regression testing, automated debugging, evolutionary computation, information theory, program analysis...
- [shin.yoo@kaist.ac.kr](mailto:shin.yoo@kaist.ac.kr)



# Course Webpage

- Reading materials will be either linked or provided on the page
- <http://coinse.github.io/teaching/2024/cs454/>
- Still being updated - sorry!



The screenshot shows the course webpage for CS454 AI Based Software Engineering (Autumn 2024). The page has a header with the COINSE logo and navigation links: News, Research, Members, Publications, Projects, and Teaching. Below the header, the course title is displayed. A tabbed interface shows the 'Syllabus' tab selected. The syllabus content includes an introduction, two main topics, lecture details, communication information, and lecturer details.

COINSE

News Research Members Publications Projects Teaching

## CS454 AI Based Software Engineering (Autumn 2024)

Syllabus Schedule Assignments Projects Reading List

### Syllabus

This class broadly covers the use of machine intelligence for software engineering. There are two major topics in the course:

- We will learn a class of algorithms that are referred to metaheuristics. This includes evolutionary computation as well as other bio-inspired algorithms. While this course is not about these algorithms, currently they are not really part of other School of Computing courses, so we will study them.
- We will learn how SE problems are formulated as optimisation and/or learning problem, and will try to replicate some of them by ourselves. Originally, CS454 was focused more on metaheuristic algorithms, but we will try to also expand into various ML algorithms. However, since we are NOT directly covering ML algorithms, it is highly recommended that you have taken some introductory courses on the relevant topics (AI, ML, Deep Learning, etc).

### Lectures

- Time: 14:30-16:00, Tuesdays and Thursdays
- Location: E3-1 Room 1220

### Communication

On the course Slack (invitation link has been distributed - contact the lecturer if you joined late and do not have it).

### Lecturer

- Shin Yoo [shin.yoo@kaist.ac.kr](mailto:shin.yoo@kaist.ac.kr)
- Office: E3-1 Room 2405

# Course Evaluation

- **Assignments (70%)**
  - Four Individual Courseworks, each with slightly different weighting (to be announced)
  - Each assignment is a scaled down real SE problem that you need to solve with optimization/AI techniques. We will grade based on the following criteria:
    - Quality of the solution
    - Performance of the solution
    - Quality of the implementation & SE practices
  - Participate in Slack discussion + Q&A
  - **Late submission: 0.7 weight if submitted within a week. Submission will not be accepted if it is late more than a week.**

# Course Evaluation

**New this year!**

- **Project (30%):** put the knowledge you obtained during the class to actual use.
  - Proposal presentation: outline your idea, get feedback.
  - Final report: submit your implementation and empirical evaluation.
  - Teams of four: start talking about teams right now, I will set up a Google Sheet where you can register your team members.
- **No Exam!**

# Textbook & reading material

- No textbook (we will read up to the bleeding edge)
- Lectures contain strongly recommend reading lists
- Supplementary books:
  - Introduction to Evolutionary Computing, *A. E. Eiben & J. E. Smith*, Springer
  - A Field Guide to Genetic Programming, *Ricardo Poli, William B. Langdon, & Nicholas McPhee* (freely available online at <http://www.gp-field-guide.org.uk>)

# Communication

- We will use Slack for all class communication. Invitations will be sent by email from KLMS, so watch out.
- Join the workspace, no excuse.

# Why AI and SE?

(wait, before that...)



**What is SE?**

Science: intellectual and practical activity  
encompassing the systemic study of the structure  
and behaviour of the physical and natural world

Mathematics: abstract science of number, quantity, and  
space, either as abstract concepts or as applied to  
other disciplines

Engineering: branch of science and technology  
concerned with the design, building, and use of  
engines, machines, and structures





### Software as science

- Precise computation and algorithm
- Study of truth
- Simplicity of theory



### Software as engineering

- Consideration of adequate cost
- Study of utility
- Scalability of theory



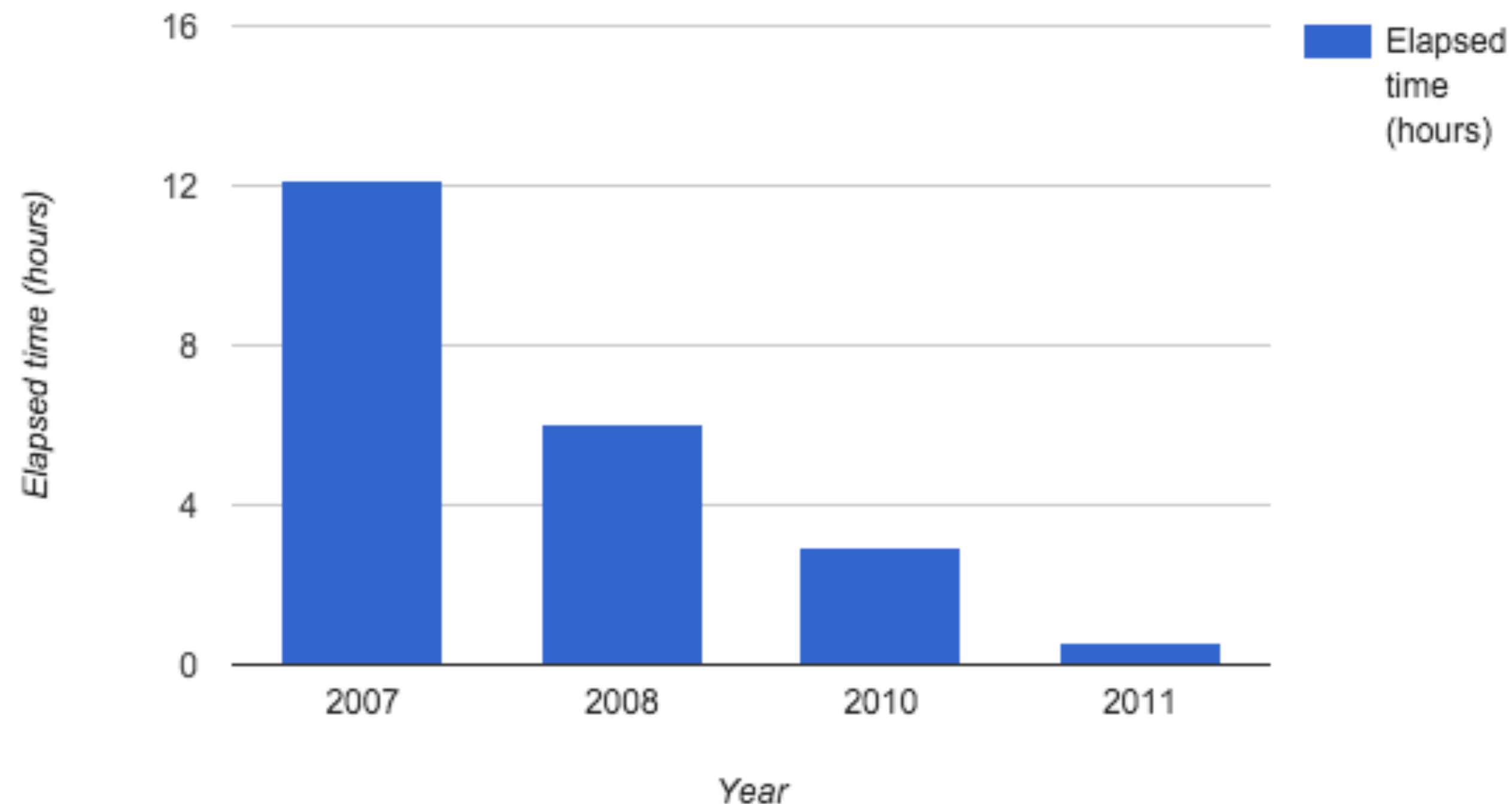
# Science of sorting

```
1 def mergesort(x):
2     """ Function to sort an array using merge sort algorithm """
3     if len(x) == 0 or len(x) == 1:
4         return x
5     else:
6         middle = len(x)/2
7         a = mergesort(x[:middle])
8         b = mergesort(x[middle:])
9         return merge(a,b)
```

#precise #theory # $O(n \log n)$  #provable

# Engineering the scalability of sorting

## from “History of massive-scale sorting experiments at Google”



- Tuning cluster configuration
- Changing the file system entirely + new encoding to reduce write amount
- Hardware tuning (I/O block size + SSD)
- Correctness check only came in 2010 :)

# Science

- Is the theory correct?

# Engineering

- Is the **implementation** correct?
- If it is not, how do we find out?
- How can we commit fewer mistakes while implementing?
- How can N members collaborate effectively and efficiently?
- How can we reduce the development cost?
- How can we maintain quality when team members change?



# Software Crisis

- As long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem. — *Edsger Dijkstra*, The Humble Programmer, Communications of the ACM, 1972





# NATO conference 1968

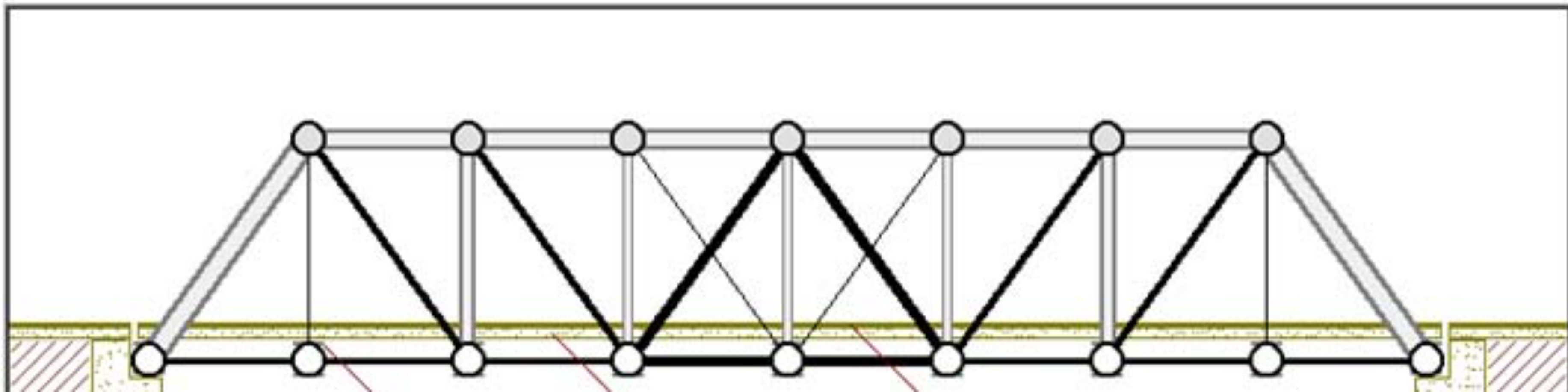
- ...concluded that software engineering should use the philosophies and paradigms of established engineering disciplines, to solve the problem of software crisis.



# In other “established engineering principles”...

- Things that can help you build the real product:
  - Theory
  - Modelling / Simulation / Optimization

**Let's say we want to build a steel bridge.**



# Theory

- When does a steel beam break?
  - Stress: force per unit area
  - Tensile strength: the maximum stress a material can resist

$$\text{stress} = \frac{\text{force}}{\text{area}}$$

## What does stress mean?

Stress allows us to get a fair comparison of the effects of a force on different samples of a material. A tensile force will stretch and, possibly, break the sample. However, the force needed to break a sample will vary - depending on the cross sectional area of the sample. If the cross sectional area is bigger, the breaking force will be bigger. However, the breaking *stress* will always be the same because the stress is the force per unit area.

In picture 4.2, sample A breaks with a force of **60 kN**. Sample B breaks with a force of **120 kN** because it has twice the area (you can think of it as being two pieces of sample A next to each other, with **each one** needing a force of 60 kN to break it). Although the force is bigger for sample B, the stress is the **same** for both samples – it is 60 kN cm<sup>2</sup>.

So breaking **stress** is a more useful measurement than breaking **force** because it is constant for a given material. It allows us to fairly compare the strengths of different materials.

### For example:

The force needed to break a piece of steel wire with a cross sectional area of  $2 \times 10^{-6} \text{ m}^2$  is 2400 N.

a) What is its breaking stress?

b) What force would be needed to break a steel bar with a cross section of  $5 \times 10^{-4} \text{ m}^2$ ?

a) The breaking stress = breaking force ÷ area  
=  $2400 \div 2 \times 10^{-6}$   
= 1,200,000,000 N m<sup>-2</sup>  
= 1.2 GPa.

b) To break the steel bar, the force needed = breaking stress x area  
=  $1.2 \times 10^9 \times 5 \times 10^{-4}$   
= 600,000 N

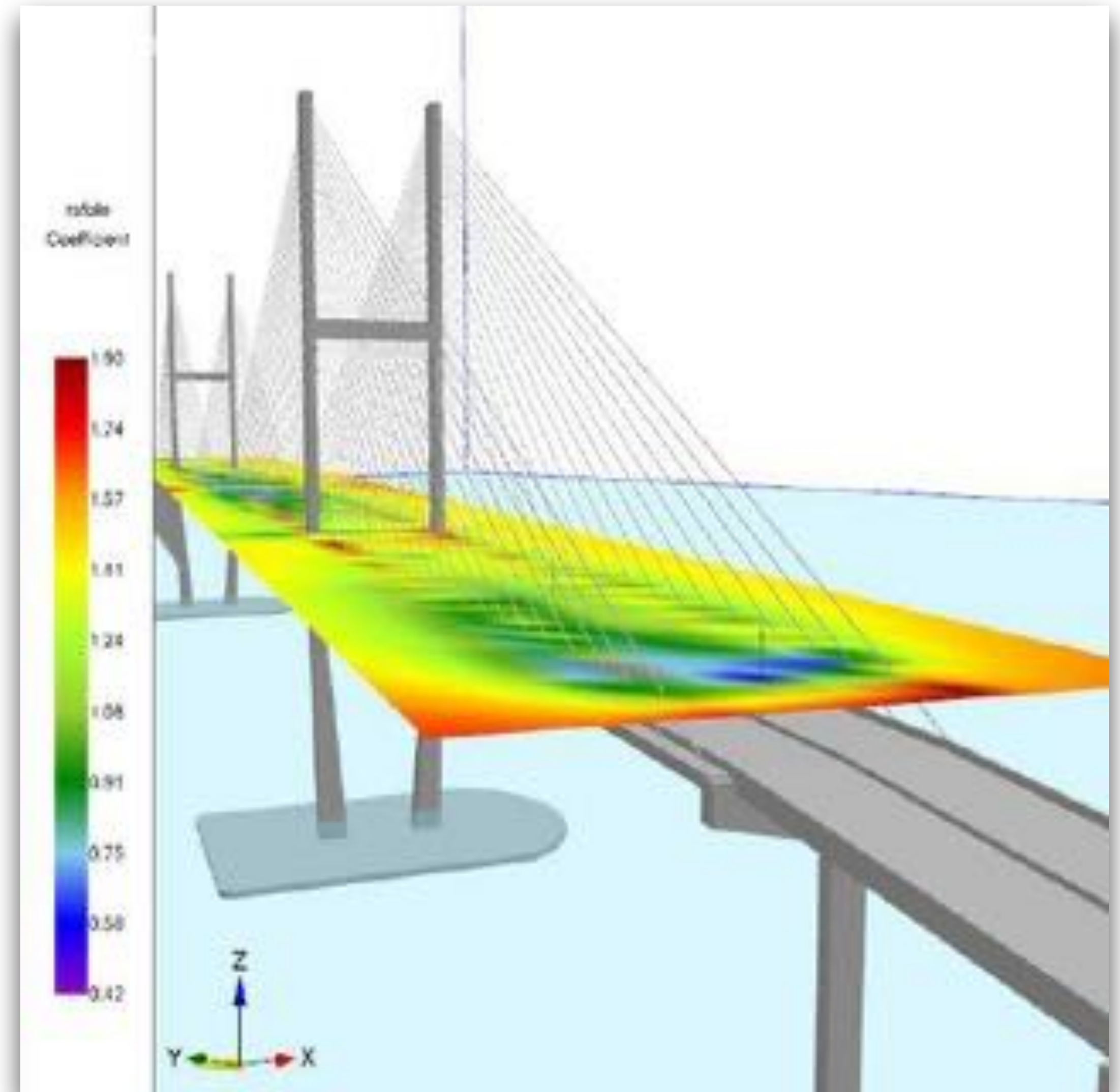
## Tensile strength

A tensile test is used to find out what happens when a material such as steel is stretched. A steel bar is placed in a device that pulls one end away from the other fixed end. The **tensile strength**



# Modelling/Simulation

- Given the physical laws as the foundation, it is possible to build simulations.

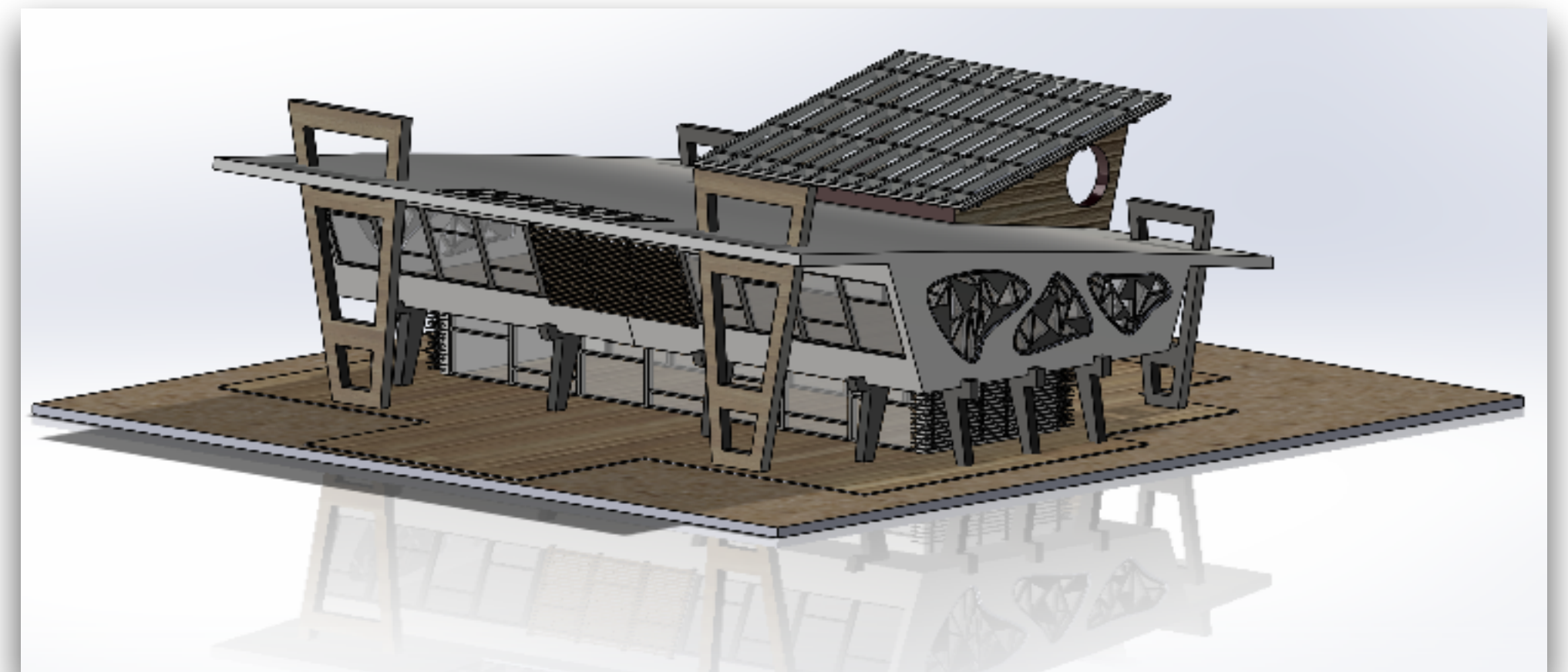


# Role of Computation

- Note that in all other engineering fields, computing now provides the means of modeling, simulation, and optimization
- Because, otherwise, these activities become too expensive and time consuming



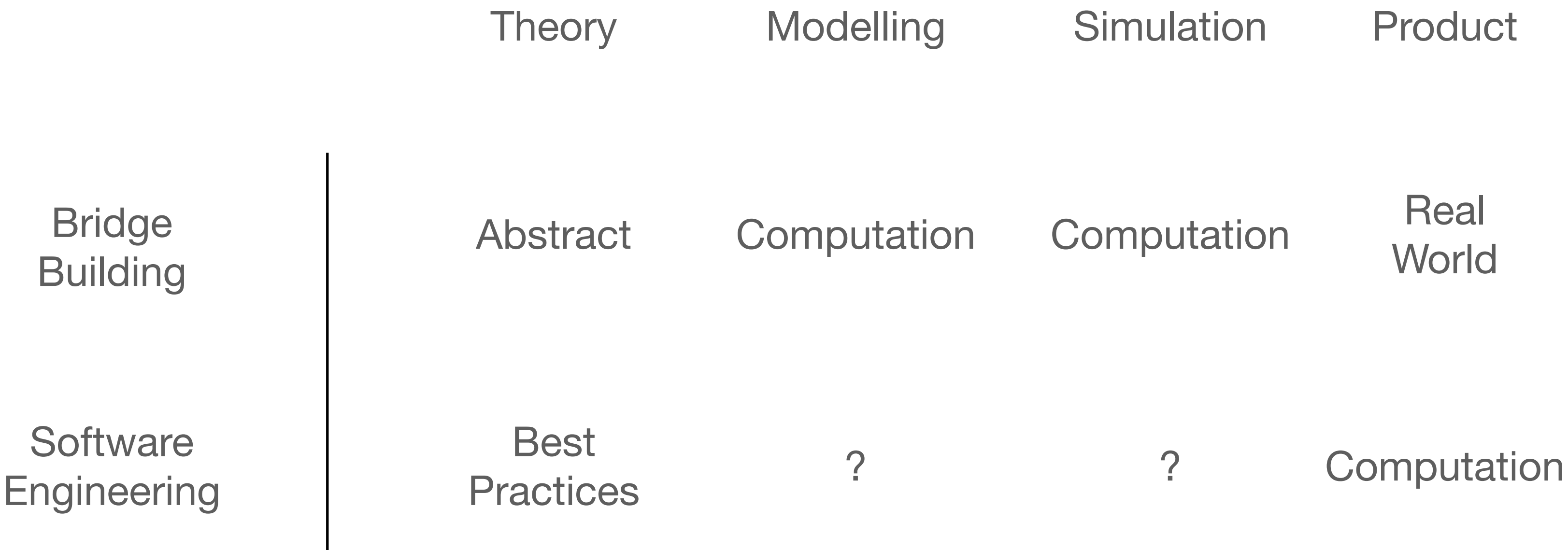
vs.





# But in our case, computation is the ingredients!

(Plus we lack unified theory of how software is supposed to work)



# Essential Properties of SW

as pointed out by Brooks Jr. in 1987

- Complexity
  - More complex than any other human constructs for their size (no two parts are identical, because we would factor them out)
  - Scaling up does not mean making the same thing larger
- Conformity
  - Physicists firmly believe that there is a unified theory of things
  - Most of complexity in SW is arbitrary - SW has to conform to countless many things (institutions, systems, users, regulations...)

# Essential Properties of SW

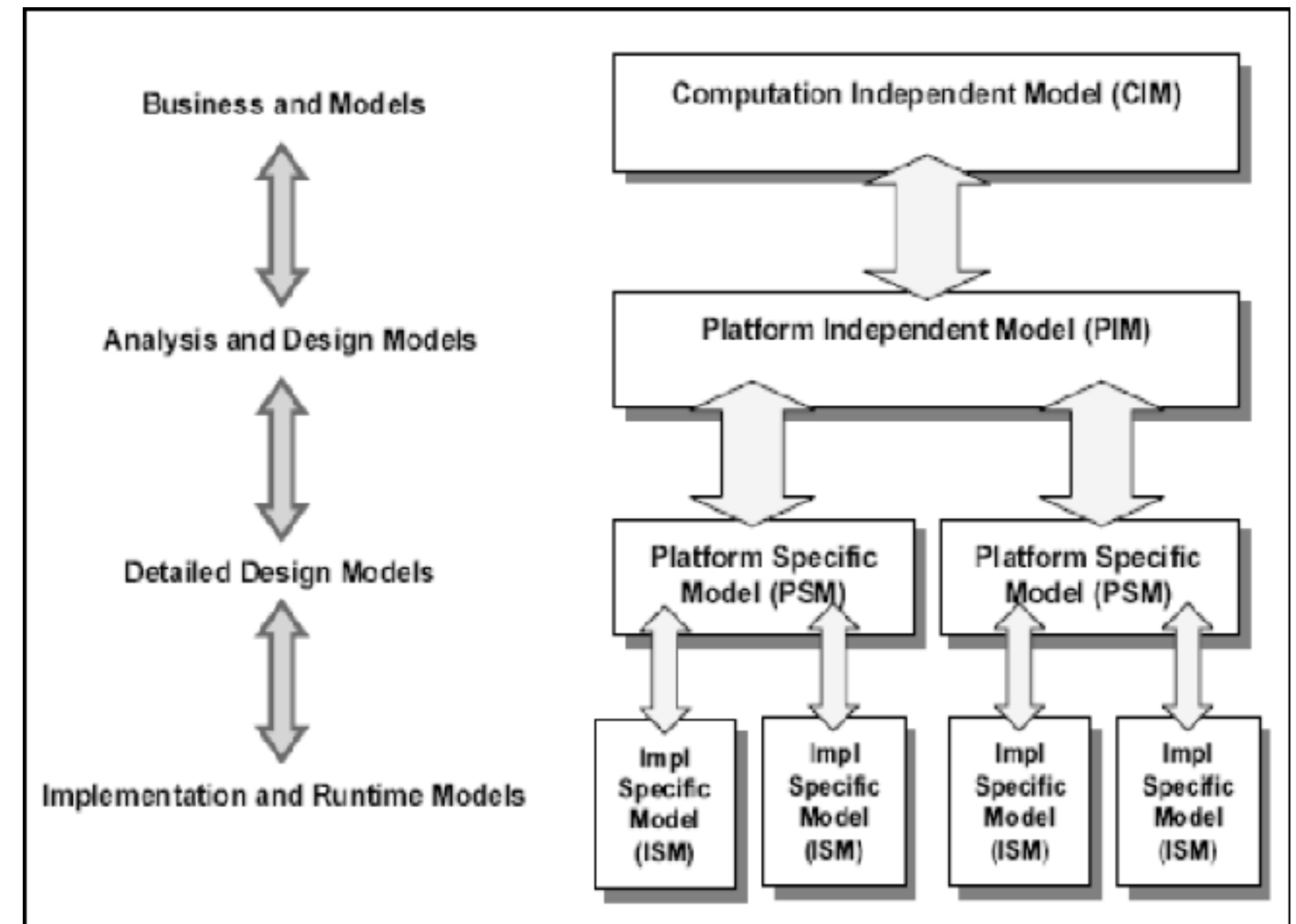
## as pointed out by Brooks Jr. in 1987

- Changeability
  - Compared to other engineering products, SW is more frequently pressured to change
  - If a software system is useful, people will try new edge cases at the borderline of the original domain
  - A successful software can outlive the underlying hardware, and therefore has to adapt
- Invisibility
  - SW cannot be easily embedded in space and therefore has no useful geometric representation
  - Without visual representation, communication and design becomes much more difficult



# Modelling / Simulation in SE is not new

- We have had numerous attempts to invent a thoroughly expressive modeling language for software (UML, anyone?)
- To start with abstract models, gradually refine and transform them into lower layers, until we have implementations —> the holy grail of model driven software engineering
- Why did this not work...? :)



# Search-Based Software Engineering

- A large movement(?) that seeks to apply various optimisation techniques to software engineering problems (**NOT** search engines or code search) - still relatively young (by academic standards), as it started in early 2000s
- Turns the notion of modeling/simulation upside down
  - Other fields: physical product —> model in computation —> optimization —> find solution —> build in physical world
  - SE: product in computation —> directly optimize —> product in computation
- We are perhaps the only engineering field where the ingredients for real product and its computation model is... the same (i.e., code).

# Code, in production AND in modeling

- We can apply all known computational optimization techniques directly to our **product**. Here are examples that will not work in other engineering field:
  - We can repeatedly execute our code (millions of times), to find a test input that will break the code (can you do crash test with million cars?)
  - We can automatically find patches for bugs, so that all test cases pass (can you automatically correct a flaw in a building by actually applying different candidate patches in real world?)

# SBSE is about application of optimization to SW

- A major part of CS454 is to understand this approach
  - Learn the underlying algorithms (mainly because no other course teaches them)
  - Look at individual application areas in SE related tasks
- Why do we want to look at SE tasks as **optimization** problems?
  - **Automate** SE tasks (either fully, or at least until human engineers can attend to the issue)
  - Gain **insights** into complicated problem domain that are either too large or too complicated for humans to understand
  - **Unbiased** decision support that is data-driven

# Another Important Angle

- CS454 is named AI-based Software Engineering, Initially, this meant that we use AI based techniques to solve SE problems (AI for SE).
- Increasingly, we need to solve SE problems in AI (SE for AI). Most importantly, there is an urgent need to **test machine learning modules in larger systems**.
- Very early stage, but we are going to look into this.

# What about that AI?

Artificial Intelligence Based Software Engineering Shin Yoo 2020 Fall

Good course! Course teaches you how to approach to optimization problems when you encounter them. Name AI can be misleading, it means traditional AI which solves problems by doing optimization.

- We are typically interested in problems that cannot be easily modeled mathematically. E.g.,
  - What is the expected speed-up in program execution if I add 10 to the value of this variable?
  - Will I get closer to executing this function if I change my input value by 2?
- When your problem defy any model, the only way of measuring what works and what does not is to actually try.
- Metaheuristics is an art of trial and error (we will see more about this next time)

# Our Stance

- We stand at the intersection of computational intelligence and software engineering.
- Pragmatic application has stimulated theoretical results in computational intelligence, and vice versa.
- Course: about 40% on optimisation techniques, 60% on applications on SE

