

# Welcome.

## Everyone:

- Pull the updates from the course GitHub repo:
  - `cd <46120-PiWE repo>`
  - `git pull upstream main` ← you might have “upstream2” instead

## Physical students:

- Sit WHEREVER you want. 😎
- Turn off laptop volume (mute). **←IMPORTANT!**
- Log into the Zoom meeting.
  - Microphone muted. Camera off.



# 46120: Scientific Programming for Wind Energy

Computational thinking and design of architecture

Ju Feng



# Agenda for today

- Pull new course material ✓
- Round robin
- Computational thinking and design of architecture
- Begin homework



# Homework in last week

- **Homework 0: Write wind turbine classes**

- **Homework 1:**

- Design a class for your final project that has at least:
  - one attribute that relates to the input data (those inside inputs if you are working on one of the pre-defined final projects).
  - one method that is useful for your final project.
- Write your code (the code for the class in inputs and script that create an object of the class and run the method in examples) and push to GitHub. Remember to use branch and PR.
- Add proper docstring and comments for all the code. If you can, also write the related tests.
- Think about how you may design multiple classes and how they should interact with each other for your final project.

- **Homework Optional:**

- Restructure Codecamp with OOP
- Design a base class `DynamicalSystem`



# Round robin

Share solutions with your peers and give feedback.





# Time to review and collaborate.

- 1 round of 25 minutes.
- 5 minutes: chaos.
- 20 minutes: present/discuss homework. Today's feedback focuses on Home 1:

1. How is the other team's code? Easy to understand?
2. Share with other groups how your class is designed and what is the thinking behind?
3. Do you have plans for designing more classes and how they should interact? Share and discuss with your peer groups.
4. Any other challenges you found? Maybe also for homework 0 and optional.

- Afterwards: plenum discussion.
  - Be ready with questions!

## PLENUM AT 09.30

```
1  Week 10
2  =====
3  BOR 0: Breeze-Tech, bug_hunters, Grustlers
4  BOR 1: Los-Programadores, FatalError2, codingteam
5  BOR 2: SOUL-FINDERS, NoOneKnows, Git_Happens
6  BOR 3: WindCoderss, Let_Me_Help_You, Lightning_McTeam
7  BOR 4: Crypto-Mania, Pythonagoras, BladePYRunner
8  BOR 5: WIndWise, Kala-Krasia, WindGPT
9  BOR 6: Mouxтин AE, WindWizards, LetsGoRepo
10 BOR 7: brunchyy, WindFusion, Push-Pray
11 BOR 8: StopFuckingSpiders, la_bombas_del_diablo
```

# Notes in plenum.



Start at ...

# Computational thinking





# Background

- What is the biggest challenge in software engineering?

**Managing complexity is the most important technical topic in software development.**

-- Steve McConnell in *Code Complete*

- Why do we compute?

**“We compute because we want to know and understand.”**

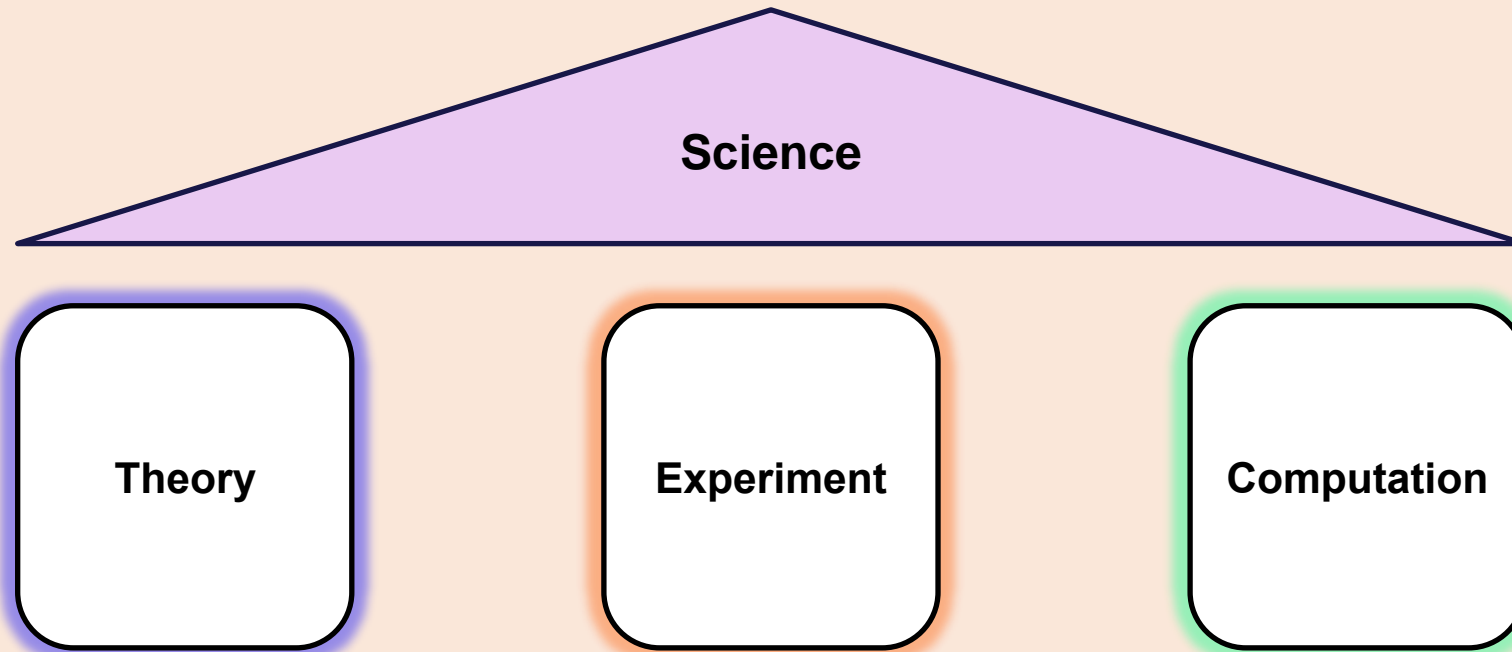
-- Daniel Reed in *Why we compute*

**“The purpose of computing is insight, not numbers.”**

-- Richard Hamming in *Numerical Methods for Scientists and Engineers*



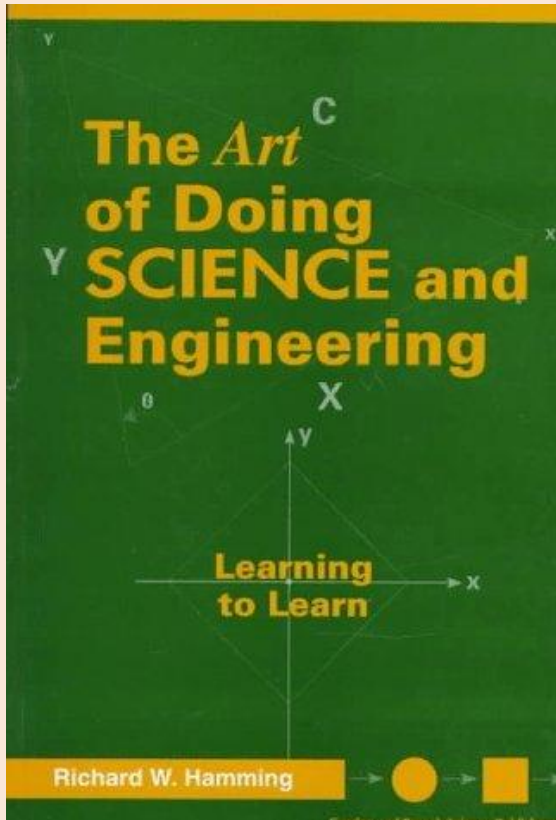
# The role of computation



Together with theory and experimentation, computational science now constitutes the **“third pillar” of scientific inquiry**, enabling researchers to build and test models of complex phenomena – such as multi-century climate shifts, multidimensional flight stresses on aircraft, and stellar explosions – that cannot be replicated in the laboratory, and to manage huge volumes of data rapidly and economically.

-- From *Computational Science: Ensuring America's Competitiveness*

# Science and Engineering



Hamming, R. W. *The art of doing science and engineering: Learning to learn*. Gordon and Breach Science Publishers, 1997

“I need to discuss science vs. engineering. Put glibly:

**In science if you know what you are doing you should not be doing it. In engineering if you do not know what you are doing you should not be doing it.**

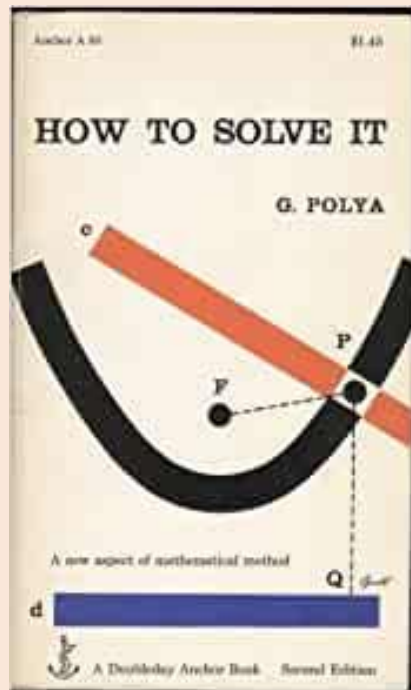
Of course, you seldom, if ever, see either pure state. All of engineering involves some creativity to cover the parts not known, and almost all of science includes some practical engineering to translate the abstractions into practice. Much of present science rests on engineering tools, and as time goes on, engineering seems to involve more and more of the science part. Many of the large scientific projects involve very serious engineering problems - the two fields are growing together!”

-- Richard Hamming in *The Art of Doing Science and Engineering*

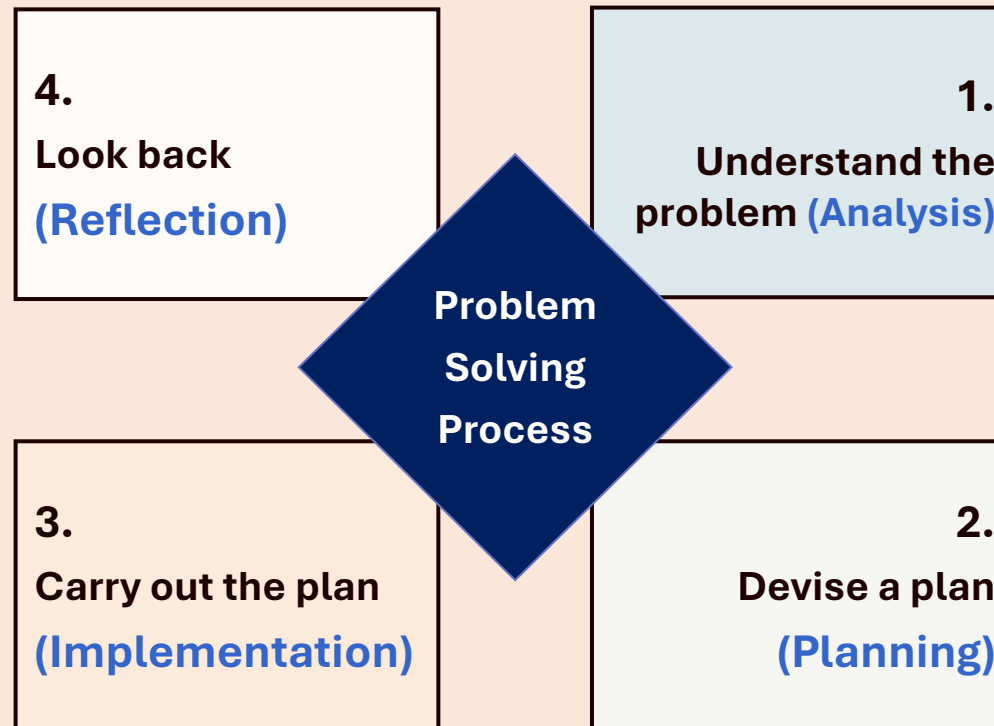


# Solving problems

- Engineering combines the fields of science and maths to **solve real world problems** that improve the world around us.



Polya, George. *How to solve it: A new aspect of mathematical method*. Princeton university press, 2004. (first in 1945)





# What is computational thinking?

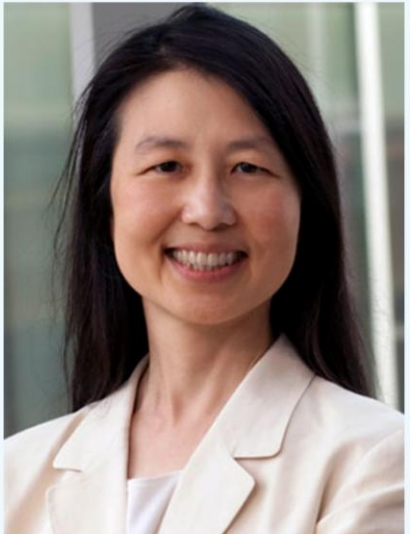
**Computational thinking is the thought processes involved in formulating a problem and expressing its solution(s) in such a way that a computer – human or machine – can effectively carry out.**

-- Jeannette M. Wing

- **Thought process** (thinking)
- **Problem formulation** (what to solve)
- **Solution expression** (how to solve)
- Computer implementation (solve it)

Source: <https://www.youtube.com/watch?v=V9Xy18YEK9M>





**Jeannette M. Wing**

Executive Vice President for Research

**The Fu Foundation School of Engineering and Applied Science**

Professor of Computer Science

Source:

<https://datascience.columbia.edu/people/jeannette-m-wing/>

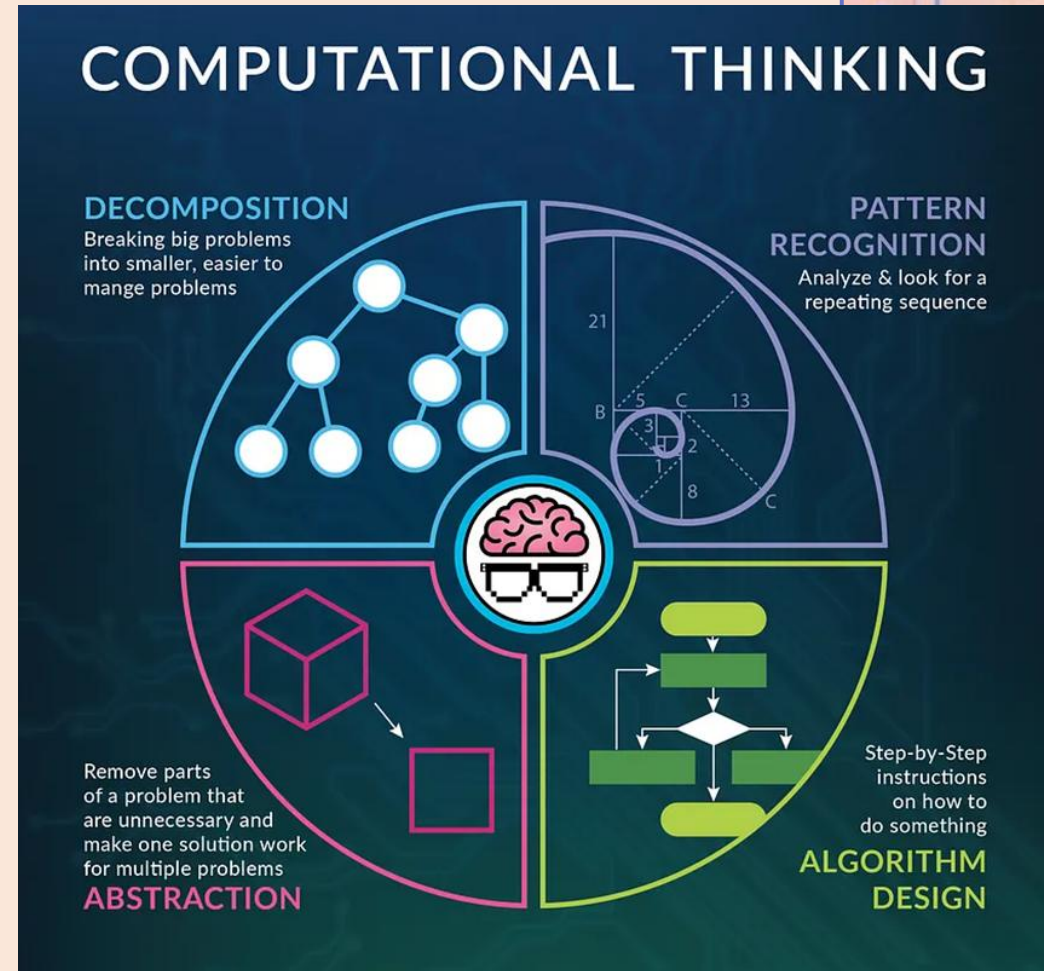


# Computational thinking

Four important concepts/techniques:

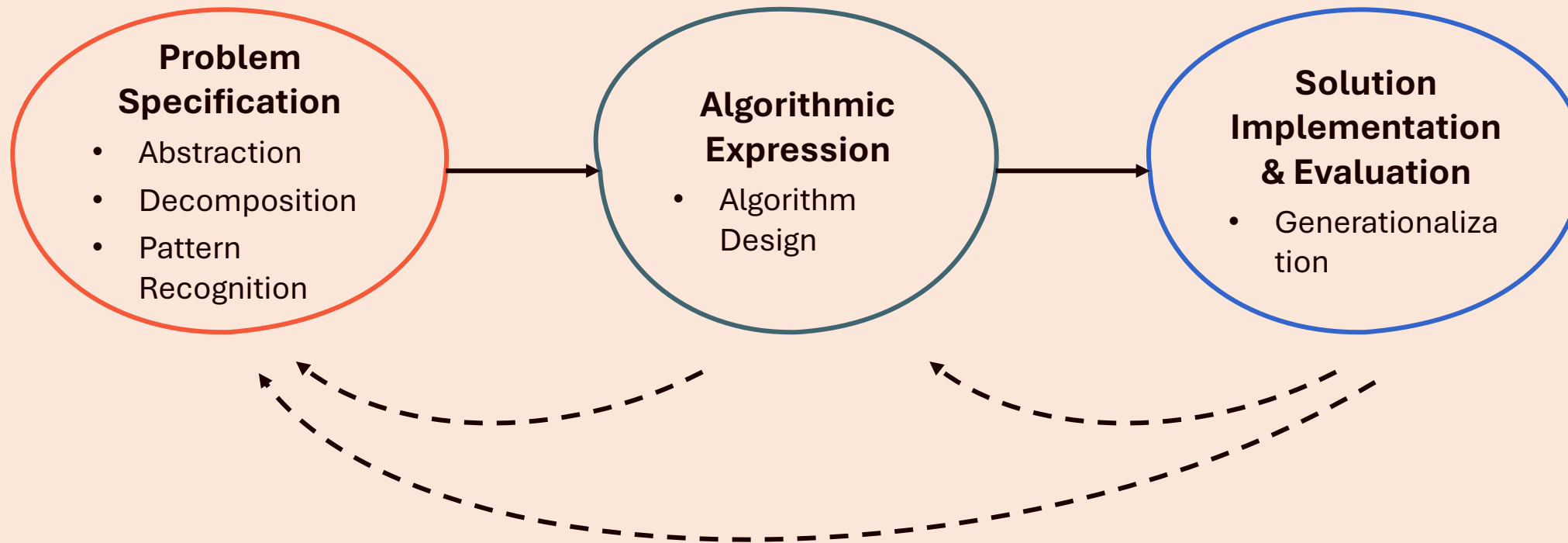
- **Abstraction:** Removing aspects of a problem that are not needed for its solution.
- **Decomposition:** Breaking a big problem down into smaller, more manageable sub-problems.
- **Pattern recognition:** Analyzing any kind of repeating elements or sequences in the problem.
- **Algorithm design:** Creating step-by-step instructions for solving the problem.
- **Generalization:** Extending a solution for a particular problem to other kinds of problems.

Source: Sethi, Ricky J. *Essential Computational Thinking: Computer Science from Scratch*. Cognella Academic Publishing, 2020.



Source: <https://medium.com/@sarahyam/how-can-we-use-computational-thinking-to-solve-business-problems-6e8e37913804hat>

# Computational thinking steps



Adapted from: Sethi, Ricky J. *Essential Computational Thinking: Computer Science from Scratch*. Cognella Academic Publishing, 2020.

# Computational thinking approach

## 1. Problem Specification

- **Computational Thinking Principles:** Problem Analysis and Abstraction
- **Computer Science Ideas:** Model Development using Decomposition and Pattern Recognition
- **Software Engineering Techniques:** Problem Requirements, Problem Specifications, UML diagrams, etc.

## 2. Algorithmic Expression

- **Computational Thinking Principles:** Data representation + Algorithm = Computational solution
- **Computer Science Ideas:** Systematically process information using modularity, flow control (including sequential, selection, and iteration), recursion, encapsulation, and parallel computing
- **Software Engineering Techniques:** Flowcharts, Pseudocode, Data Flow Diagrams, Class-responsibility-collaboration (CRC) cards for Class Diagrams, etc.

## 3. Solution Implementation & Evaluation

- **Computational Thinking Principles:** Systematic Testing and Generalization
- **Computer Science Ideas:** efficiency analysis, debugging, testing, etc.
- **Software Engineering Techniques:** Code Reviews, Refactoring, Test Suites, Quality Assurance (QA), etc.

From: Sethi, Ricky J. *Essential Computational Thinking: Computer Science from Scratch*. Cognella Academic Publishing, 2020.



# Design of architecture



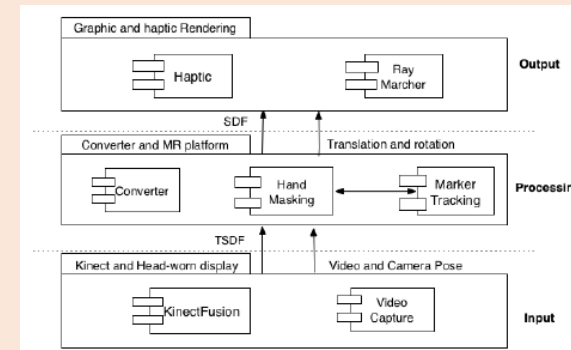
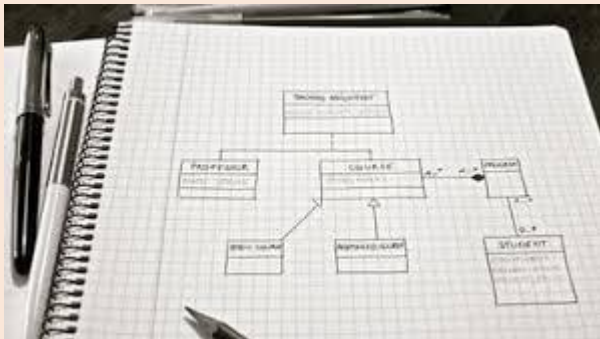
# What is software architecture?

All architecture is design, but not all design is architecture. Architecture represents the set of **significant design decisions** that shape the form and the function of a system, where significant is measured by cost of change.

-- Grady Booch in <https://handbookofsoftwarearchitecture.com>

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

-- Len Bass, et al. in *Software architecture in practice*





# Do we need software architecture?

- From waterfall to Agile, do we still need software architecture?  
Definitely, yes.

In preparing for battle, I have always found that plans are useless but planning is indispensable.

-- Dwight D. Eisenhower

Big up-front design is dumb! No up-front design is dumber!

-- Dave Thomas



# Why is software architecture important?

1. Allow reasoning about and managing changes.
2. Enhance communications among stakeholders.
3. Is a carrier of the earliest and hence most fundamental, hardest-to-change design decisions.
4. Provide the basis for evolutionary prototyping.
5. Focus attention on the assembly of components, rather than simply on their creation.
6. Channel the creativity of developers, reducing design and system complexity.

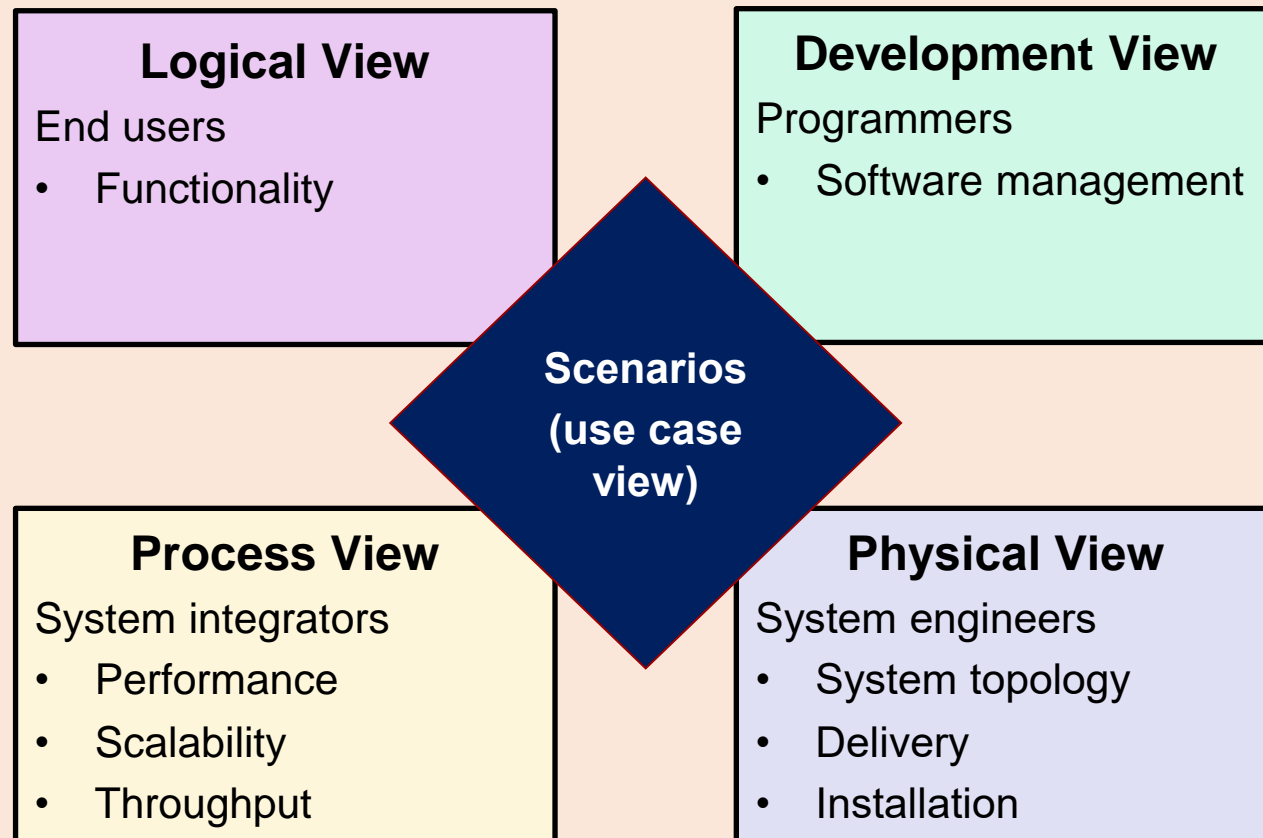
In general, a good software architecture can help increasing a software product's quality in terms of: Modularity, Understandability, Reusability, Extensibility, Modifiability, Testability, etc.

Selected from: Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice*. Addison-Wesley Professional.



# The 4 + 1 view model

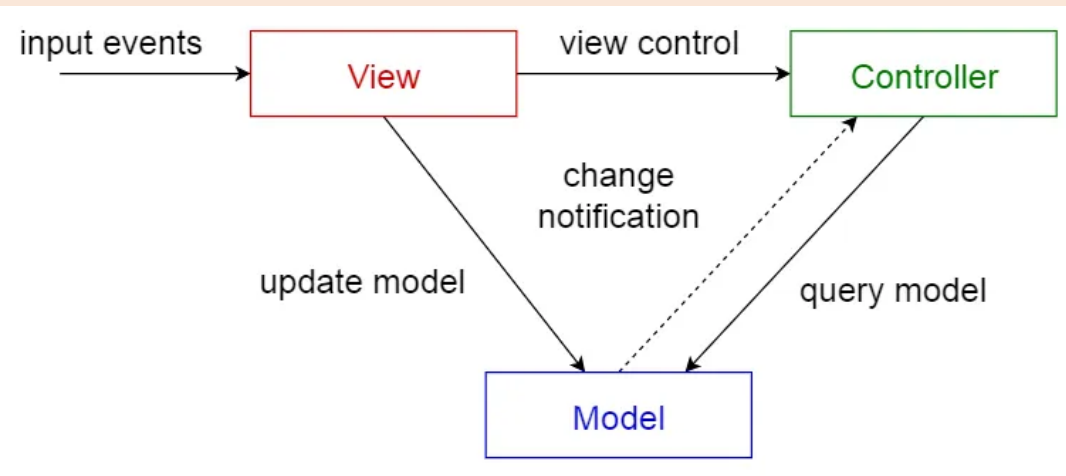
- Describe a software architecture using five concurrent views, each addresses a specific set of concerns of interest to different stakeholders in the system.



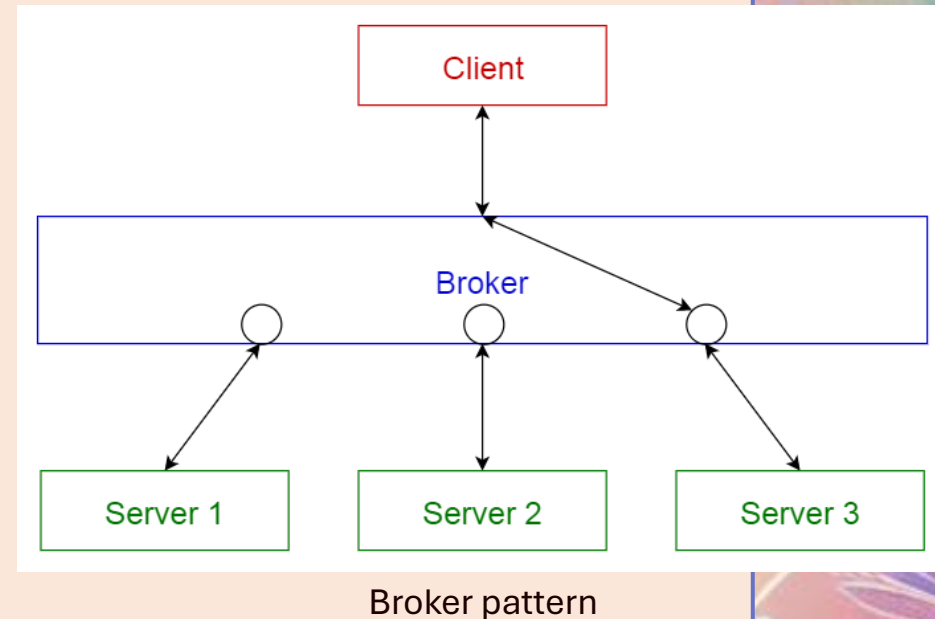
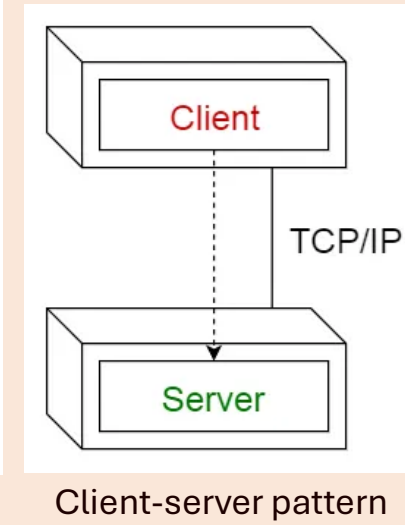
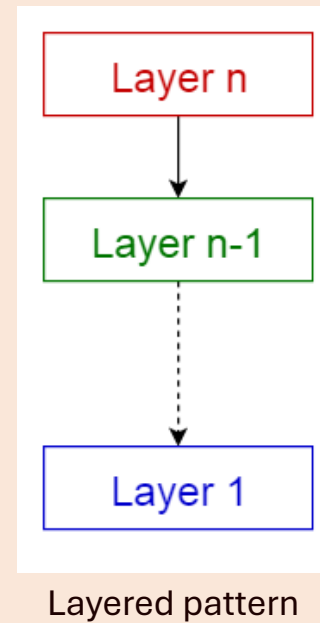
Adapted from: Kruchten, Philippe B.  
"The 4+1 view model of architecture."  
*IEEE software*. 12.6 (1995): 42-50.

# Software architectural patterns

- An architectural pattern is a general, reusable solution to a commonly occurring problem in software architecture within a given context. Architectural patterns are similar to software design pattern but have a broader scope.



Model-view-controller pattern

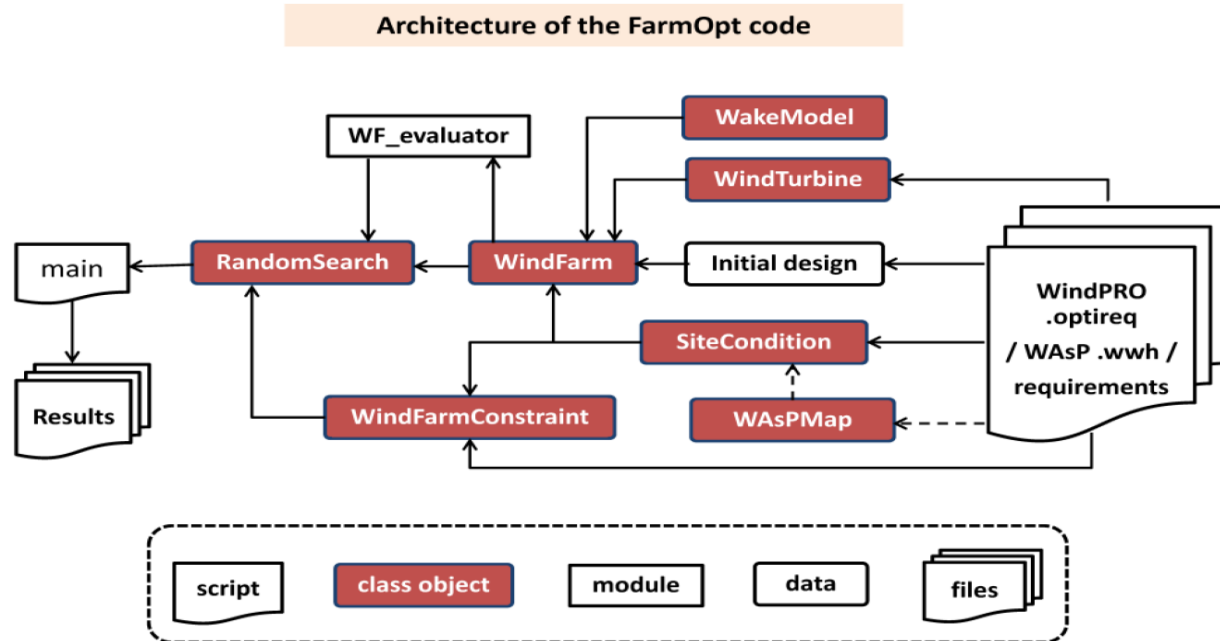


Source: <https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013>



# Draw your own architecture diagram

- Please draw an architecture diagram of your own project, using boxes and arrows (on paper or whiteboard, use PowerPoint draw.io, or Miro).



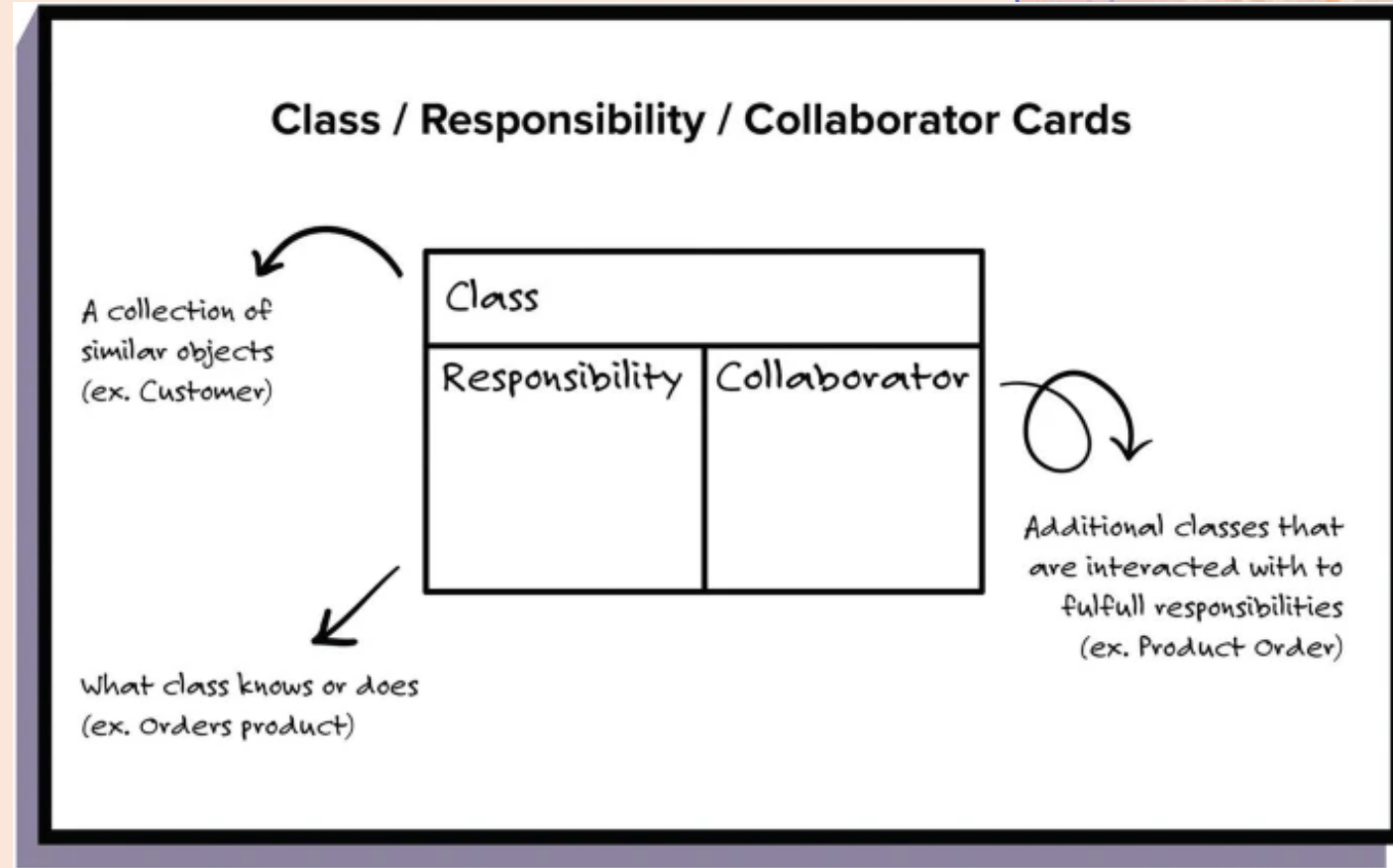
FarmOpt is written in Python 3.6, depending on open source Python modules, including [numpy](#), [scipy](#), [matplotlib](#), [zipfile](#), [xml](#) and [shapely](#).

A personal example for the FarmOpt code developed 9 years ago.



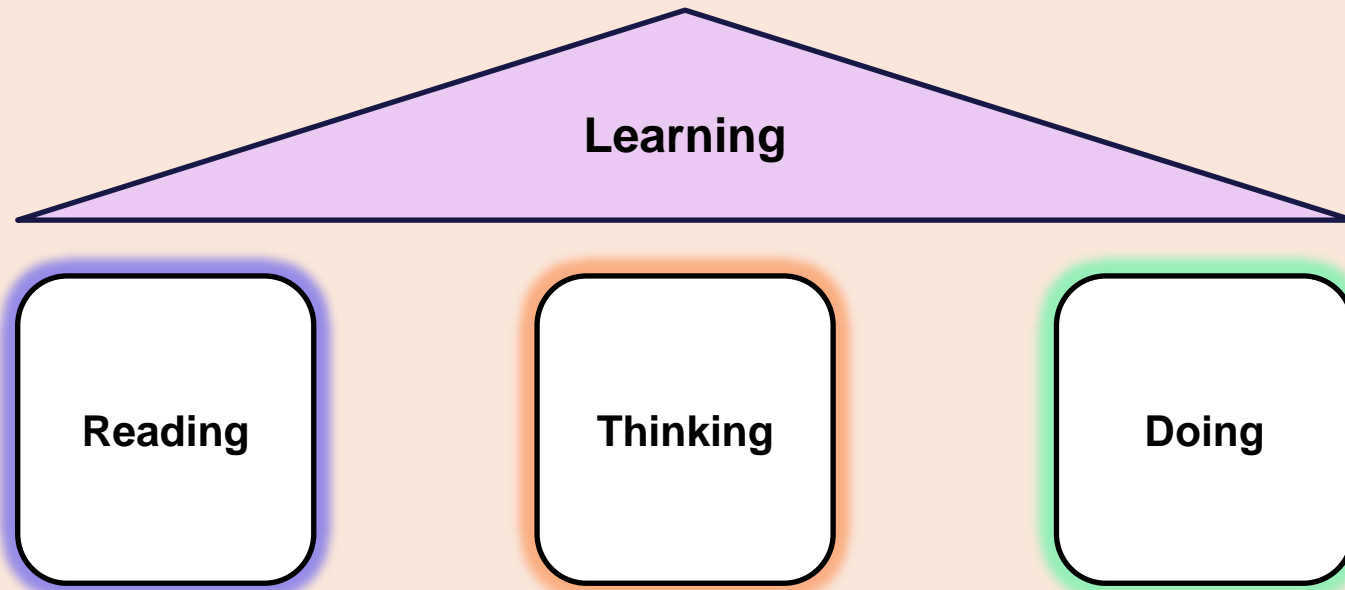
# Class, Responsibilities, Collaborators (CRC) Cards

- First, two or more team members write down the names of the most critical classes involved in the feature on index cards.
- Second, the cards are fleshed out with lists of the responsibilities of each class and the names of collaborators (i.e., other dependent classes).
- Third, team members perform a role-playing exercise and assume the role of one or more classes while playing out a plausible scenario of the design.



Source: Dalton, J. (2018). *Great Big Agile: An OS for Agile Leaders*. Apress.

# How to develop computational thinking?



学而不思则罔，  
思而不学则怠。

— 孔子

Learning without thought is  
labor lost;  
thought without learning is  
perilous.

— Confucius

# Questions?



*Remember, you're expected to work  
about 6 hours outside of class. Schedule  
accordingly.*

# Homework.

- **Homework 0:** Design the architecture of your final project and document it in a diagram. Note that this will be up to changes when you progress more with your final project, don't treat it as final and fixed.
- **Homework 1:** Keep working on your final projects!
- We'll open BORs in a minute. Enter room corresponding to your Team ID (on Learn).
- **To get help during class:** Post in Slack / #debugging, if you want a TA to enter your BOR or come find your group.

## Any questions?





# References

- McConnell, Steve. *Code complete*. Pearson Education, 2004.
- Daniel Reed, (2011) *Why We Compute* [link: <https://cacm.acm.org/blogcacm/why-we-compute/>]
- Hamming, R. W. *Numerical Methods For Scientists And Engineers*, 2<sup>nd</sup> Edition. Dover Publications, 1973
- United States. President's Information Technology Advisory Committee. *Computational Science: Ensuring America's Competitiveness*. National Coordination Office for Information Technology Research & Development, 2005.
- Hamming, R. W. *The art of doing science and engineering: Learning to learn*. Gordon and Breach Science Publishers, 1997.
- Polya, George. *How to solve it: A new aspect of mathematical method*. Princeton university press, 2004.
- Sethi, Ricky J. *Essential Computational Thinking: Computer Science from Scratch*. Cognella Academic Publishing, 2020.
- Grady Booch. *Handbook of Software Architecture*. [link: <https://handbookofsoftwarearchitecture.com/>]
- Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice*. Addison-Wesley Professional.
- Kruchten, Philippe B. "The 4+1 view model of architecture." *IEEE software*. 12.6 (1995): 42-50.
- Dalton, J. (2018). *Great Big Agile: An OS for Agile Leaders*. Apress.
- Confucius. *The Analects*, ca. 500 B.C.E [Available at: <https://classics.mit.edu/Confucius/analects.html>]

