

# CSE2115 Software Engineering Methods

## Introduction

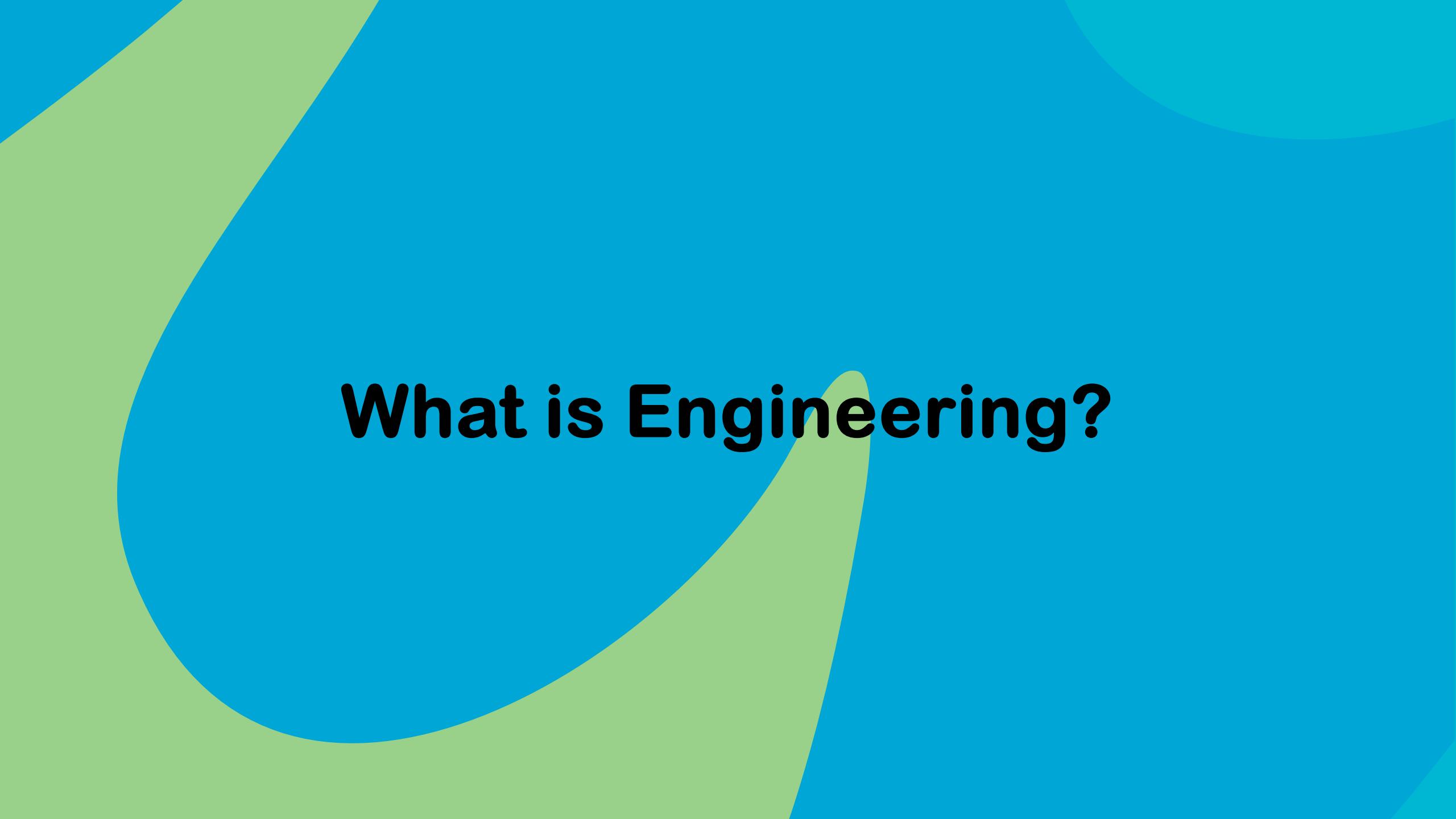
Taico Aerts & Arie van Deursen



**Why are you here?**

# **Why is Software Engineering Important?**

# **What is Software Engineering?**



# **What is Engineering?**

# Engineering

- **Engineering** is the practice of using natural science, mathematics, and the engineering design process to solve technical problems, increase efficiency and productivity, and improve systems.

# Engineering Design Process

- The **engineering design process**, also known as the **engineering method**, is a common series of steps that engineers use in creating functional products and processes. The process is highly iterative – parts of the process often need to be repeated many times before another can be entered – though the part(s) that get iterated and the number of such cycles in any given project may vary.
- It is a decision making process (often iterative) in which the basic sciences, mathematics, and engineering sciences are applied to convert resources optimally to meet a stated objective. Among the fundamental elements of the design process are the establishment of objectives and criteria, synthesis, analysis, construction, testing and evaluation.

# Engineering Design Process

- The **engineering design process**, also known as the **engineering method**, is a common series of steps that engineers use in (for) creating functional products and processes. The process is highly iterative – parts of the process often need to be repeated many times before another can be entered – though the part(s) that get iterated and the number of such cycles in any given project may vary.
- It is a decision making process (often iterative) in which the basic sciences, mathematics, and engineering sciences are applied to convert resources optimally to meet a stated objective. Among the fundamental elements of the design process are the establishment of objectives and criteria, synthesis, analysis, construction, testing and evaluation.

# The Engineering Method

- The engineering design process, also known as **the engineering method**, is a common series of steps that engineers use in (for) creating functional products and processes. The process is highly iterative – parts of the process often need to be repeated many times before another can be entered – though the part(s) that get iterated and the number of such cycles in any given project may vary.
- It is a decision making process (often iterative) in which the basic sciences, mathematics, and engineering sciences are applied to convert resources optimally to meet a stated objective. Among the fundamental elements of the design process are the establishment of objectives and criteria, synthesis, analysis, construction, testing and evaluation.

[https://en.wikipedia.org/wiki/Engineering\\_design\\_process](https://en.wikipedia.org/wiki/Engineering_design_process)

# The Engineering Method

- The engineering design process, also known as the engineering method, is a common series of steps that engineers use in (for) creating functional products and processes. **The process is highly iterative** – parts of the process often need to be repeated many times before another can be entered – though the part(s) that get iterated and the number of such cycles in any given project may vary.
- It is a decision making process (often iterative) in which the basic sciences, mathematics, and engineering sciences are applied to convert resources optimally to meet a stated objective. Among the fundamental elements of the design process are the establishment of objectives and criteria, synthesis, analysis, construction, testing and evaluation.

# The Engineering Method

- The engineering design process, also known as the engineering method, is a common series of steps that engineers use in (for) creating functional products and processes. The process is highly iterative – parts of the process often need to be repeated many times before another can be entered – though the part(s) that get iterated and the number of such cycles in any given project may vary.
- It is a decision making process (often iterative) in which the basic sciences, mathematics, and engineering sciences are applied to convert resources optimally to meet a stated objective. Among the fundamental elements of the design process are the establishment of objectives and criteria, synthesis, analysis, construction, testing and evaluation.

# The Engineering Method

- The engineering design process, also known as the engineering method, is a common series of steps that engineers use in (for) creating functional products and processes. The process is highly iterative – parts of the process often need to be repeated many times before another can be entered – though the part(s) that get iterated and the number of such cycles in any given project may vary.
- It is a decision making process (often iterative) in which the basic sciences, mathematics, and engineering sciences are applied to convert resources optimally to meet a stated objective. Among the fundamental elements of the design process are the establishment of objectives and criteria, synthesis, analysis, construction, testing and evaluation.

# CSE2115 Software Engineering Methods

Taico Aerts & Arie van Deursen

# Overview

- What is Software Engineering?
  - Brief History of Software Engineering
  - Course Organisation
- 
- Software Development Life Cycle
  - Requirements Engineering

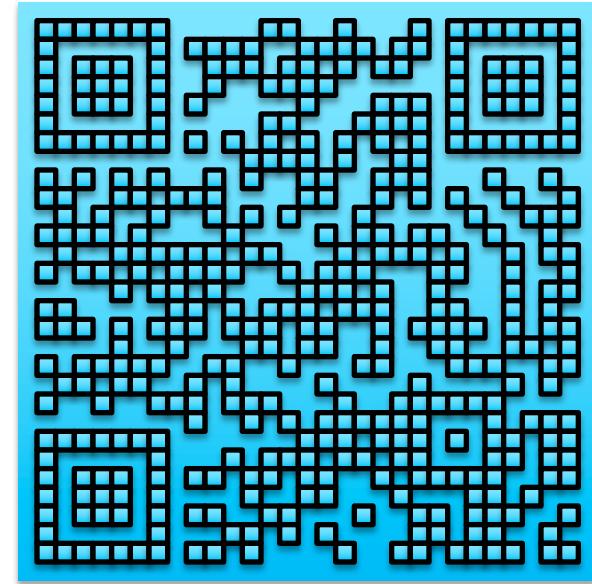
# **What is Software Engineering?**

# Join the Vevox session

Go to **vevox.app**

Enter the session ID: **159-004-429**

Or scan the QR code





##/##

Join at: [vevox.app](https://vevox.app) ID: 159-004-429

Open in a Poll

# What is Software Engineering?

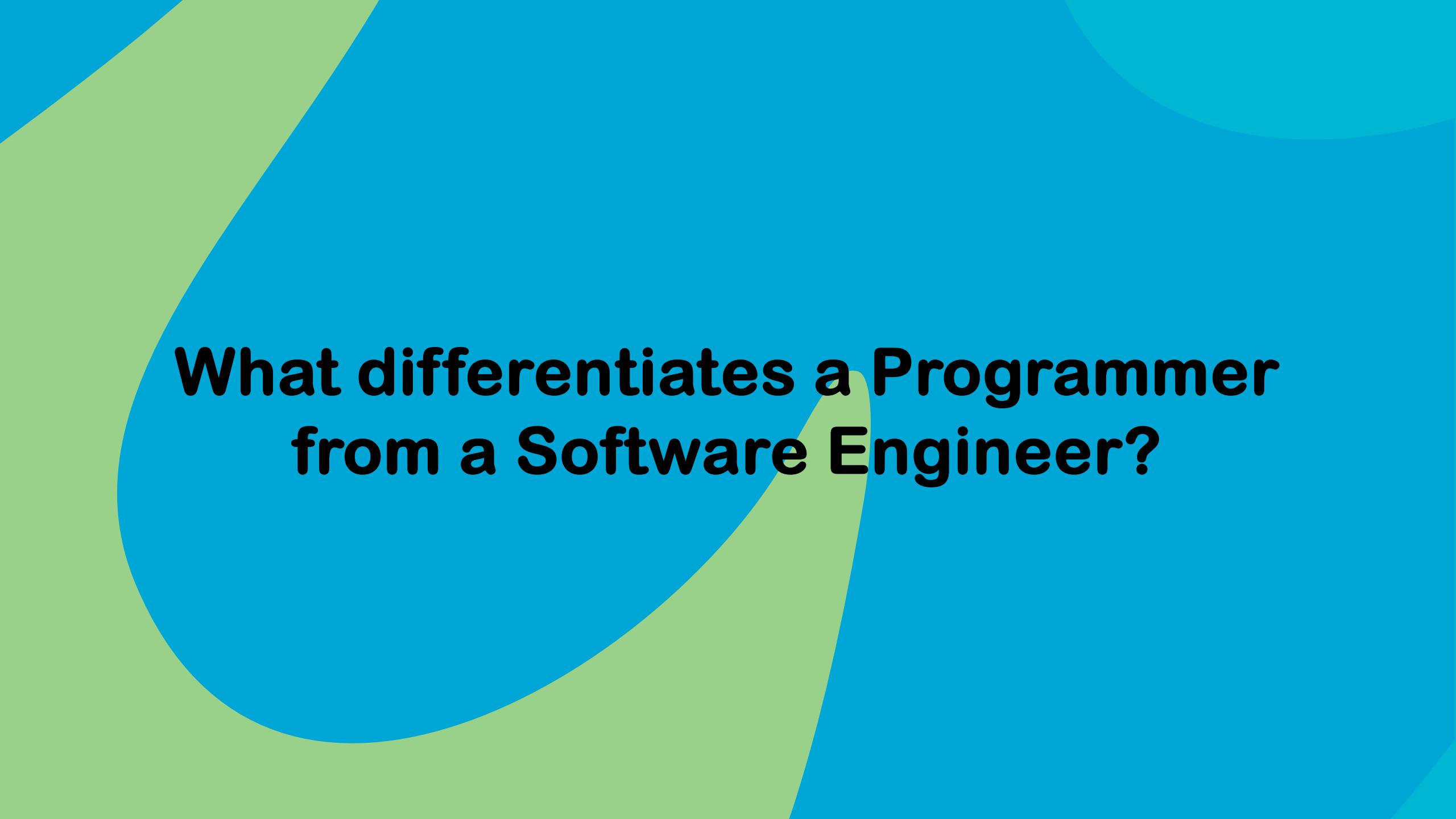


##/##

Join at: [vevox.app](https://vevox.app) ID: 159-004-429

Open in a Poll

# What is Software Engineering?



# **What differentiates a Programmer from a Software Engineer?**



##/##

Join at: [vevox.app](https://vevox.app) ID: 159-004-429

Open in a Poll

# What is the difference between a Programmer and a Software Engineer?



##/##

Join at: [vevox.app](https://vevox.app) ID: 159-004-429

Open in a Poll

# What is the difference between a Programmer and a Software Engineer?

# Software Engineering

- **Software engineering** is an engineering approach to software development.<sup>[1][2][3]</sup> A practitioner, called a **software engineer**, applies the engineering design process to develop software.
- The terms programmer and coder overlap software engineer, but they imply only the construction aspect of typical software engineer workload.<sup>[4]</sup>
- A software engineer applies a software development process,<sup>[1][5]</sup> which involves defining, implementing, testing, managing, and maintaining software systems and, creating and modifying the development process.

# Software Engineering

- Software engineering is an engineering approach to software development.<sup>[1][2][3]</sup> A practitioner, called a software engineer, applies the engineering design process to develop software.
- The terms programmer and coder overlap software engineer, but they imply only the construction aspect of typical software engineer workload.<sup>[4]</sup>
- A software engineer applies a software development process,<sup>[1][5]</sup> which involves defining, implementing, testing, managing, and maintaining software systems and, creating and modifying the development process.

# Software Engineering

- Software Engineer applies the **engineering design process (the engineering method)** to develop software
- Programmer/coder only does software construction
- SE involves defining, implementing, testing, managing and maintaining software systems

# Software Engineering

According to Google

Three critical differences between programming and SE



Time and Changes



Scale and Growth



Trade-offs and Cost

T. Winters et al. “Software Engineering at Google: Lessons Learned from Programming Over Time”, O’ Reilly (2020)

# Software Engineering

According to Google



Time and Change

- What is the lifespan of your code?
- We aim to write sustainable code that can be easily changed and adapted over time.

# Software Engineering

According to Google



Scale and Growth

- How many people are involved in the software?
  - Programming (act of individual creation)
  - Software Engineering (team effort)

# Software Engineering

According to Google



Trade-offs and Cost

- Optimal solution (silver bullet) that works in all contexts doesn't exist!
- We need to consider the different trade-offs.

# **Why is Software Engineering Important?**

# Why do we need Engineering Methods?

≡ Forbes

OCT 26, 2016 @ 06:26 AM 3,432 Ⓛ Sell

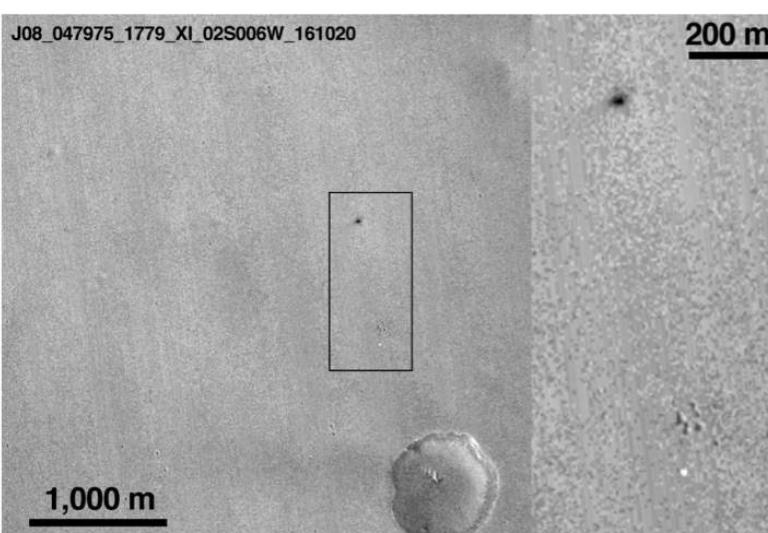
## Software Glitch May Have Crashed ExoMars Lander

[✉](#) [f](#) [t](#) [in](#) [g](#)

Brid-Aine Parnell, CONTRIBUTOR  
*I write about science, technology and the wonders of the universe* [FULL BIO](#) [▼](#)  
Opinions expressed by Forbes Contributors are their own.

The ESA's ill-fated Mars lander may have crashed and burned because a software glitch made it think it was on the surface when it was still kilometres away.

The European Space Agency, in partnership with Russia's Roscosmos, attempted its first Martian landing last week with the Schiaparelli probe, a prelude to a much bigger mission slated for 2020. But although the descent started out well, the probe eventually crashed on the surface and likely exploded on impact, according to images taken by NASA's Mars Reconnaissance Orbiter.



J08\_047975\_1779\_XI\_02S006W\_161020

200 m

1,000 m

This annotated Oct. 26, 2016 image shows a spot, indicated by a rectangular box, that likely shows the crash site of the European Space Agency's Schiaparelli lander. The image was taken by the HiRISE camera on NASA's Mars Reconnaissance Orbiter. [+]

19/10/2016

Schiaparelli lander crashed on Mars during the landing procedure (Mission cost ≈ 230 Million €)

The European Space Agency (ESA) said the crash was due to a software glitch in the inertial measurement unit.

ESA said the failure “has been clearly reproduced in computer simulations of the control system’s response to the erroneous information”

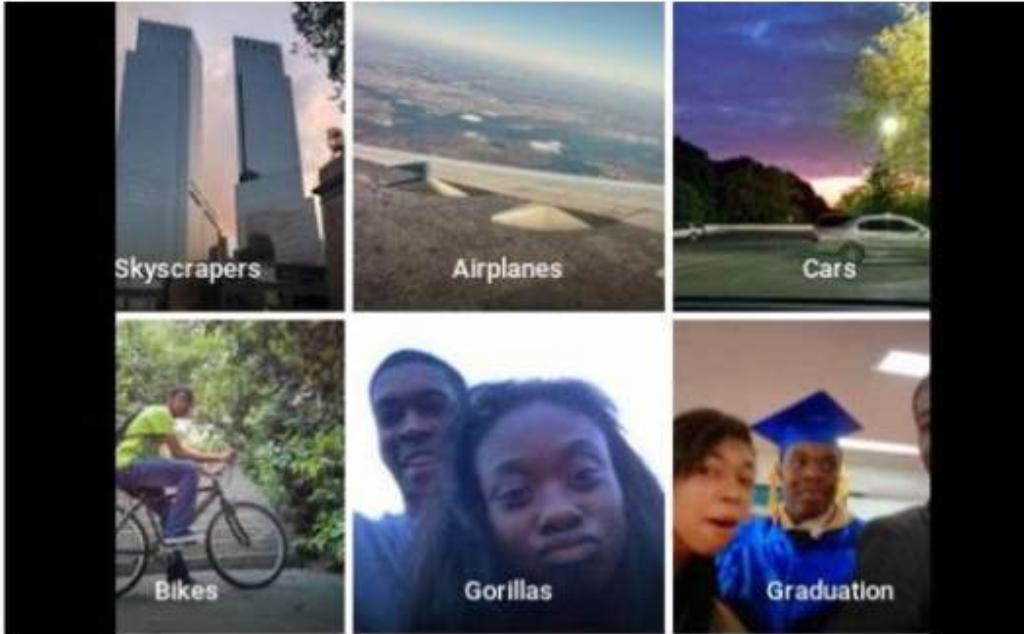
Finding out what went wrong is of tantamount importance, as Schiaparelli was meant to test the landing technology that will also be used on the second part of ESA's ExoMars mission in four years' time. The agency will then be sending a scientific station and rover to Mars to look for signs of ancient life in the planet's soil by drilling down two metres.

# What Could Possibly Go Wrong?

## Google apologises for Photos app's racist blunder

⌚ 1 July 2015

f Share



diri noir avec banan @jackyalcine · Jun 29  
Google Photos, y'all [REDACTED] My friend's not a gorilla.

813

394

TWITTER

Mr Alcine tweeted Google about the fact its app had misclassified his photo

### Top Stories

Fire ravages 200-year-old museum

⌚ 33 minutes ago

Rio museum co...

⌚ 7 hours ago

Myanmar jails reporter for secret papers

⌚ 3 hours ago

### Features



High-speed boat migrant tragedy

Software bug or wrong requirement engineering?

# (In)Famous Bugs

[https://en.wikipedia.org/wiki/List\\_of\\_software\\_bugs](https://en.wikipedia.org/wiki/List_of_software_bugs)

Not logged in [Talk](#) [Contributions](#) [Create account](#) [Log in](#)

Article [Talk](#) Read Edit View history Search Wikipedia 

## List of software bugs

From Wikipedia, the free encyclopedia

*This is a dynamic list and may never be able to satisfy particular standards for completeness. You can help by expanding it with reliably sourced entries.*

Many [software bugs](#) are merely annoying or inconvenient but some can have extremely serious consequences – either financially or as a threat to human well-being. The following is a [list of software bugs](#) with significant consequences:

**Contents** [hide]

- 1 Space
- 2 Medical
- 3 Tracking years
- 4 Electric power transmission
- 5 Administration
- 6 Telecommunications
- 7 Military
- 8 Media
- 9 Video gaming
- 10 Encryption
- 11 Transportation
- 12 Finance
- 13 See also
- 14 External links
- 15 References

### Space [edit]

- A booster went off course during launch, resulting in the destruction of [NASA Mariner 1](#). This was the result of the failure of a transcriber to notice an [overbar](#) in a written specification for the guidance program, resulting in the coding of an incorrect formula in its [FORTRAN](#) software. (July 22, 1962).<sup>[1]</sup> Note that the initial reporting of the cause of this bug was incorrect.<sup>[2]</sup>

# **History of Software Engineering**

# Origin of Software Engineering

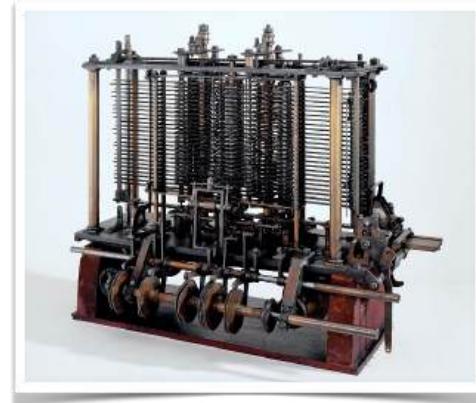
1880-1936	Software before the hardware
1937-1960	Hardware Era
1960-1968	Software Crisis
1968-1969	First Software Engineering Conference
1970s	Waterfall Model
1975	“The Mythical Man-Month” by Fred Brooks
...	

# Software Before the Hardware

In 1843, Ada Lovelace created the first programming language for the Analytical Engine to compute Bernoulli Numbers.



Ada Lovelace



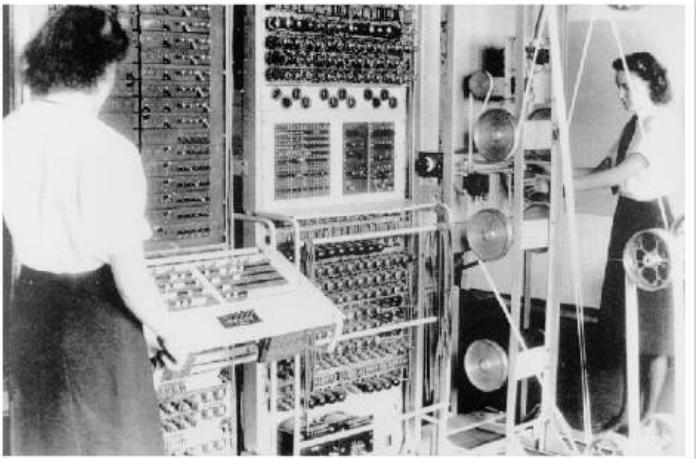
Analytical Engine

In 1936, Alan Turing invented the Turing machine, posing the basis to Computer Science and Theory of Computability.

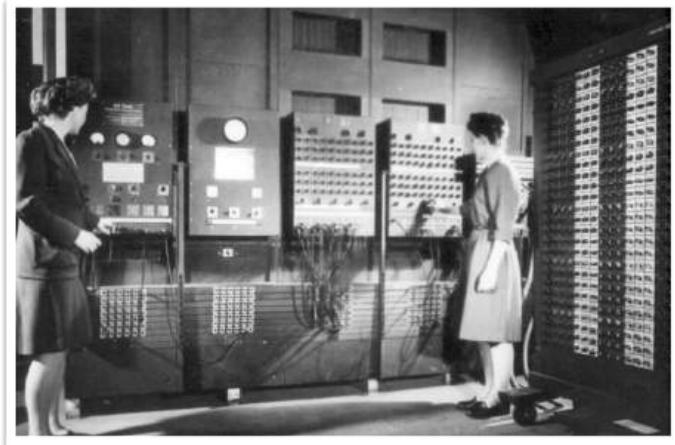


Alan Turing

# The Hardware Era



[https://en.wikipedia.org/wiki/Colossus\\_computer](https://en.wikipedia.org/wiki/Colossus_computer)



<https://en.wikipedia.org/wiki/ENIAC>

**Colossus** (1943-1945) used by the British Government for cryptanalysis of German Communication (WW-II)

Specifications: binary arithmetic, programmable via switches and plugs

**ENIAC** (Electronic Numerical Integrator and Computer) designed to solve numerical problems by the University of Pennsylvania in 1945-1946.

Specification: decimal operators; it included loops, conditions and subroutines.

# Software Crisis



“The major cause of the software crisis is that the machines have become several orders of magnitude more powerful!

To put it quite bluntly: 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 W. Dijkstra  
“The humble programmer”  
Communications of ACM, 1972

# The Apollo Project



# The Apollo Project

"When I first came up with the term [Software Engineering], no one had heard of it before, at least in our world. It was an ongoing joke for a long time.

It was a memorable day when one of the most respected hardware gurus explained to everyone in a meeting that he agreed with me that the process of building software should also be considered an engineering discipline, just like with hardware."

**Margaret Hamilton**



# Keynotes at ICSE 2018

<https://www.icse2018.org>

The screenshot shows the ICSE 2018 website's keynotes page. At the top, the ICSE logo is displayed next to the text "40<sup>TH</sup> INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING" and the dates "MAY 27 - JUNE 3 2018 GOTHENBURG, SWEDEN". Below the header, there are navigation links for "Attending", "Program", "Tracks", "Committees", "Search", and "Series". The main content area is titled "Keynotes" and contains the text "All plenary presentations are recorded and can be found [here](#) and [here](#)". Below this, there is a section titled "Keynotes Plenary sessions" featuring four speakers with their names and links to their presentations and videos:

Speaker	Description
Magnus Frodigh	<a href="#">Presentation / Video</a>
Fred Brooks, Jr.	<a href="#">Presentation / Video</a>
Margaret Hamilton	<a href="#">Presentation / Video</a>
Brian Randell	<a href="#">Paper / Video</a>

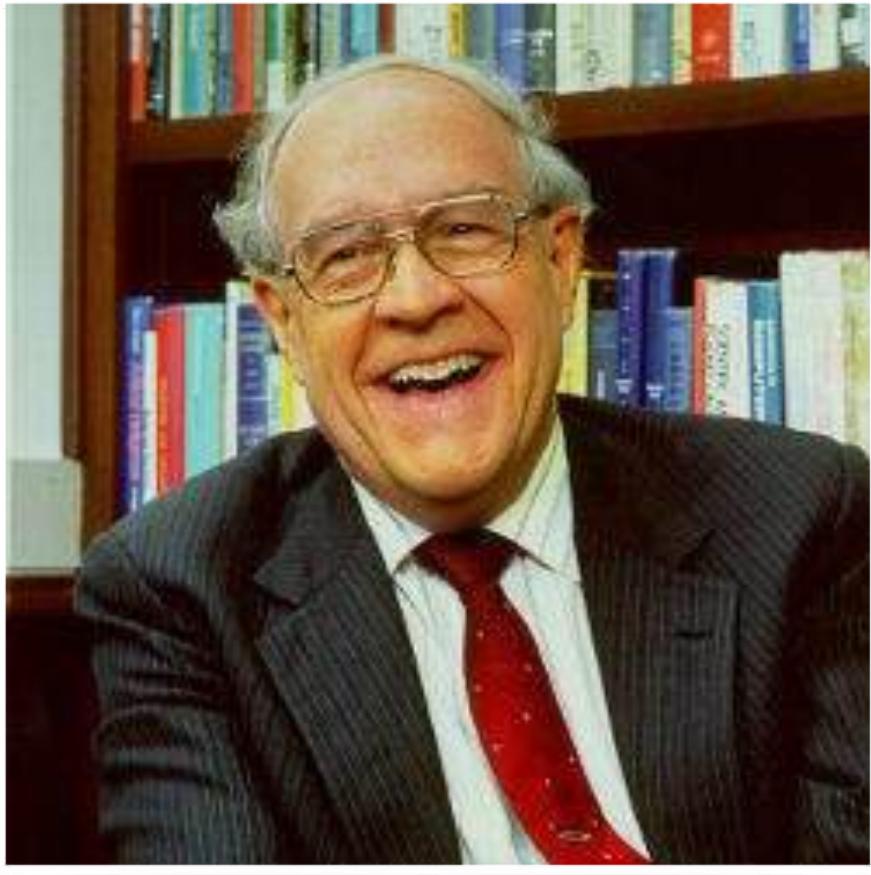
<https://www.youtube.com/watch?v=ZbVOF0Uk5IU>

Medal of Freedom Ceremony 2016



[https://www.youtube.com/watch?v=X1PNp\\_YggAA](https://www.youtube.com/watch?v=X1PNp_YggAA)

# The Mythical Man-Month



In 1975, Fred Brooks published a book entitled “The Mythical Man-Month: Essays on Software Engineering.”

The original book has been reprinted in 1982 and 1995 with corrections and extra chapters.

Brooks has received many awards, including the **National Medal of Technology** in 1985 and the **Turing Award** in 1999.

# No Silver Bullet

**Brooks' Law:** “adding manpower to a late software project makes it later”.  
**(Adding one person to a late project leads to further delays)**

**No Silver bullet:** “there is no single development, in either technology or management technique, which by itself promises even one order of magnitude [tenfold] improvement within a decade in productivity, in reliability, in simplicity.”

# **Course Organisation**

# Course Staff



**Taico Aerts**

*Lecturer & Software Engineer for the CSE Teaching Team*



**Arie van Deursen**

*Full Professor in the Software Engineering Research Group*



**Angelika Mentzelopoulou**

*Head TA*



**Ana Băltărețu**

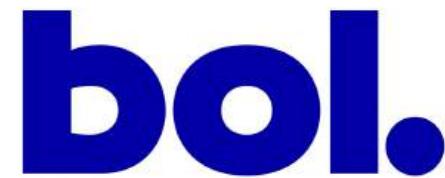
*Head TA*

# Additional Lecturers

- Various lecturers from Software Engineering and Cyber Security



- Guest Lectures



# Flipped Classroom

- Preparation for Lectures important!
- Lecture content partially guided by your questions (on WebLab)
- Exercises and discussion in lectures
- Task lists on Brightspace

# Lab

- Exercises for (Exam) Practice
- Putting theory into practice
- Preparation for Software Project



# Oral Presentations (OP)

- Subcourse as part of SEM
- Pass/Fail
- Once per week
  - 6 possible timeslots
  - Same time/place every week
- **Mandatory attendance**



# Oral Presentations – Enrolment

- **Enrol on Project Forum** (see Brightspace)
  - Bridging students are exempt
  - Passed previously? Request exemption.
- **Today before 23:59**
- Questions/Issues? Contact OP coordinator:



**Florian Gerritsen**

*Lecturer at Technology, Policy and Management (TPM)*

[F.W.Gerritsen@tudelft.nl](mailto:F.W.Gerritsen@tudelft.nl)

# Why Oral Presentations?

- Communication skills are essential for engineers
- Helps with:
  - Selling products/designs
  - Getting message across
  - Communicating effectively with stakeholders

# OP Final Presentation

- Budget for improving system
- Give your ideas for improvements
- Apply your SEM knowledge
  - Requirements, Architecture changes, Quality, etc.
- Choose between
  - Queue, <https://queue.tudelft.nl>
  - Submit (+ Boxer), <https://submit.tudelft.nl>
  - TAM (TA Management), <https://ta.ewi.tudelft.nl>

# Course Schedule

Week	Topic	Thursday 10:45-12:45	Friday 08:45-10:45	Oral Presentations
1.1	Defining the Project	Introduction & Requirements Engineering	Requirements Engineering & UML	-
1.2	Software Design	Software Architecture	Design Patterns	Introduction
1.3	Domain Driven Design	Open Source	Domain Driven Design	Pitch Presentations
1.4	Teamwork	Professional Agility at Scale	[Guest] SE at Bol.com	Intro & Ending
1.5	Software Quality	Software Metrics and Technical Debt	Sustainable Software	Trial Presentations
1.6	Software at Scale	Software at Scale	Software Security	Trial Presentations
1.7	The Future of SE	AI for SE	[Guest] AI in the IDE by Jetbrains	Final Presentations
1.8	-	-	Recap and Q&A	-
1.9	-	-	-	-

# Exam

- Monday 4 November, 13:30-16:30
- Digital Exam on WebLab
  - Multiple-choice Questions
  - Open Questions
  - UML Drawings in PlantUML

# Final Grade

- OP Passed → Exam Grade
- OP Failed → Insufficient

# Changes from Previous Year(s)

- Mandatory project replaced with Lab
- Switched to flipped classroom
- Content changes
  - >> Requirements Engineering
  - << Microservices and Domain Driven Design (project)
  - + Teamwork
  - + Sustainable Software Engineering
  - + Software Security
  - – Mutation Testing
  - – Regression and Performance Testing

# Support

- Content
  - Pre-lecture: WebLab
  - Answers-EWI
  - TAs in Lab
  - Lecturer before/after lectures
- Oral Presentations
  - Normally: your OP lecturer (see Brightspace)
  - General: [F.W.Gerritsen@tudelft.nl](mailto:F.W.Gerritsen@tudelft.nl)
- Other
  - [sem-cs-ewi@tudelft.nl](mailto:sem-cs-ewi@tudelft.nl)

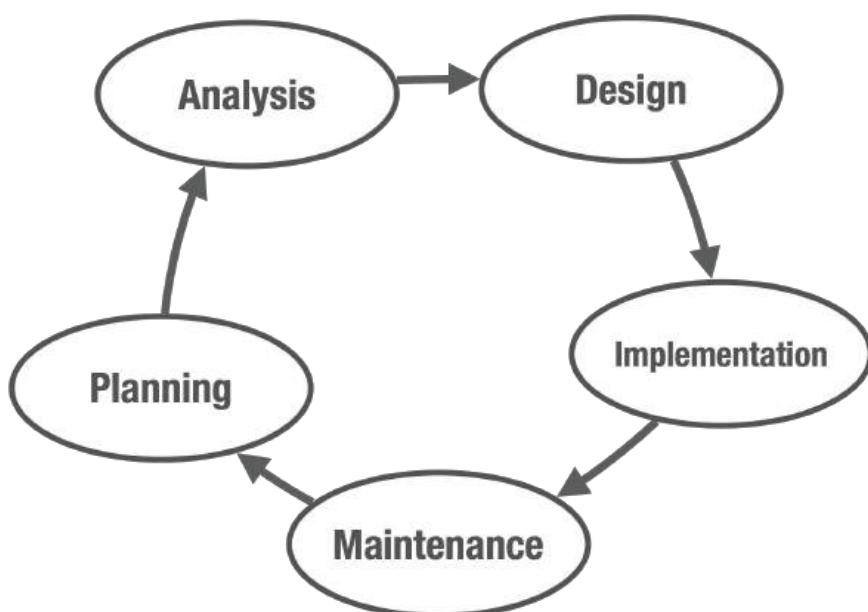
# CSE2115 Software Engineering Methods

**Requirements Engineering**

Taico Aerts & Arie van Deursen

# Software Development Lifecycle (SDLC)

- Process in SE for planning, designing, developing, testing and deploying software projects

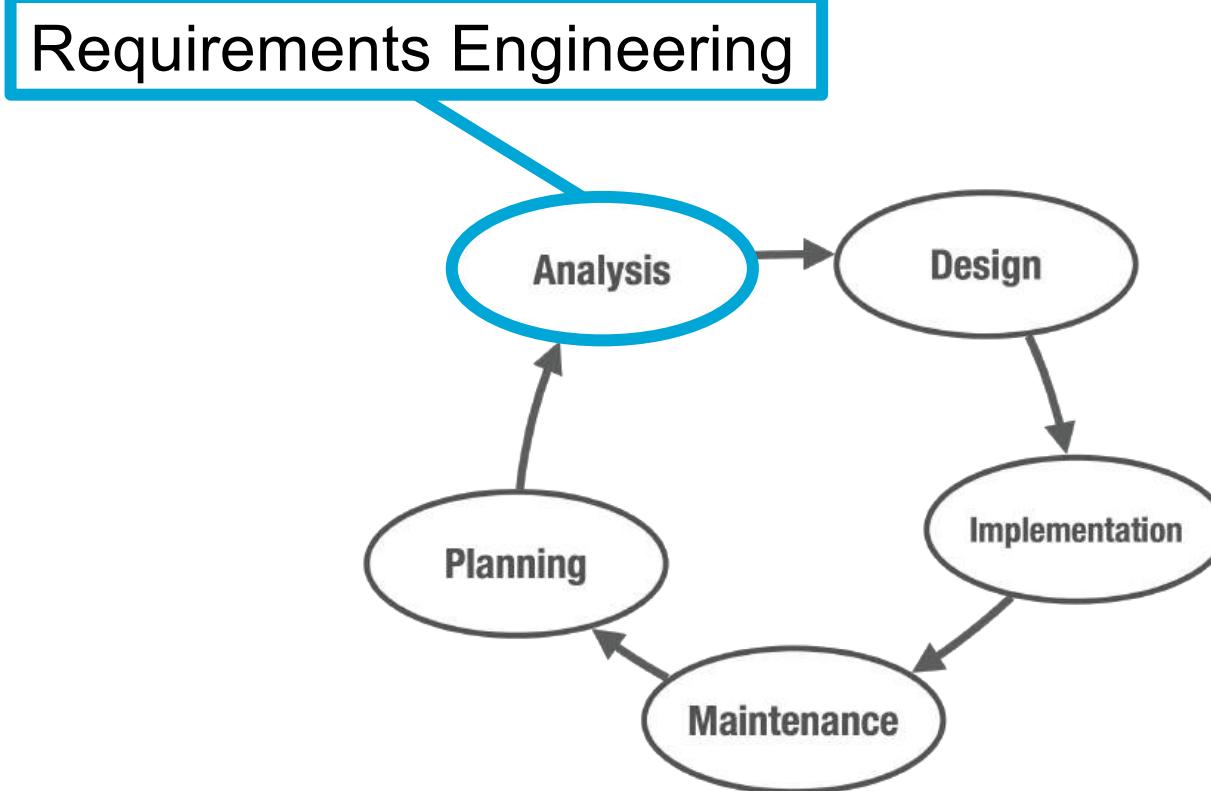


- There are different SDLC models
- Usually same steps, but different emphasis, frequency or order
  - Waterfall
  - Spiral
  - V-model
  - Agile
  - Scrum

# What have you already seen?

- Some parts of the SLDC were covered in OOPP and SQT
  - We will go more in depth for most parts
  - Skipping testing
- OOPP: Given these requirements, prioritize them and break up into issues
- But, how do you get requirements?

# Software Development Lifecycle



# **Requirements Engineering**

# Requirements

- A requirement is a statement about one feature (or constraint) of the system
- Express with:

# Requirements

- A requirement is a statement about one feature (or constraint) of the system
- Express with:
  - Natural language

Excerpt of the Apollo 11  
requirements (Rev. #1)

15 June 1962

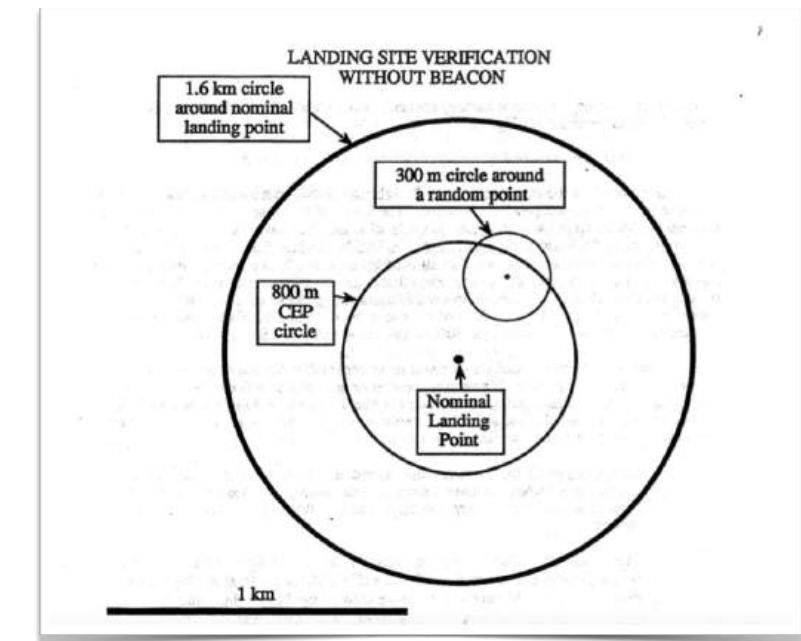


- Resolution within the landing zone had to be sufficient to allow interpretation of "50-foot<sup>1</sup> (15.3-m) size objects in areas of prime interest and 150-foot (46-m) size object capability over the entire zone. Areas of prime interest [were] to be selected on basis of current lunar knowledge with revisions reflecting acquired data and development of Apollo operational requirements" (NASA, 1962b).
- Further resolution requirements for each landing area were to allow, "identification of 4-foot (1.2-m) size objects – cracks, protuberances, ridges, craters, etc. Optical resolution of 1-foot (31-cm) per line pair<sup>2</sup> is required to assure this photo interpretation capability" (NASA, 1962b). Landing areas were defined as a portion of a landing zone that was of specific interest as a location for Apollo landings.
- Area covered by this resolution was to be roughly circular, with a 30,000-foot (9.2-km) diameter.
- Photographs would be indexed with existing lunar maps/photography.
- Data returns had to be of adequate resolution to determine surface roughness consisting of 1) discrete protuberances or objects (in 4-foot [1.2-m] size range) on a relatively smooth surface, 2) a pattern of small scale (10-foot [3.1-m] crest to crest) slopes or ridges, 3) cracks or depressions, and 4) a pattern of jagged protuberances. Further, the document called for a knowledge of the extent of roughness, particularly to answer the questions: 1) does rough terrain exist in a uniform distribution over large surface area, 2) are there patches of rough area in otherwise smooth areas and, 3) do patches of rough terrain constitute more than 50% of a large area?<sup>3</sup>
- Slope information needed was 1) the percentage of a landing area exceeding a slope of 20°, 2) whether the pattern of slopes within a landing area were parallel or random-crested, and 3) for areas with parallel crests, a measurement of the

# Requirements

- A requirement is a statement about one feature (or constraint) of the system
- Express with:
  - Natural language
  - Model-based

Excerpt of the Apollo 11  
requirements (Rev. #1)  
15 June 1962



# Requirements

- A requirement is a statement about one feature (or constraint) of the system
- Express with:
  - Natural language
  - Model-based
    - Graphs (e.g. UML diagrams)
    - Formulas
    - Pseudo-code

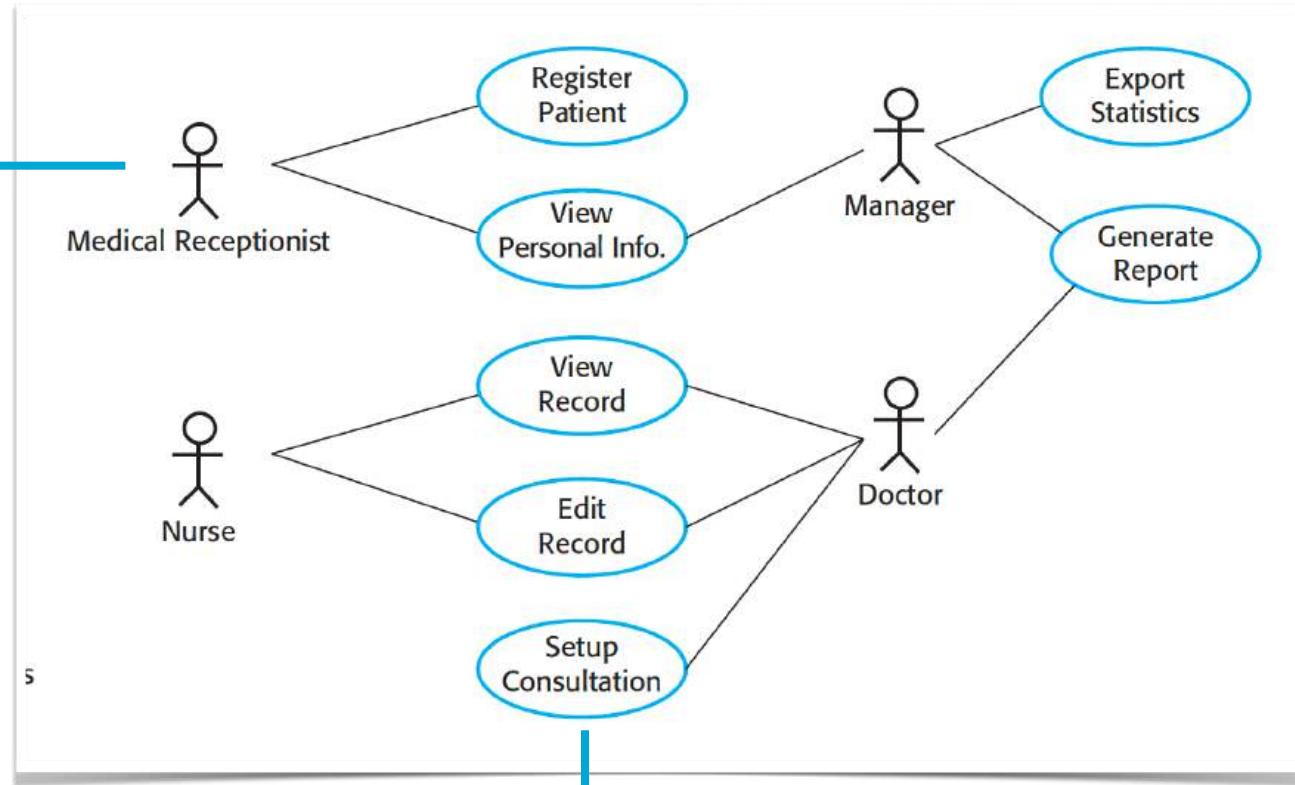
# Requirements

- A requirement is a statement about one feature (or constraint) of the system
- Express with:
  - Natural language
  - Model-based
    - Graphs (e.g. UML diagrams)
    - Formulas
    - Pseudo-code

Used to describe scenarios  
and how users would  
interact with the systems

# Use Case Diagrams

**Actors**  
(who interacts  
with the system)

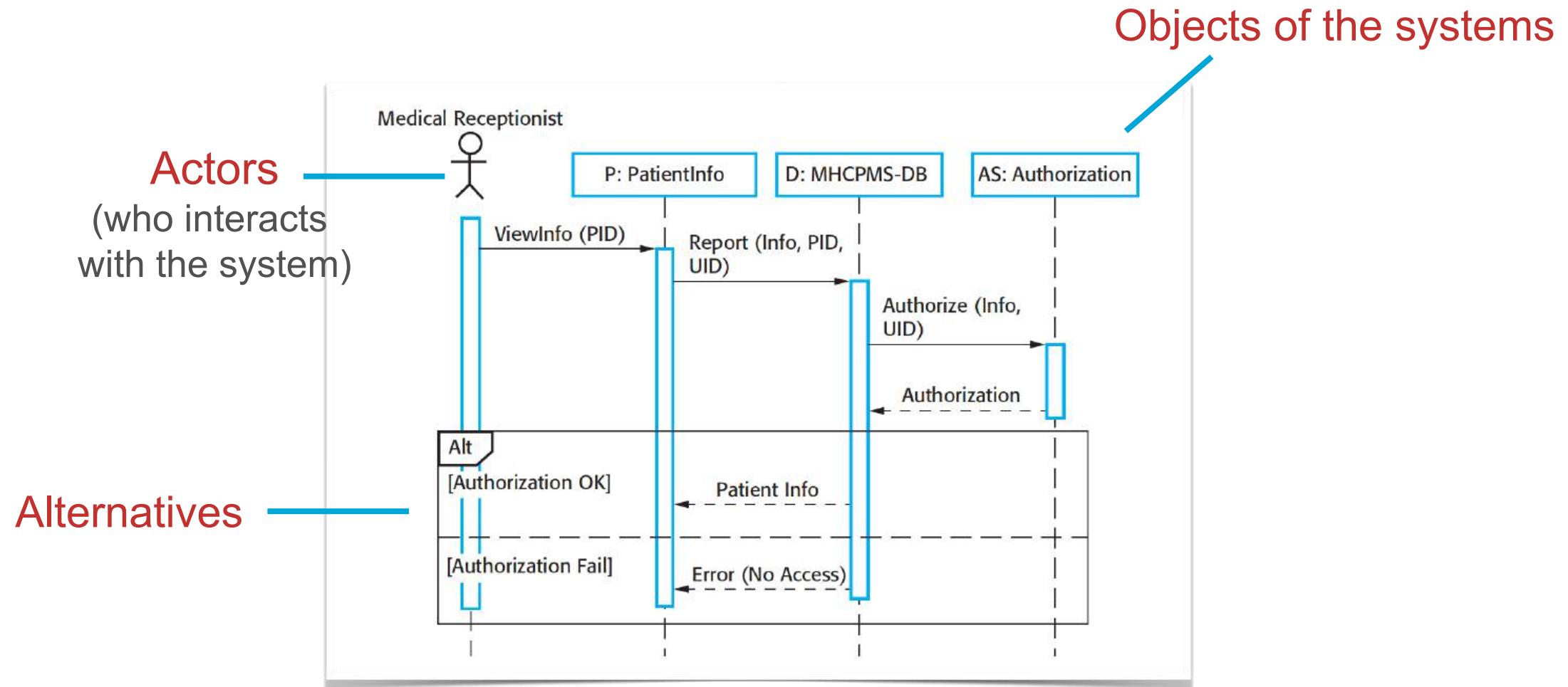


**Features!**

More details about UML  
diagrams in the next lectures

“Software Engineering”  
Ian Sommerville (Figure 4.15)

# Sequence Diagrams



“Software Engineering”  
Ian Sommerville (Figure 5.6)

# Requirements

- A requirement is a statement about one feature (or constraint) of the system
- Express with:
  - Natural language
  - Model-based
    - Graphs (e.g. UML diagrams)
    - Formulas
    - Pseudo-code

Very precise and complicated. Used for systems with high security and safety level (e.g. trains, planes). Formulas can be used to formally verify that security requirements are fulfilled.

# Example of Formal Specification

Example of ATM data dictionary

PIN	::= digit(4)
digit	::= <char><[0..9]>
TransactionReceipt	::= TerminalNo + TransactionNo + CardNo + CustomerName + Date + RequestTypeInfo
CustomerName	::= LastName + FirstName + MI
MI	::=[null letter]
LastName	::= Name
FirstName	::= Name
Name	::= letter(1..26)
letter	::= <char><[a..z,A..Z]>
RequestTypeInfo	::=[InquiryInfo   TransferInfo   DepositInfo   WithdrawInfo]
InquiryInfo	::= AccountNo + CurrentBalance
WithdrawInfo	::= AccountNo + WithdrawAmount + CurrentBalance
AccountNo	::= BankNo + BankAccountNo
CardNo	::=[digit(10)  digit(13)  digit(16)]
BankNo	::= <string(6)><(100000 .. 999999]>
BankAccountNo	::= <string(8)><(10000000 .. 99999999]>
CurrentBalance	::= <float><minimalBalance .. maximalBalance><0.01>
WithdrawAmount	::= <integer><[20 .. 200]><5><><norm><86,42.06>

K.H. Chang et al. "Testing object-oriented programs: from formal specification to test scenario generation". JSS 1998

# Requirements

- A requirement is a statement about one feature (or constraint) of the system
- Express with:
  - Natural language
  - Model-based
    - Graphs (e.g. UML diagrams)
    - Formulas
    - Pseudo-code

Used when the systems have to implement specific algorithms (e.g. bubble sort) or extend existing prototypes.

# Requirements

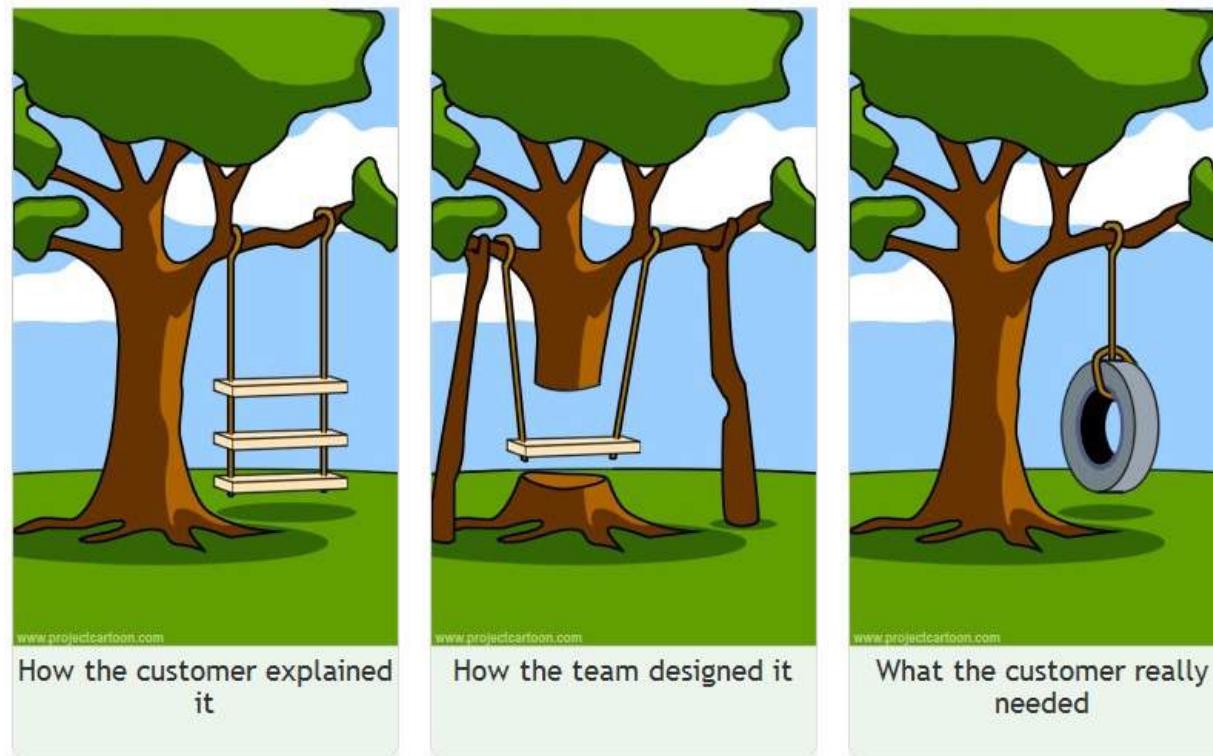
- A requirement is a statement about one feature (or constraint) of the system
- Express with:
  - Natural language
  - Model-based
    - Graphs (e.g. UML diagrams)
    - Formulas
    - Pseudo-code
  - *Artifacts-based*

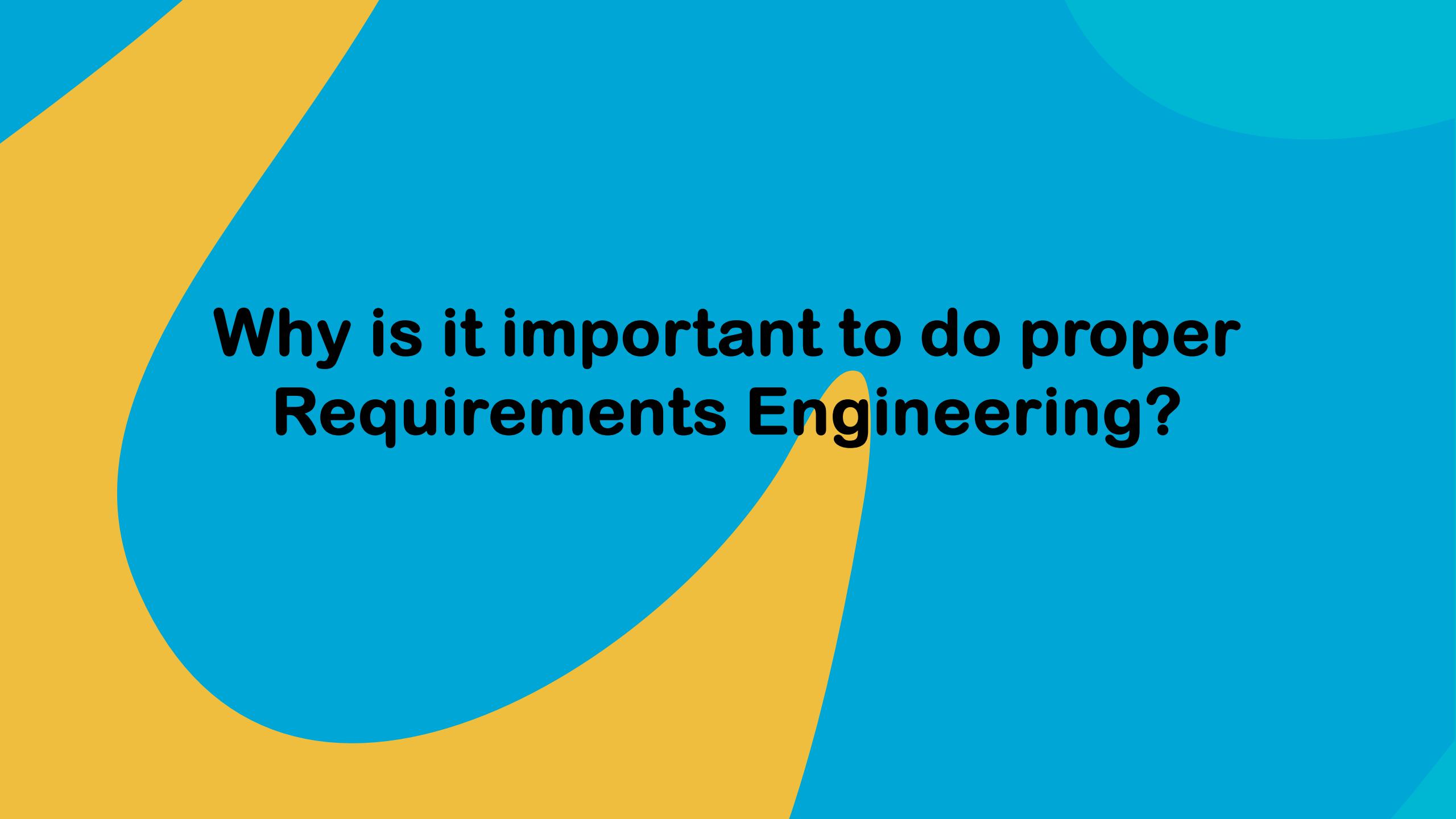
Any other SE artifacts (informal slides, business goals, etc.)

# **Requirement Elicitation**

# Requirement Elicitation

- Determining requirements for your system





# **Why is it important to do proper Requirements Engineering?**

# Bad Requirements Engineering

- Dutch “Omgevingswet”
  - Replaces a lot of laws regarding spatial planning
- Heavily delayed due to the necessary software systems being delayed
  - Every municipality had different requirements and wishes
  - Scope of the project became way too big
- From advice from ICT advice committee
  - Reduce scope, you can't make everyone happy!

# Requirement Elicitation

- Generally (from my experience)
  - What a stakeholder says they want, is often not what they need
  - Stakeholders don't know what is and isn't possible/feasible
  - Developers often lack domain-knowledge which can cause misunderstandings and confusion (DDD, week 3)

# Stakeholder

- A person or organization who influences a system's requirements or who is impacted by that system.
- Additionally, a person who has an interest in the system under development.

# Identifying Stakeholders

- Developers are also stakeholders!
- Constraints introduced by the environment  
*(passive stakeholders, e.g. legislation)*



##/##

Join at: [vevox.app](https://vevox.app) ID: 159-004-429

Opening Poll

# Which stakeholders does Brightspace have?



##/##

Join at: [vevox.app](https://vevox.app) ID: 159-004-429

Open in a Poll

# Which stakeholders does Brightspace have?



**Why is it important to  
identify all stakeholders?**

# TU Delft – Vocareum

- Platform for evaluating student code (Jupyter Notebooks)
- International tender written out
- Many stakeholders considered
  - ... but not Computer Science and Engineering
- Results:
  - Vocareum cannot spec test Java code
  - Computer Science can mostly not use vocareum
  - WebLab was not considered as an alternative

# How not to miss Stakeholders?

- Think about edge cases: rare stakeholders
  - People with disabilities
  - Underrepresented groups
- Future user groups
  - Stakeholders that aren't there yet

# Underrepresented Stakeholders

- TU Delft coffee machines “detect” whether there is a cup
- ... it does not work with black cups



vs



# Elicitation Techniques

- Interviews
- Questionnaires
- Writing Scenarios
  - Use Cases
- Contextual Inquiry
- Brainstorming
- ...



# Interviews with Stakeholders

- People love talking about their work
- Closed vs. semi-structured vs. open
- Can give a lot of insight
- Cons:
  - Domain knowledge may be difficult to explain
  - Fundamental details may be skipped (everyone knows ...)

# Questionnaire

- Can be useful to get general opinions
- Good at discovering edge cases to specific requirements/situations
- Cons:
  - Already requires some knowledge to make a useful questionnaire
  - Biases can occur from the questionnaire

# Scenarios with Stakeholders

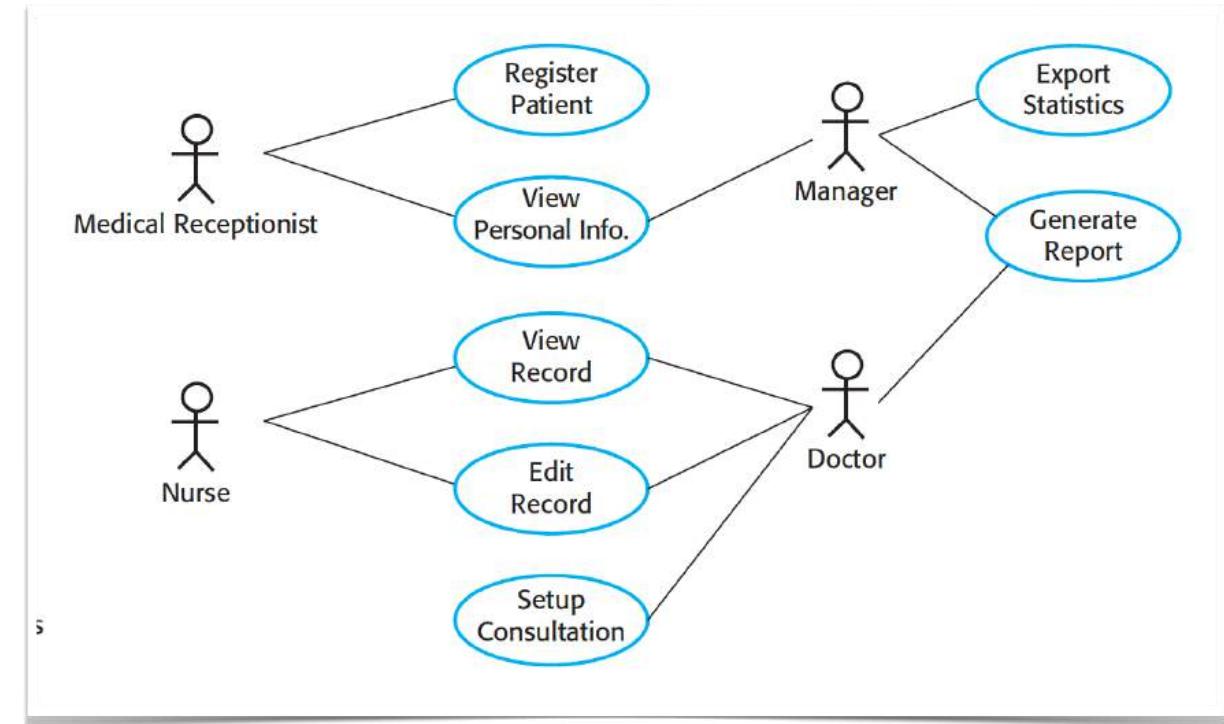
- Easy for stakeholders to relate to real life examples
- Can understand and criticize scenario
- Scenario components:
  - 1. Start situation of user + system, expectations
  - 2. Flow of events
  - 3. What can go wrong + handling
  - 4. Information on other parallel activities
  - 5. Finish situation: system state + user outcome

# Example Scenario

- Start state
  - Various students have entered preferences for timeslots and saved them.
  - Preference editing is no longer possible due to passing of the deadline.
  - Teacher is signed in and wants to create the allocation.
- Events:
  - Teacher clicks on “Group Allocations”
  - Teacher enters parameters for the allocation (group sizes, algorithm, pregrouping types)
  - Teacher clicks on “Create draft allocation”
- What can go wrong? + Handling
  - Entered parameters can be incorrect. An error should be shown which points out which specific parameters are wrong and why.
  - Allocation can be impossible. An error should be shown to the teacher instructing them how to resolve it.
- Parallel activities
  - Other users are using the system normally.
  - Other allocations are happening in parallel.
- Finish state
  - The system shows a draft allocation based on the preferences and parameters given.

# Use Cases

- One type of scenarios
- Captured in high-level use-case diagrams
- More on this tomorrow



# Contextual Inquiry (“Apprenticing”)

- Observe people while they work
- Immersion in working environment where system will be deployed
- Have people talk about what they are doing (and why)
- Don’t understand something? Start a discussion
- Also called “Apprenticing”

# Which Technique to use when?



0/9

Join at: [vevox.app](https://vevox.app) ID: 159-004-429

Question

# Which elicitation technique would you use with students?

Interview

0%

Questionnaire

0%

Writing Scenarios together

0%

Contextual Inquiry

0%

Brainstorming

0%

# Which elicitation technique would you use with students?

Interview

 0%

Questionnaire

 0%

Writing Scenarios together

 0%

Contextual Inquiry

 0%

Brainstorming

 0%



#/#/#

Join at: [vevox.app](https://vevox.app) ID: 159-004-429

Question

# Which elicitation technique would you use with teachers?

Interview

0%

Questionnaire

0%

Writing Scenarios together

0%

Contextual Inquiry

0%

Brainstorming

0%

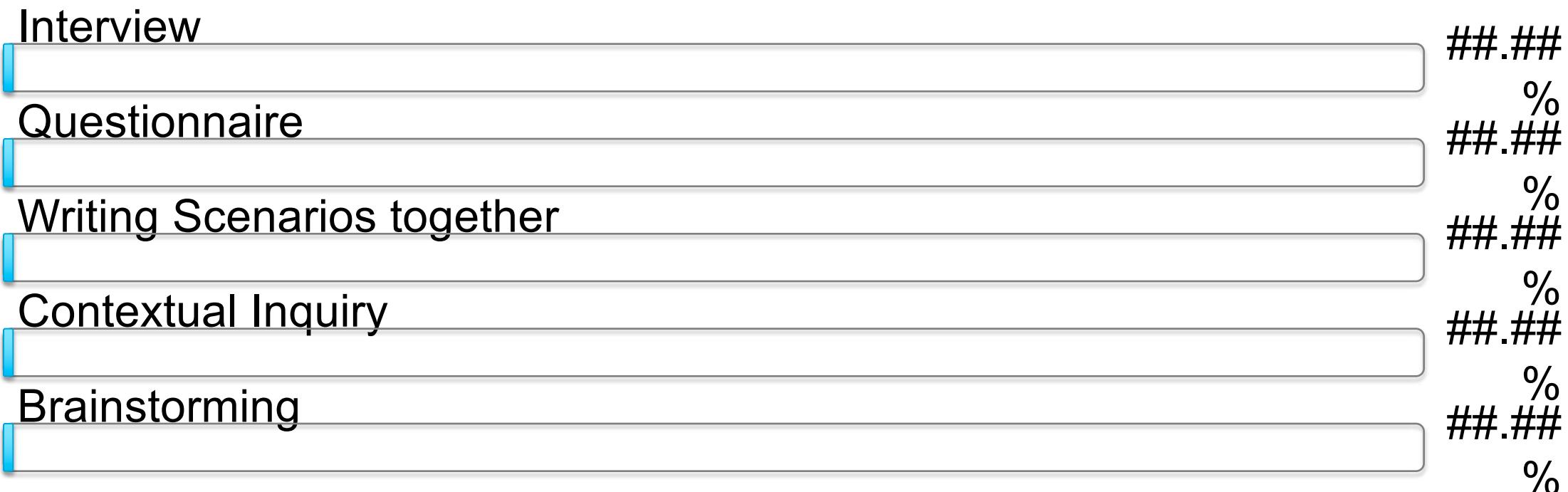


#/#/#

Join at: [vevox.app](https://vevox.app) ID: 159-004-429

Results slide

# Which elicitation technique would you use with teachers?



# Special Lab Session

- Requirements Elicitation
- Interview a TA (stakeholder in the system)
- Enrol for a slot on Brightspace
- Fixed location assignment

# Special Lab Session

TU Delft CSE2115 Software Engineering Method... Taico Aerts as Student

Course Home Content Collaboration ▾ Assignments Grades Help

## Groups

[View Available Groups](#)

My Groups

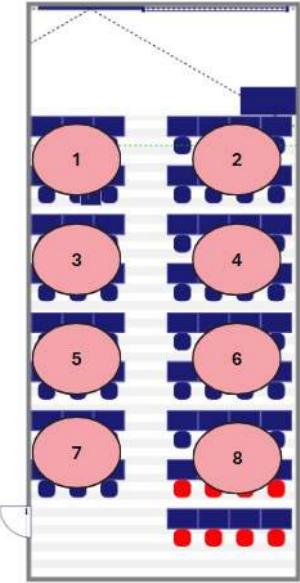
- Discussions
- Virtual Classroom
- Groups
- Group Self Enrollment

Help

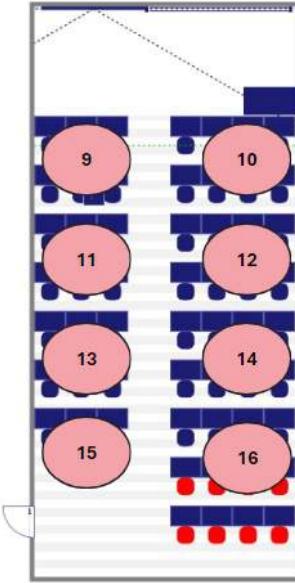
Groups	Members	Assignment	Discussions	Locker	Email	Actions
Lab Week 1 - Requirements						Expiry Date: 05 September, 2024 14:30
Requirements (14:00-14:25) # 1 - Queue	5/5 (Full)					<a href="#">Leave Group</a>

# Lab – Location Assignment

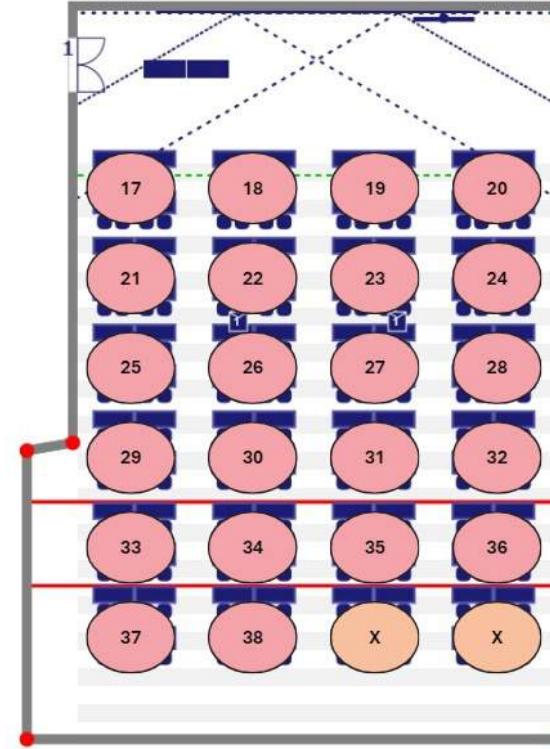
Dreibelweg IZ-1



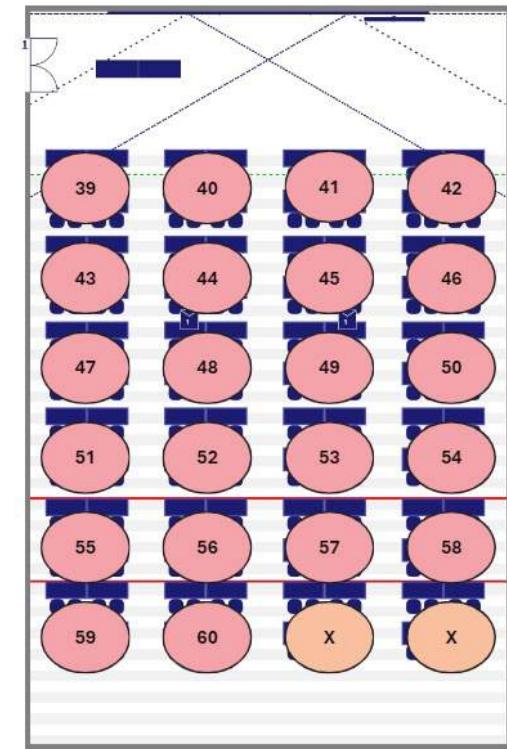
Dreibelweg IZ-2



Flux Hall A



Flux Hall B



# Requirements Engineering

- Figuring out what to make is essential
- → Identifying all stakeholders is important
- → You need to understand what they need
- → Use appropriate requirements elicitation techniques

# Preparation for Friday

- Read UML @ Classroom
- Chapter 1 and 3
- Available online:
  - <https://tudelft.on.worldcat.org/oclc/904123284>

# Friday

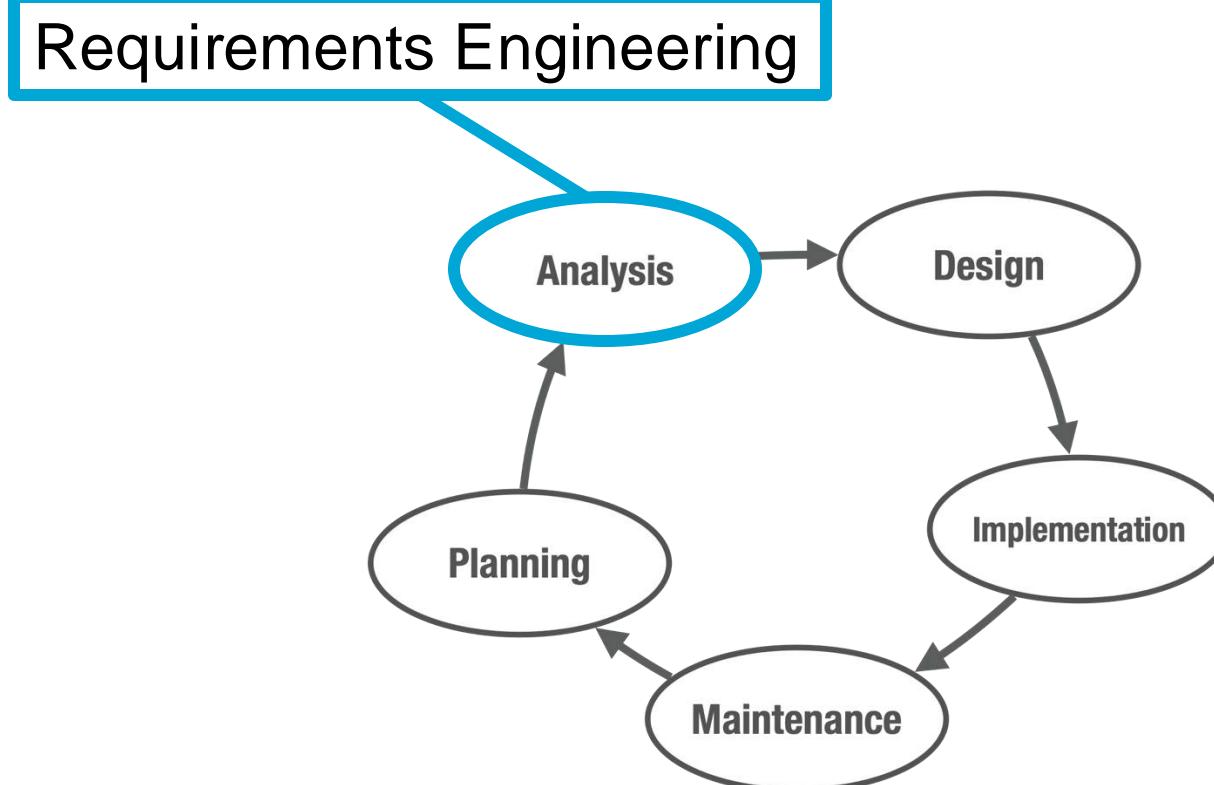
- UML diagrams
- How to communicate effectively with stakeholders
- How to make a good requirement
- Moving from requirements to design

# **CSE2115 Software Engineering Methods**

**Requirements Engineering & UML**

Taico Aerts & Arie van Deursen

# Software Development Lifecycle



# Learning Objectives

- Identify functional vs. non-functional requirements.
- Determine when to use user vs system requirements.
- Draft a simple requirements specification document for a given software system.
- Evaluate and revise requirements to ensure they are SMART
- Draw use case diagrams and determine the correct level of detail for the situation.
- Use MoSCoW and Kano methods to prioritize requirements.

The background features a large, light blue circle on the left side, partially overlapping a yellow triangle pointing towards it. The yellow triangle is positioned in the lower right quadrant. The overall composition is minimalist and modern.

# Last Time

# Requirements

- A requirement is a statement about one feature (or constraint) of the system
- Express with:
  - Natural language
  - Model-based
    - Graphs (e.g. UML diagrams)
    - Formulas
    - Pseudo-code
  - *Artifacts-based*

# Elicitation Techniques

- Interviews
- Questionnaires
- Writing Scenarios
  - Use Cases
- Contextual Inquiry
- Brainstorming
- ...



# Example Scenario

- Start state
  - Various students have entered preferences for timeslots and saved them.
  - Preference editing is no longer possible due to passing of the deadline.
  - Teacher is signed in and wants to create the allocation.
- Events:
  - Teacher clicks on “Group Allocations”
  - Teacher enters parameters for the allocation (group sizes, algorithm, pregrouping types)
  - Teacher clicks on “Create draft allocation”
- What can go wrong? + Handling
  - Entered parameters can be incorrect. An error should be shown which points out which specific parameters are wrong and why.
  - Allocation can be impossible. An error should be shown to the teacher instructing them how to resolve it.
- Parallel activities
  - Other users are using the system normally.
  - Other allocations are happening in parallel.
- Finish state
  - The system shows a draft allocation based on the preferences and parameters given.

# **Special Lab Assignment**

# Special Lab Assignment

- How was it to do this?
- Any special insights gained?
- “Apprenticing” vs Interviews?

# Stakeholders

- Queue
- Submit
- TAM
- Edge case stakeholders?

# **Different Types of Requirements**

# User vs System Requirements

- **User requirement:** Statements (natural language + diagrams) of what services the system is expected to provide to users.
- **System requirement:** Detailed descriptions of the software system's functions, services and operational constraints (how the system provides this).

# Tenders / Contract Bidding

- You write a list of *user requirements*
- Other companies propose their “solution” which can meet these requirements
- Once selected, a **System Requirement Specification** is made
  - System requirements (detailed requirements)

# Example User Req → System Req

- Students can search for books by title, author or ISBN
- System requirements?
  - Fuzzy matching on the title/author
  - Case insensitive
  - Pagination
  - Results must be returned within 5 seconds

# Example User Req → System Req

- Students can search for books by title, author or ISBN
- Functional
  - The system shall provide a search function that allows users to search for books by title, author or ISBN
  - The search results shall display a list of books matching the search criteria, including the book title, author, ISBN and availability status.
  - The system shall allow users to filter search results by genre, publication date and customer ratings.
  - The system shall support pagination to handle large sets of search results.
- Non-functional
  - The search feature should return results within 2 seconds for up to 100,000 books.
  - The system shall be able to handle up to 500 simultaneous search requests without degradation of performance.

# Difference requirements and functionalities

- What is the difference between a requirement and a functionality?
  - A requirement is something that someone wants from the system
  - A functionality is something that the system exhibits
  - A requirement can require a functionality from the system.
  - There are requirements that are not functionalities.

# Classifying Requirements

- Requirement is description of what the system should do:
- The services that it provides Functional
- The constraints on its operation Non-Functional

# Functional vs. Non-functional

- Functional
  - What services the system should provide
  - How the system should react to particular inputs
  - What the system shouldn't do
- Non-functional
  - Constraints on the services or functions offered
  - Timing constraints
  - Constraints on the development process
  - Constraints imposed by standards

# Functional?

- Question: Which of the following are functional requirements:
  - The system enables users to place lunch orders.
  - The system always responds to user clicks in less than one tenth of a second.
  - The system displays a list of hotel vacancies.
  - Users' credential must be stored in a MySQL database



##/##

Join at: [vevox.app](https://vevox.app) ID: 177-628-747

Question

# Which of the following are functional requirements?

The system enables users to place lunch orders.

0%

The system always responds to user clicks in less than one tenth of a second.

0%

The system displays a list of hotel vacancies.

0%

Users' credential must be stored in a MySQL database

0%



#/#

Join at: [vevox.app](https://vevox.app) ID: 177-628-747

Results slide

# Which of the following are functional requirements?

The system enables users to place lunch orders.

##.##

The system always responds to user clicks in less than one tenth of a second.

##.## %

The system displays a list of hotel vacancies.

##.## %

Users' credential must be stored in a MySQL database

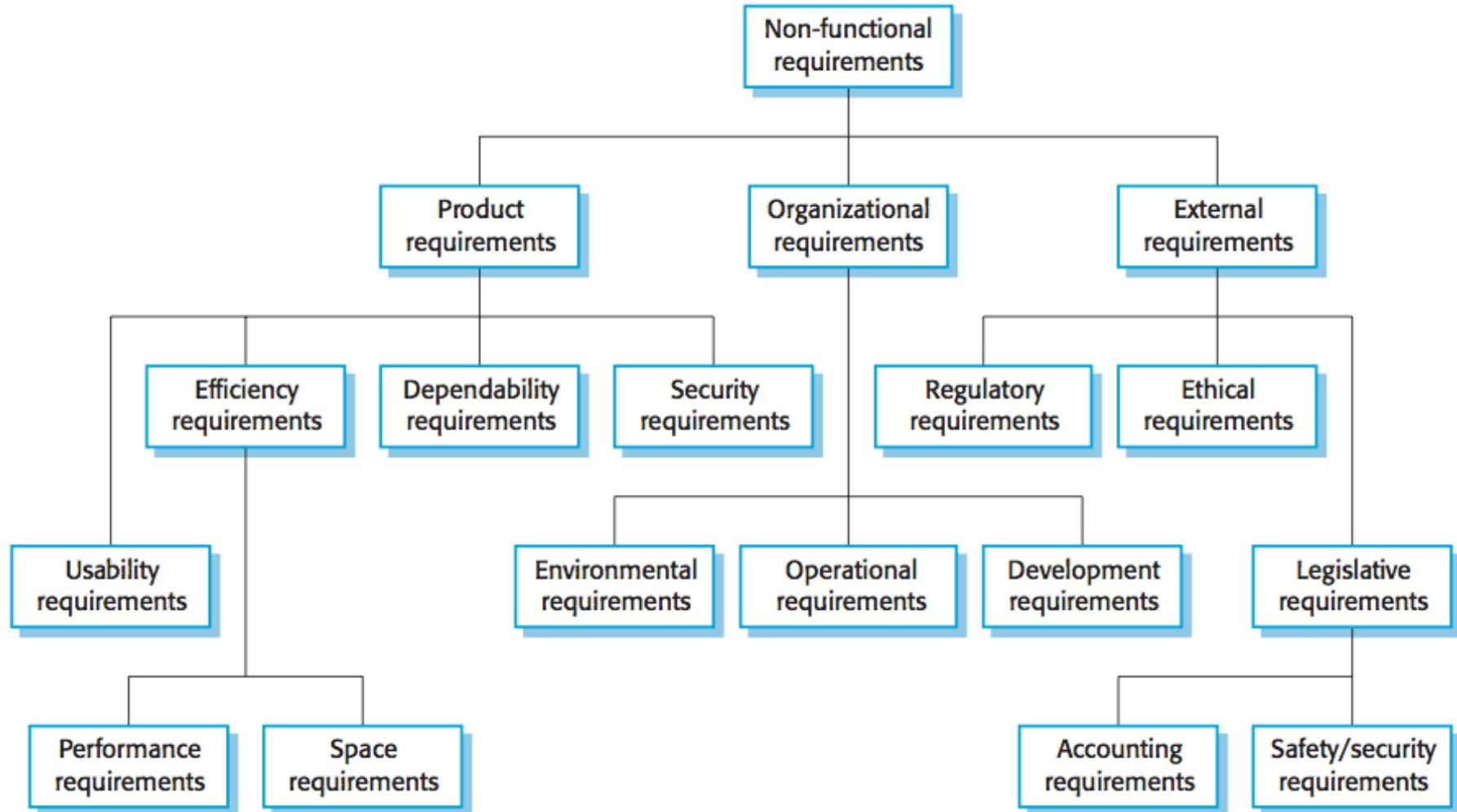
##.## %

# RESULTS SLIDE

# Non-Functional Requirements

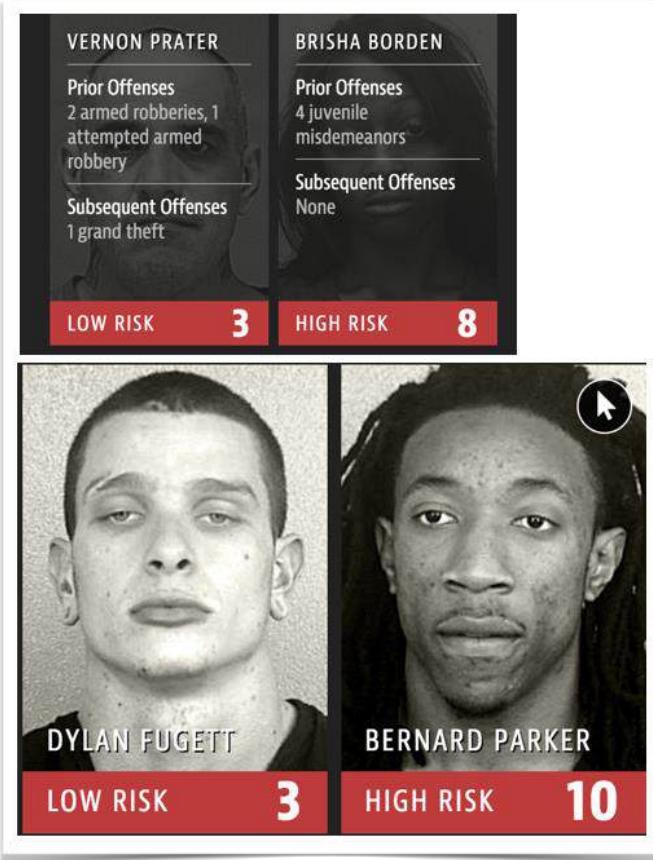
Product requirements	Specify that the product <b>must behave</b> in a particular way. Examples: performance requirements on how fast the system must execute and how much memory it requires, reliability, etc.
Organizational requirements	These requirements are broad system requirements derived from policies and procedures in the customer's and developer's organization. Examples: <ul style="list-style-type: none"><li>• Process standards used</li><li>• Programming language</li><li>• Operating System</li></ul>
External requirements	Arise from <b>factors which are external</b> to the system and its development process e.g., legislative requirements (such as GDPR)

# Categories of Non-functional Reqs



“Software Engineering”, Ian Sommerville (Figure 4.3)

# Ethical Requirements



COMPAS was a system used in Florida (and some other states in US) to predict whether or not a criminal was likely to recidivate.

The system was designed to optimize for overall accuracy (on a training set) through regression models.

The model predicted double the number of false positives for recidivism for African American ethnicities than for Caucasian ethnicities.

# Ethical Requirements

Let's use Google Translate for the language pair English -> Chinese

English	Chinese (Google Translation)	Notes
<b>Men</b> do <b>good</b> research in computer science.	Nanren zaijisuanji kexue fangmian zuole <b>hen hao de yanjiu</b> 男人在计算机科学方面做了很好的研究	good → hen hao de (very good)
<b>Women</b> do <b>good</b> research in computer science.	Nüxing zaijisuanji kexue fangmian zuole <b>henduo yanjiu</b> 女性在计算机科学方面做了很多研究	good → henduo (a lot)
<b>Male</b> students do <b>good</b> research in computer science.	Nan xuesheng zaijisuanji kexue fangmian zuole <b>hen hao de yanjiu</b> 男学生在计算机科学方面做了很好的研究	good → hen hao de (very good)
<b>Female</b> students do <b>good</b> research in computer science.	Nü xuesheng zaijisuanji kexue fangmian zuole <b>henduo yanjiu</b> 女学生在计算机科学方面做了很多研究	good → henduo (a lot)

Inconsistent (unfair) translation when changing the gender of the subject

Z. Sun et al, "Automatic Testing and Improvement of Machine Translation," arXiv 2019

# **Creating good (system) requirements**

# Requirement Verifiability

Requirements must be written to be objectively verified

## Imprecise:

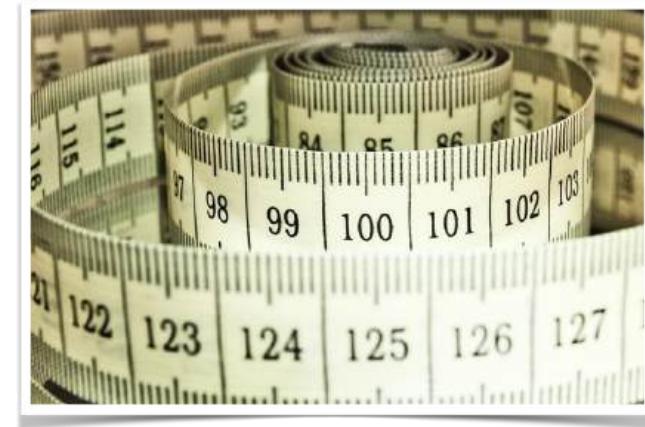
- The system should be easy to use by experienced controllers and should be organized in such a way that user errors are minimized
- Phrases like ‘easy to use’ and ‘errors shall be minimized’ are useless specifications

## Verifiable:

- Experienced controllers should be able to use all the system functions after a total of two hours of training. After this training, the average number of errors made by experienced users should not exceed two per day

# Precise Requirements Measures

Property	Measure
Speed	Processed transactions/second, User/Event response time, Screen refresh time
Size	Megabytes, Number of RAM chips
Ease to use	Training time, Rate of errors made by users, Number of help dialogs
Reliability	Mean time to failure, Probability of unavailability, Rate of failure occurrence, etc.
Robustness	Time to restart after failure, Percentage of events causing failure, etc.
Portability	Percentage of target dependent statements, Number of target systems



# **SMART Requirements**

- **Specific**
- **Measurable**
- **Achievable**
- **Relevant**
- **Time-related**

# SMART Requirements

- Specific
  - The goal is concrete and tangible
- Measurable
  - There is an objective measure of success
- Achievable
  - Achievable with the resources available
- Relevant
  - Meaningfully contributes to objectives
- Time-related (sometimes optional)
  - Includes a timeline for expected results

# SMART WebLab?

- The system should be user-friendly.
- The system should provide feedback to students on their submissions.
- The system should support code submissions in Python, Java, and C++.
- The system should test and evaluate code submissions within 1 second.
- The system should allow students to customize the background colour of the interface.

# Writing Requirements Together

- The system should be user-friendly.
- System should achieve certain level of user satisfaction when used.
  - Should have keyboard shortcut
  - High contrast
  - Universal guidelines to usability
  - Average time for executing a particular action

# Writing Requirements Together

- The system should provide feedback to students on their submissions.
- What type of feedback
  - Feedback configurable to ... criteria relevant to the task
  - Number of tests passed/failed
  - Output from extensions
  - Give feedback before your next submission?
  - Add at most 1 second to the execution time of the task outside of WebLab



##/##

Join at:

vevox.app

ID: 177-628-747

Question

# SMART? "The system should be user-friendly."

Specific

0%

Measurable

0%

Achievable

0%

Relevant

0%

Time-Bound

0%



#/#/#

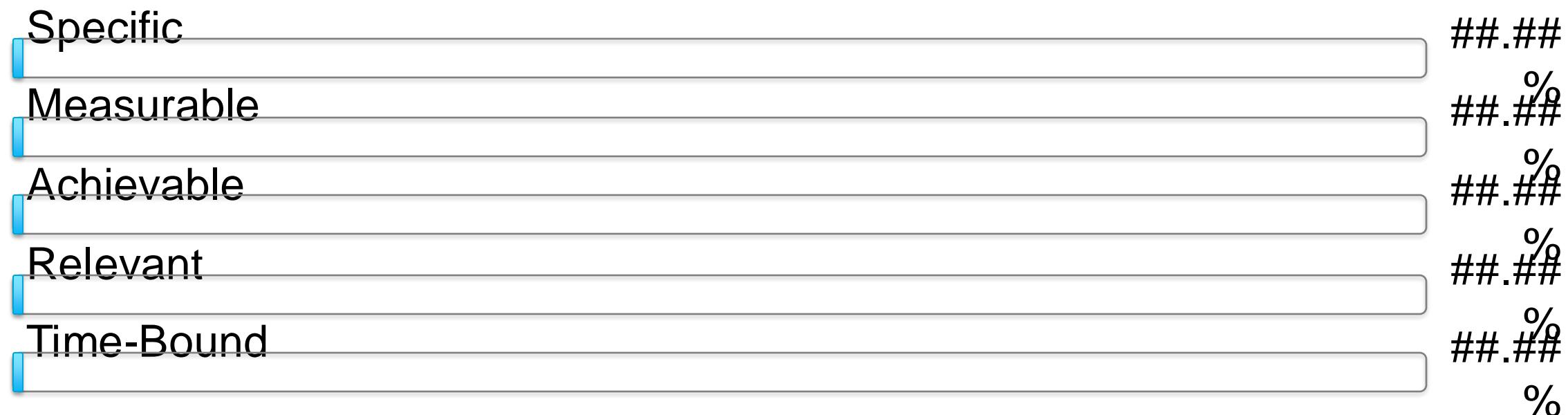
Join at:

vevox.app

ID: 177-628-747

Results slide

# SMART? "The system should be user-friendly."



# RESULTS SLIDE



0/31

Join at:

vevox.app

ID: 177-628-747

Question

# SMART? "The system should provide feedback to students on their submissions."

Specific

0%

Measurable

0%

Achievable

0%

Relevant

0%

Time-Bound

0%

# SMART? "The system should provide feedback to students on their submissions."

Specific	##.##
Measurable	##.%
Achievable	##.##%
Relevant	##.##%
Time-Bound	##.##%

# RESULTS SLIDE



0/39

Join at:  
[vevox.app](https://vevox.app)

ID: 177-628-747

Question

# SMART? "The system should support code submissions in Python, Java, and C++."

Specific

0%

Measurable

0%

Achievable

0%

Relevant

0%

Time-Bound

0%



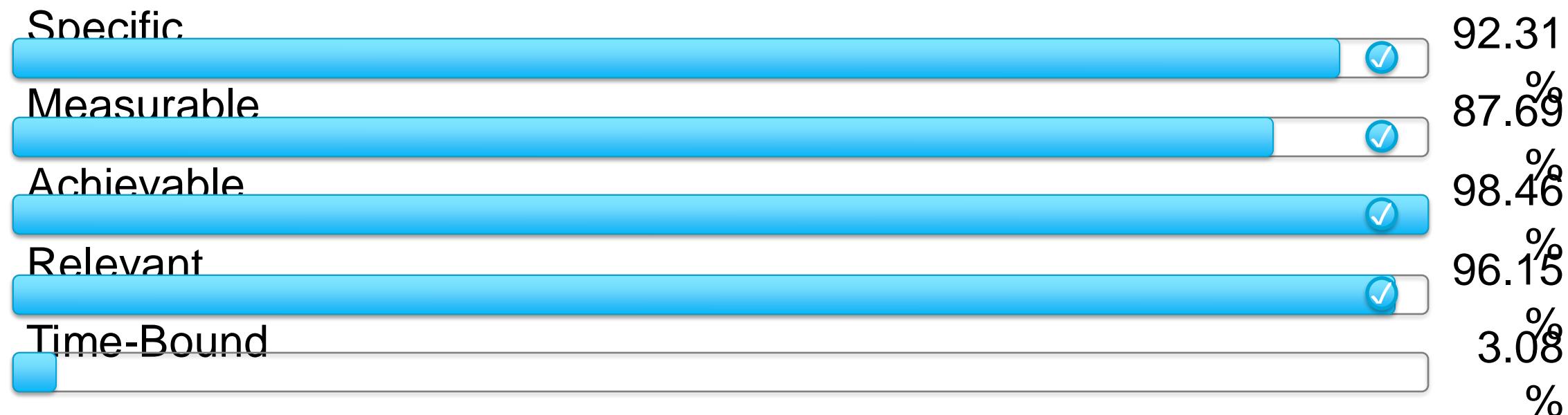
0/39

Join at:  
vevox.app

ID: 177-628-747

Preparin

# SMART? "The system should support code submissions in Python, Java, and C++."



# RESULTS SLIDE



##/##

Join at:  
vevox.app

ID: 177-628-747

Question

# SMART? "The system should test and evaluate code submissions within 1 second."

Specific

0%

Measurable

0%

Achievable

0%

Relevant

0%

Time-Bound

0%



##/##

Join at:  
vevox.app

ID: 177-628-747

Showing

# SMART? "The system should test and evaluate code submissions within 1 second."

Specific



0%

Measurable



0%

Achievable

0%

Relevant



0%

Time-Bound



0%

# RESULTS SLIDE



0/46

Join at:  
[vevox.app](https://vevox.app)

ID: 177-628-747

Question

# SMART? "The system should allow students to customize the background colour of the interface."

Specific

0%

Measurable

0%

Achievable

0%

Relevant

0%

Time-Bound

0%



0/46

Join at:  
[vevox.app](https://vevox.app)

ID: 177-628-747

Showing

# SMART? "The system should allow students to customize the background colour of the interface."

Specific



0%

Measurable



0%

Achievable



0%

Relevant



0%

Time-Bound



0%

# RESULTS SLIDE

# SMART – Specific

- The system should provide feedback to students on their submissions.
- The system should provide feedback on code submissions by identifying syntax errors, suggesting possible corrections, and providing examples within 5 seconds after the code is submitted.

# SMART – Measurable

- The system should be user-friendly.
- The system should achieve a usability score of at least 80 out of 100 in usability tests conducted with a group of 20 students (diverse group).

# SMART – Achievable

- The system should test and evaluate code submissions within 5 seconds.
- The system should test and evaluate code submissions within 10 seconds for programs with up to 500 lines of code and within 30 seconds for programs with up to 2,000 lines of code.

# SMART – Relevant

- The system should allow students to customize the background colour of the interface.
- The system should allow students to choose between a light mode and a dark mode for better readability and comfort while writing and debugging code.

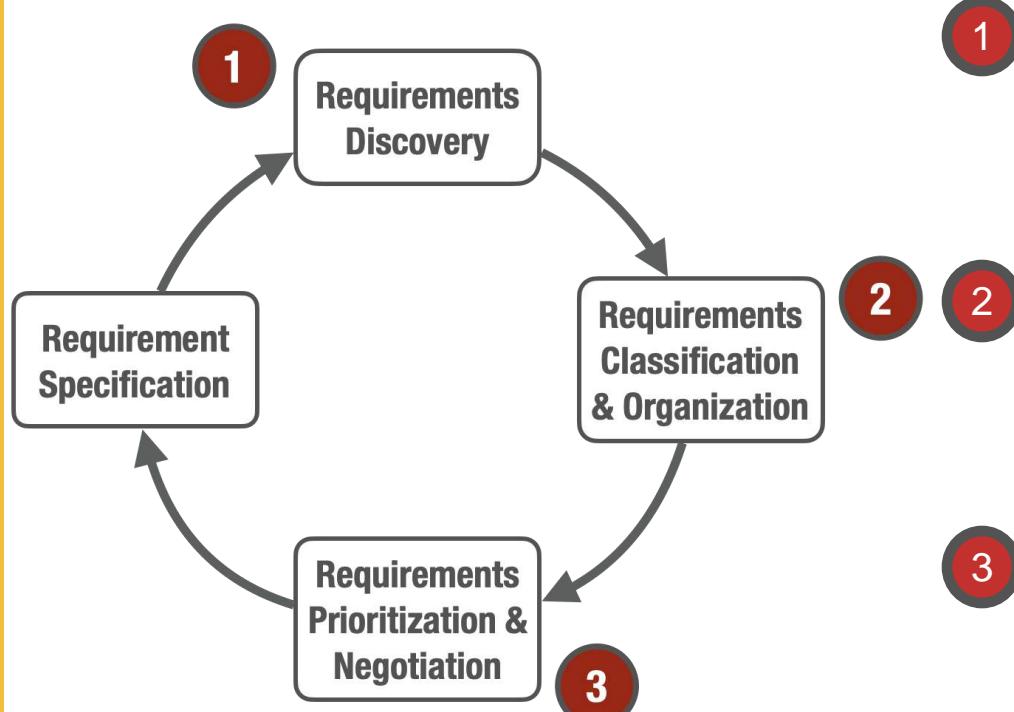
# SMART – Time-Bound

- The system should support code submissions in Python, Java, and C++.
- The system should support code submissions in Python by Q1 2024, Java by Q2 2024, and C++ by Q3 2024.

NB: Not every requirement needs to be time-bound!

# Prioritization

# Requirement Analysis Process



- 1 Identify the stakeholders and use elicitation techniques (e.g., interview) to find requirements
- 2 Classify the requirements in function and non-functional requirements. The non-functional requirements can be further organized/classified
- 3 Make an agreement on the priorities for the requirements
- 4 Output: the requirement specification document

# Prioritization

- OOPP: MoSCoW
- Other techniques?

# Prioritization – MoSCoW Method

- Must have
  - Critical to the current delivery time-box
- Should have
  - Important but not as time-critical, delivered in future time-box
- Could have
  - Desirable but not necessary. Usually improves user or customer experience
- Won't have
  - Least-critical by stakeholders. Includes dropped requirements considered for future versions of the project.

# Prioritization – MoSCoW Method

- Advantages
  - Simple
  - Helps in managing scope
  - Easy to communicate with stakeholders
- Disadvantages
  - Does not help decide between multiple requirements in the same priority.
  - Ambiguity over timing for won't haves
  - Political focus on building new features over technical improvements

# Prioritization – Kano

- Developed by Noriaki Kano (1980s)
- Based on customer satisfaction and delight
- Weigh high-satisfaction against its costs to implement

# Prioritization – Kano – Categories

- Basic Needs (Must-be)
- Performance Needs (One-dimensional)
- Excitement Needs (Attractive)
- Indifferent Needs
- Reverse Needs

# Prioritization – Kano – Categories

- Basic Needs (Must-be)
  - Fundamental features that customers expect
  - Missing? → Dissatisfied customers.
  - Present? → No significant increase in satisfaction
- Performance Needs (One-dimensional)
  - Directly impact customer satisfaction

# Prioritization – Kano – Categories

- Excitement Needs (Attractive)
  - Unexpected features
  - Present? → Delight customers
  - Absent? → No dissatisfaction
- Indifferent Needs
  - Do not significantly impact customer satisfaction
- Reverse Needs
  - Can cause dissatisfaction if implemented. Don't align with customer expectations

# Prioritization – Kano – Steps

- 1. Identify Features
- 2. Survey Customers
- 3. Analyze Data
- 4. Prioritize Features
- 5. Implement and Review

# Prioritization – Kano – Survey

- These 3 questions will be present for each feature you have as a consideration for your product:
  - How do you feel if this feature is present?
  - What is your reaction if you do not have this feature?
  - How important is this feature for you?
- Answers using a scale
  - I like it that way
  - I expect it to be that way
  - I am neutral
  - I can live with it that way
  - I dislike it that way

# Prioritization – Kano – Pros/Cons

- Advantages
  - Helps aligning product development with actual customer needs and expectations
  - Focus on delivering features that enhance customer satisfaction and loyalty
  - Provides insights into features that are essential, which can be enhanced, and which can be omitted
- Disadvantages
  - Requires customer surveys and analysis to accurately categorize features (time-consuming and costly)
  - May require regular reevaluation as customer expectations and perceptions change.



0/10

Join at: [vevox.app](https://vevox.app) ID: 177-628-747

Question

# Rate according to the Kano Method

Interactivity in lectures

No interactivity in lectures

Importance Interactivity

A lot of text on slides

Little text on slides

Importance text amount

Learning objectives per lecture

- I like it that way
- I expect it to be that way
- I am neutral
- I can live with it that way
- I dislike it that way

# Rate according to the Kano Method

I dislike it that way / not important

I like it that way / important

Interactivity in lectures

2

No interactivity in lectures

5

Importance Interactivity

1

A lot of text on slides

5

Little text on slides



Importance text amount

5

Learning objectives per lecture

# RESULTS SLIDE

# **Requirements to Issues**

# Requirements to Issues

- Requirements usually encompass larger features
  - Especially user requirements
- Hard to put directly into an issue
- Break up into subtasks
  - Why?
  - How?

# WebLab Exam Statistics

- When requested, WebLab shall show course managers statistics [about an exam] with for [every subquestion] the [average grade] and [Rit value] (correlation with total score) within [100 milliseconds].
  - Interface with storage/database
  - What is an exam?
  - Storing statistics and recompute when necessary

# Possible approach

- Break down into components a feature needs to work
  - All components already present?
  - Every component already sufficiently extensive to support this functionality?
- Break down into subtasks for the implementation

# Queue Chat

- Queue will have a different mode where students can request a chat with a TA instead of having the TA come by.

# **Using Diagrams in Requirement Elicitation**

# UML



- Object Management Group (OMG)

# UML Intention

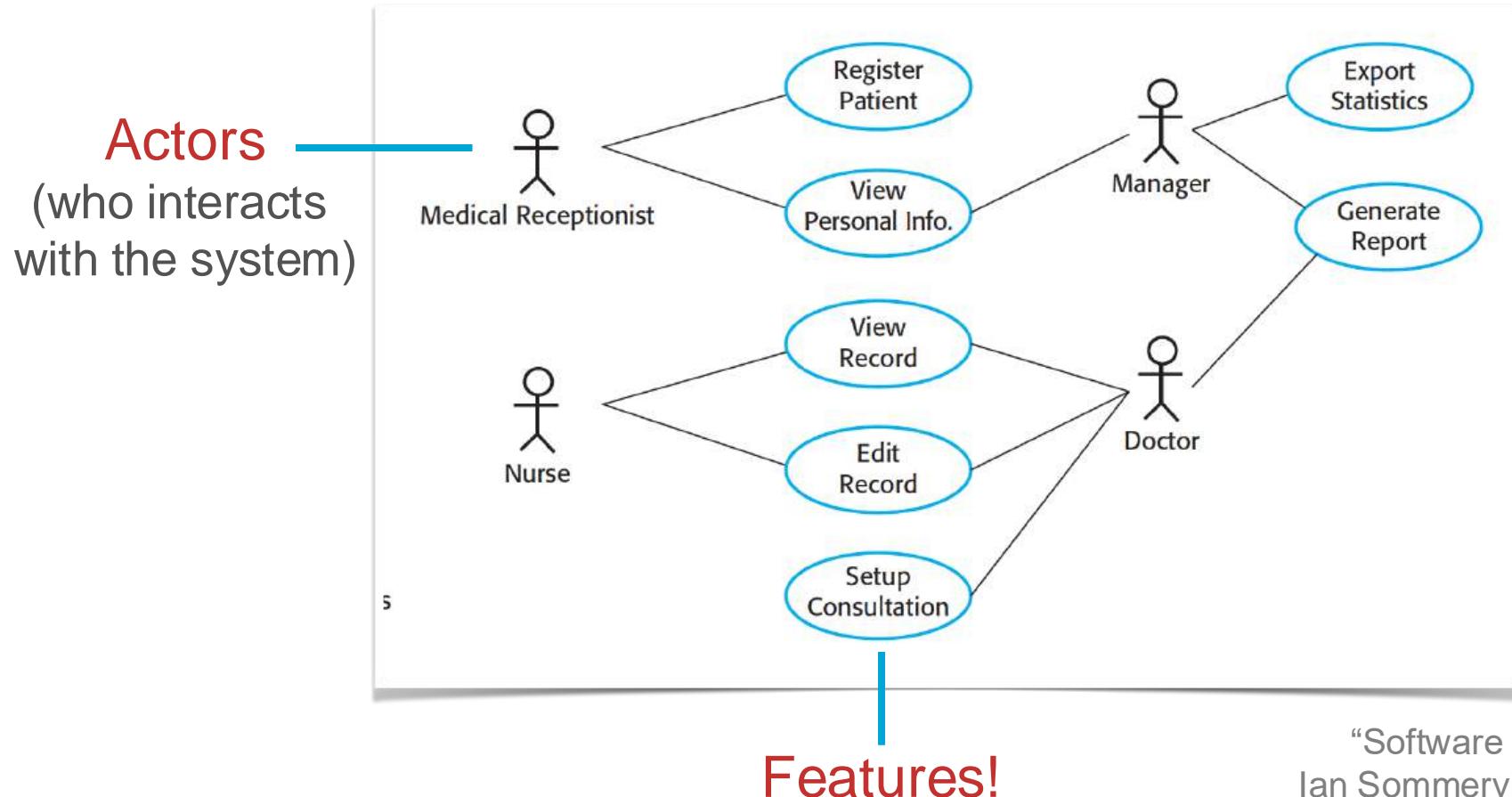
- Intention: powerful modeling language
  - You can completely model all information of what is going on in a system
  - Through transformations between models and in the end to code, you can (in theory) program using UML diagrams
  - That is why UML has so many fancy features
  - (The spec is ~ 800 pages!)

# UML is used for communication

- Between developers/software engineers
- Communication with stakeholders
- For most communication, use a simple subset of UML

# Use Case Diagrams

- Show interactions between a system and its environment



“Software Engineering”  
Ian Sommerville (Figure 4.15)

# Use Case Diagrams for Requirement Elicitation

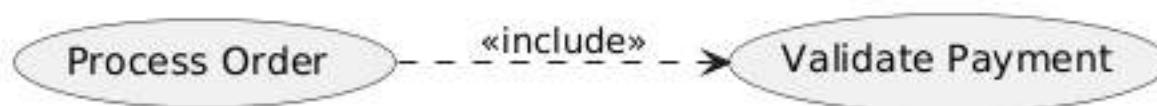
- Simple use case diagrams
  - Easier overview of functionalities to communicate to stakeholder
- Discussion over the use cases
- Identification of missing use cases

# Complex Use Case Diagrams

- More suited for communication between developers or software engineers when it comes to design.
- Include is used for indicating actions that also need to happen after other uses cases (automatically)

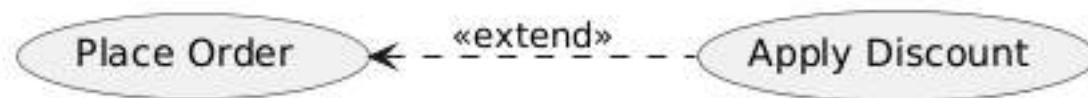
# Complex Use Case Diagrams

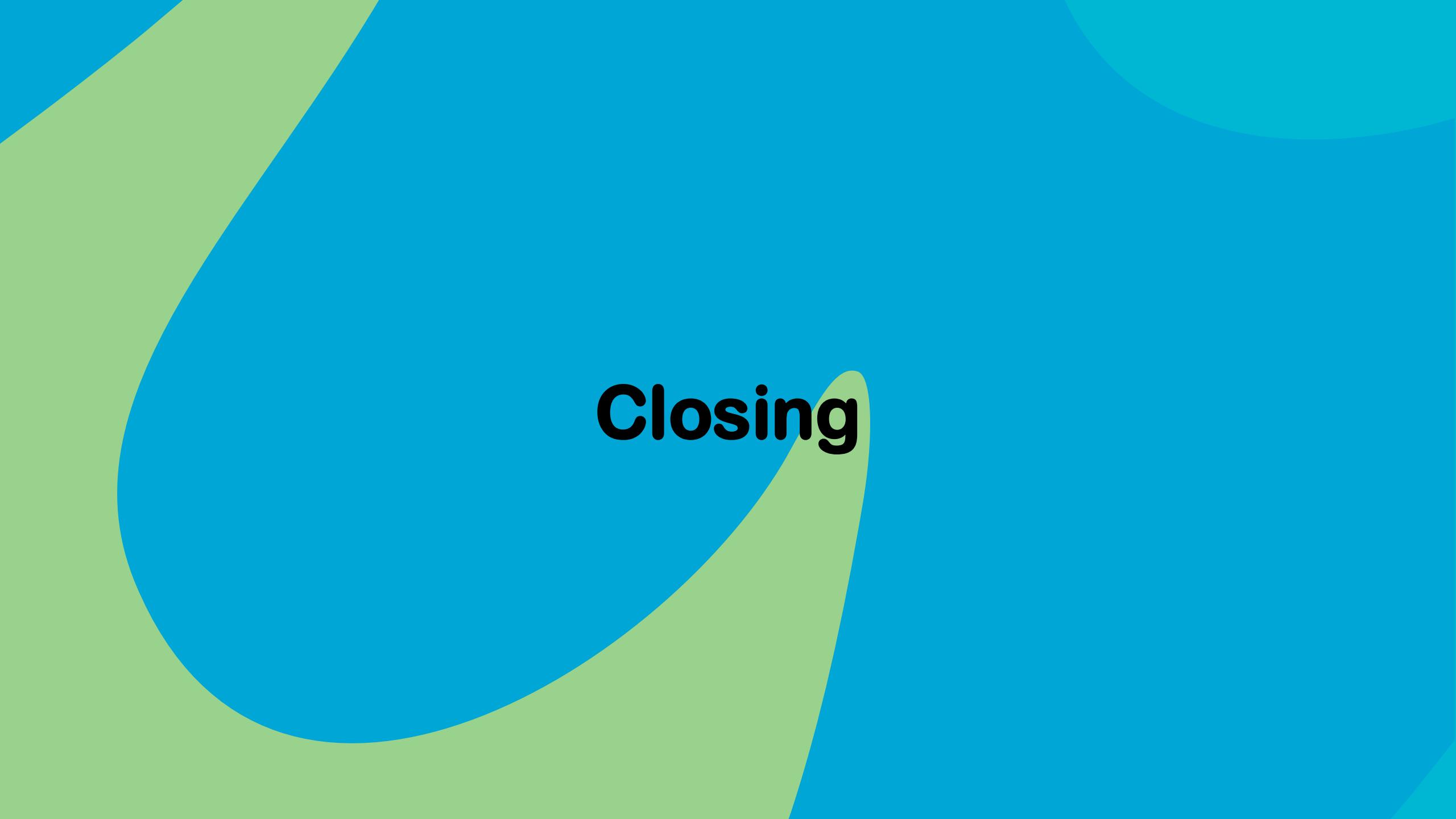
- Difference <<include>> and <<extend>>
  - Include
    - Indicates that one use case requires the behavior of another. It cannot be completed without the included use case also being executed at least once.
    - Can be used to extract common behavior across multiple use cases



# Complex Use Case Diagrams

- Difference <<include>> and <<extend>>
  - Extend
    - When a use case conditionally adds behavior to another use case. Optional relationship.
    - The base use case can be used without triggering the extending use case. Under certain conditions, it will trigger the extending use case.





# **Closing**

# Question from WebLab

- What is the difference between the characteristics by Herbert Stachowiak and the properties of a model from Bran Selic, are they 2 interpretations for what a model should look like?
  - Stachowiak more general. Selic specific to SE
  - Both agree on importance of abstraction

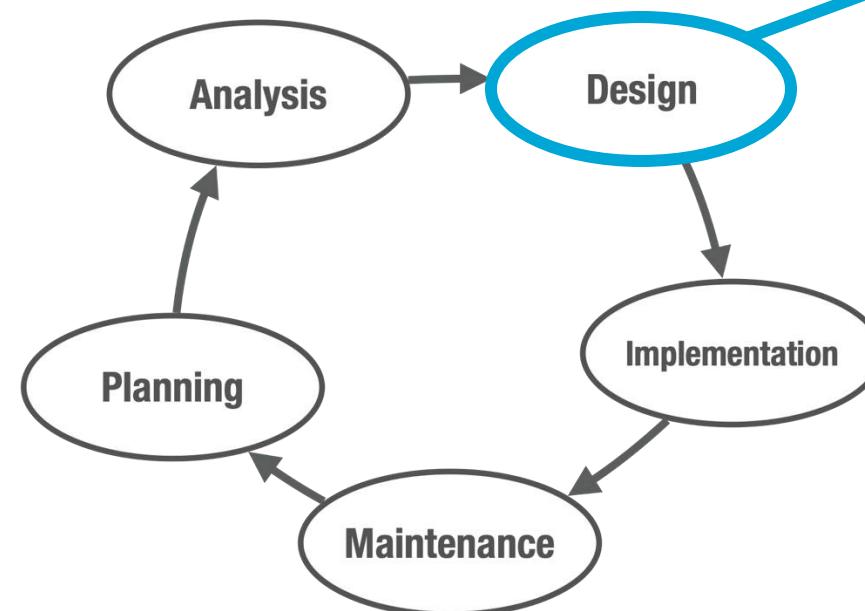
# Other Questions

# OP Allocation

- Brightspace Announcement with assigned slot and location
- Today 13:45~17:45 (depending on issues)
- Make sure to check!
  - First slots on Monday afternoon

# Next Time

Software Design / Architecture

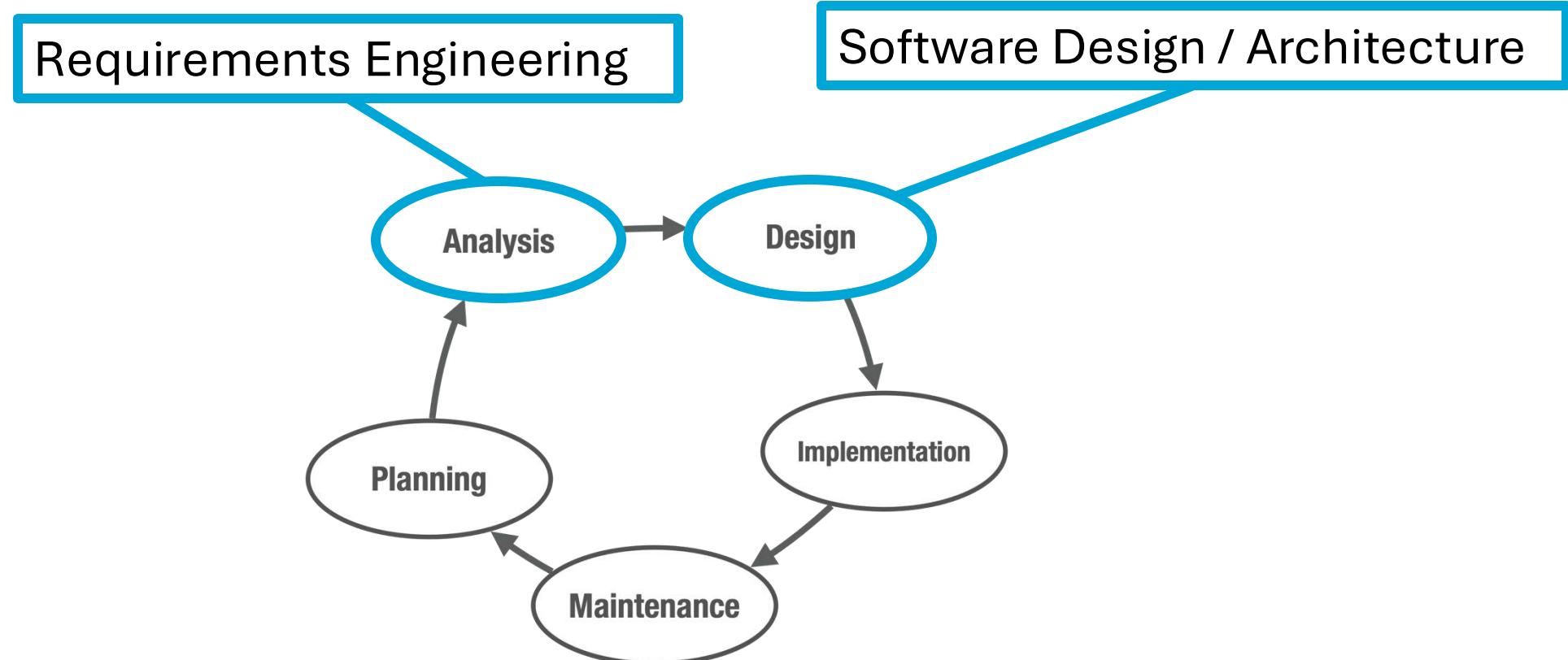


# **CSE2115 Software Engineering Methods**

**Software Architecture**

**Taico Aerts & Arie van Deursen**

# Software Development Lifecycle



# Learning Objectives

- Understand the fundamentals of Software Architecture.
- Describe common architectural patterns, including their benefits and limitations.
- Apply different architectural views to present an architecture tailored to different stakeholders and purposes.
- Create high-level UML diagrams (Component, Class, Sequence) to represent software structures and interaction.
- Apply the C4 model to design clear and comprehensive architectural overviews.

# Goal of the Lecture

- This one lecture will not make you a Software Architect
- We aim to give a high level overview of the important aspects and considerations

# Overview

- What is Software Architecture (SA)?
- How to show/discuss SA?
- Architectural Patterns

# Last Time

# User vs System Requirements

- **User requirement:** Statements (natural language + diagrams) of what services the system is expected to provide to users.
- **System requirement:** Detailed descriptions of the software system's functions, services and operational constraints (how the system provides this).

# **SMART Requirements**

- **S**pecific
- **M**easurable
- **A**chievable
- **R**elevant
- **T**ime-related

# Prioritization – MoSCoW Method

- Must have
  - Critical to the current delivery time-box
- Should have
  - Important but not as time-critical, delivered in future time-box
- Could have
  - Desirable but not necessary. Usually improves user or customer experience
- Won't have
  - Least-critical by stakeholders. Includes dropped requirements considered for future versions of the project.

# Prioritization – Kano

	Interactive	Text on slides	Learning Objectives	Scheduling of SEM	Optional lab	Slides available
Basic Need	40	18	44	14	30	20
Performance	19	5	10	6	20	41
Excitement	1	0	4	4	5	0
Indifferent	17	14	24	16	13	16
Reverse	12	44	6	49	3	6
Importance	3.59	3.68	3.67	3.87	3.22	4.4
Conclusion	Lectures must have interactivity / excites you but there are some that dislike it.	You dislike a lot of text on slides	You expect LOs per lecture	You hate how SEM is scheduled (though some expect it that way)	You expect or prefer an optional lab (though it's not so important)	Not expected, but heavily preferred. Very important

# **Moving from Requirements to Architecture**

# **What is Architecture?**

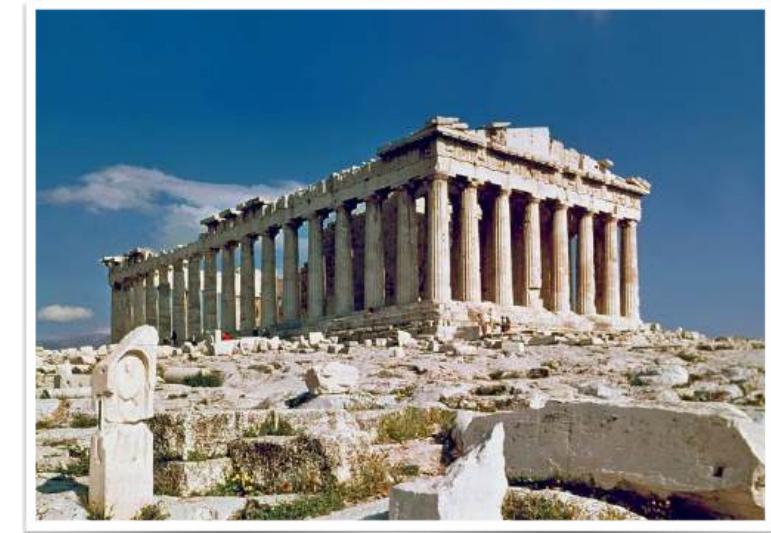
# Architecture in Civil Engineering



Step pyramid of Djoser  
(Monolith)



Milan Cathedral  
(Gotic Architecture)



Parthenon in Athens  
(Classic Architecture)

# What is Software Architecture?

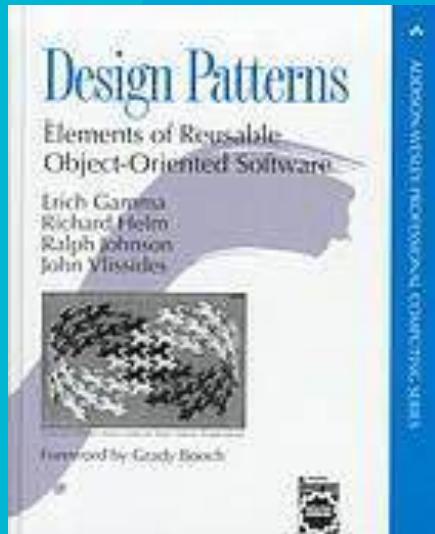
- According to you? (Vevox)
- Many different definitions

# What is Software Architecture?

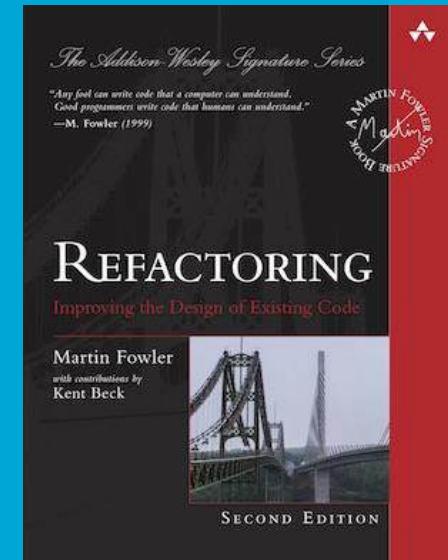
- The fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.  
- *IEEE*

# What is Software Architecture?

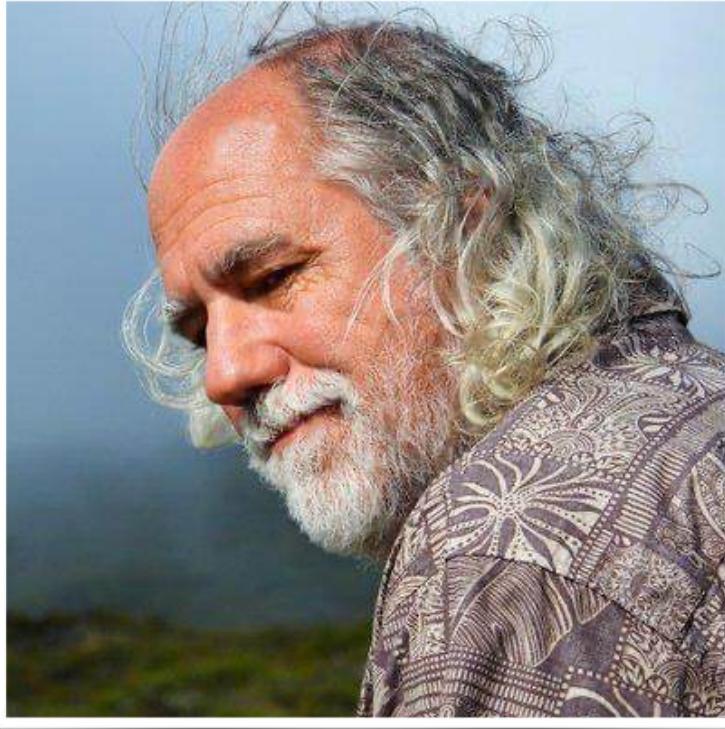
“The important stuff. Whatever that is.”  
– *Ralph Johnson and Martin Fowler*



What do we consider important?



# Software Architecture



Grady Booch  
(creator of UML)

- “All architecture is design but not all design is architecture...”
- “Architecture represents the **significant decisions** that shape a system...”
- “... where **significant** is measured by **cost of change**...”

# SA – Cost of Change

## ? Recurrent problems

1. We need to change the visibility of one method
2. We need to add more test cases
3. We need to apply a design pattern
4. We need to convert the game into a web-application with multiple, remote players



## Cost of changes

1. A few seconds
2. A few minutes/hours
3. A few hours/days
4. Weeks/Months

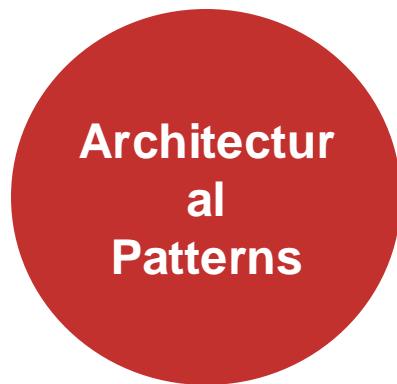


Architectural Change

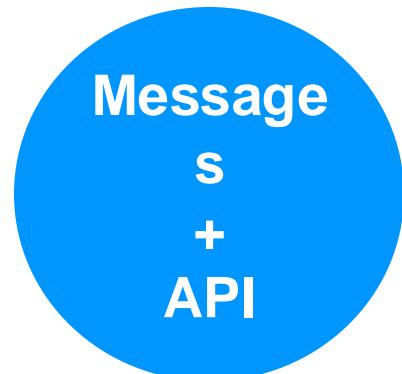
# Software Architecture

Software architecture is how the **components** of a software system are organized and assembled. How they **communicate** with each other. And the **constraints** that rule the whole system.

Software architecture has three main parts:



How components are  
organized



How components interact with  
one another



Constraints + non-  
functional Req.

# **How to look at architecture**

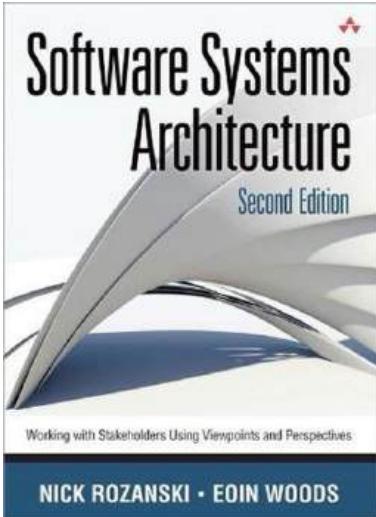
# How to show/discuss architecture?

- It depends on
  - What you want to show
  - Whom you want to show it to
  - Why you want to show it

# Architectural Views

- There are different viewpoints for looking at a system
  - E.g. different stakeholders have different interests
- You cannot consider everything at once
  - Why?

# Architectural Views



“Software Systems Architecture: Working with Stakeholders  
Using Viewpoints and Perspectives”  
Nick Rozanski and Eoin Woods, 2012 ([TU Delft Library eBook](#))

- **Architectural view:** A representation of structural aspects of an architecture that illustrates how the architecture addresses concerns held by stakeholders.

# Architectural Views

- Context View
- Functional View
- Information View
- Concurrency View
- Development View
- Deployment View
- Operational View

# Context View

- System scope and responsibilities
- External Services and their responsibilities
- External Interfaces
  - API Specifications

# Functional View

- Breaking the system down into (high-level) components
- Defining interfaces of the components
- Describing interactions between components

# Information View

- How the system stores, manipulates, manages and distributes information
- Archiving and deletion requirements (GDPR)

# Architectural Views

- Context View
  - (C4) Context Diagram
- Functional View
  - Component Diagrams, (Sequence Diagrams)
- Information View
  - Class Diagrams, ER Diagrams, Activity Diagrams
- Concurrency View
- Development View
- Deployment View
- Operational View

# **How to represent architecture**

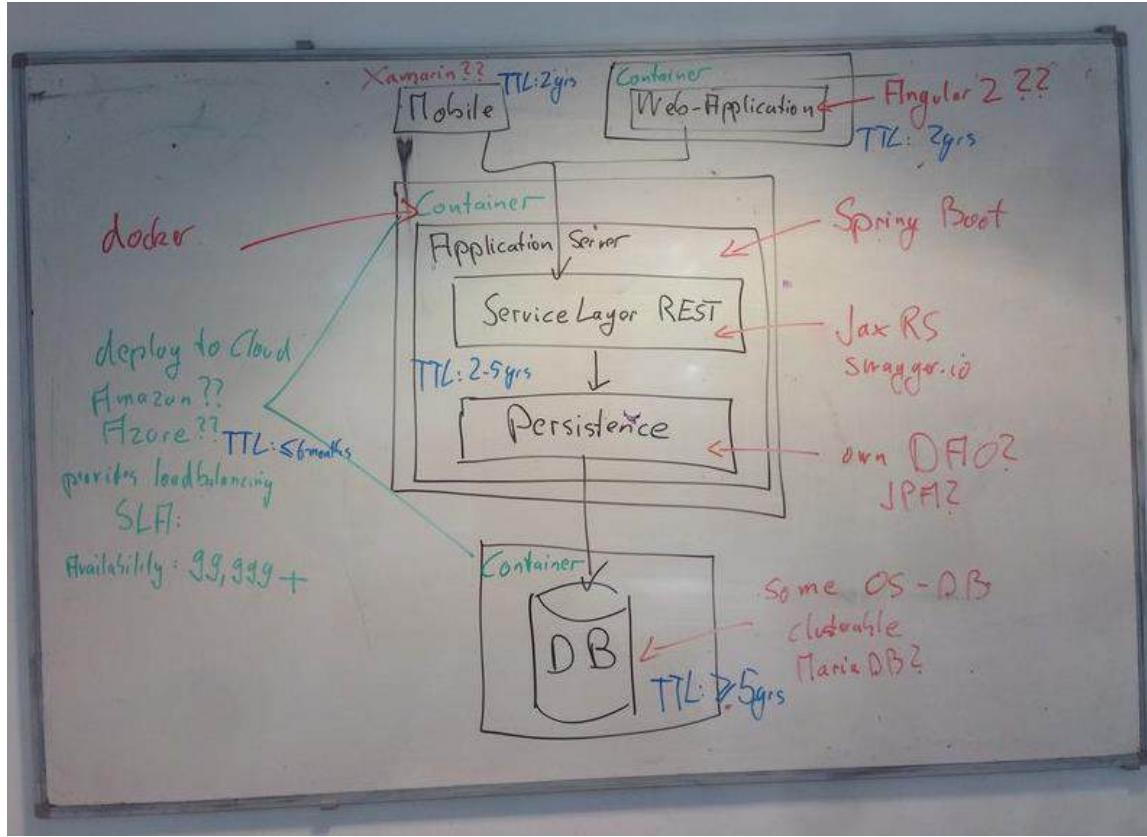
# How to represent SA?

- *Boxes-and-lines*
- Component Diagram
- Detailed Diagrams
- C4 Model

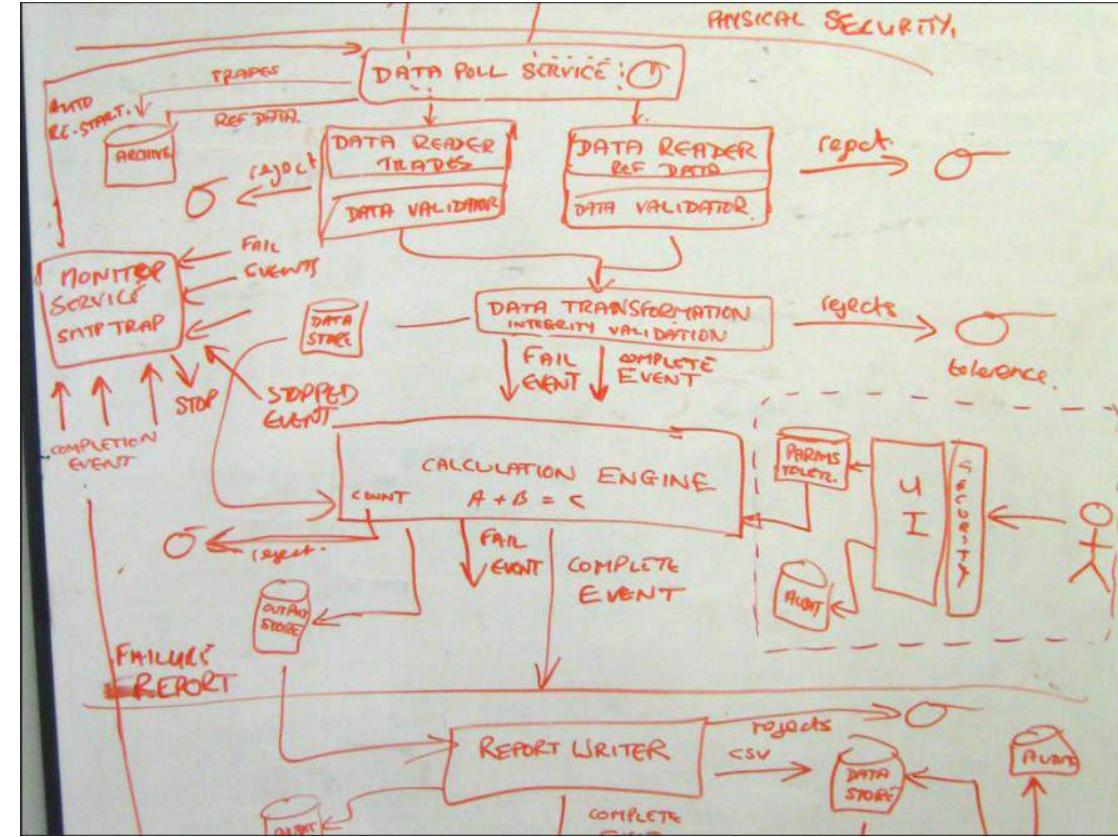
# Important concepts in SA

- Notions of separation and independence are fundamental to architectural design
  - Allow changes to be localized
- => important to break up our system into parts (components)

# Boxes-and-Lines



T. Theunissen et al,  
“Specifications in Continuous  
Software Development”, 2017



© Simon Brown, 2021,  
<https://c4model.com>

# Boxes-and-Lines

- Pros
  - Easy and quick
  - Good for a discussion
- Cons
  - Hard to understand
  - Bad for documentation (only has meaning in the moment)
  - Inconsistent notation
  - Ambiguous naming
  - Mixed levels of abstractions

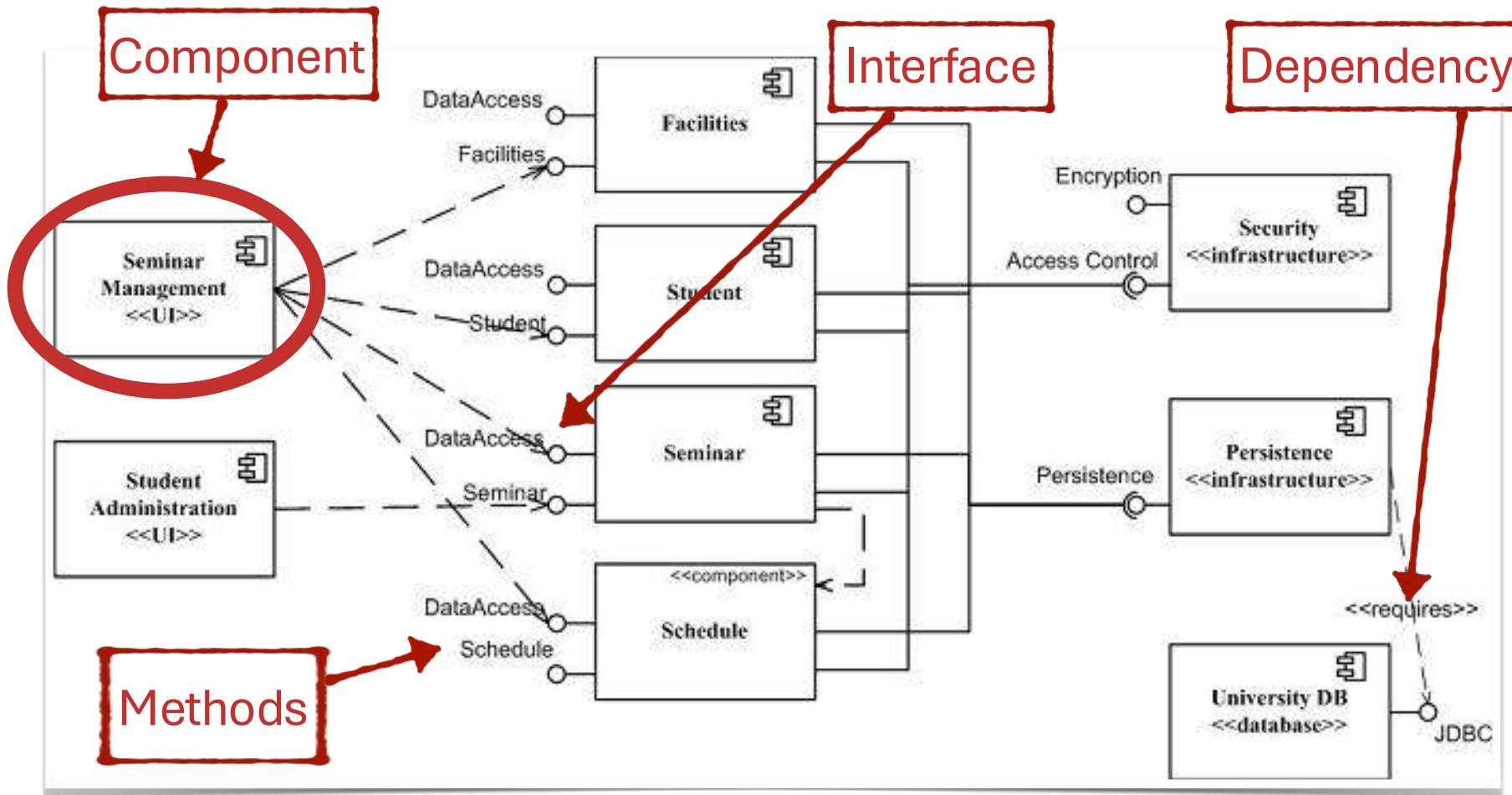
# Aside: Diagram Bingo

Horizontal and vertical "layers"	Unexplained numbers	Stacked boxes to represent scaling	Lines without arrowheads	Ambiguous abstractions
Unexplained shapes	Unlabelled boxes	Unexplained arrowheads	Writing upside-down	Unreadable handwriting
A box labelled "Error"	A clock symbol	<b>C4 model</b> for visualising software architecture	Acronyms	Different box sizes
Incomplete key/legend	No diagram title	Crossing lines	Unlabelled arrows	Unexplained line styles
Unexplained line colours	The text "..."	A box labelled "Security"	A box labelled "Business Logic"	Unexplained symbols/icons

# UML Component Diagram

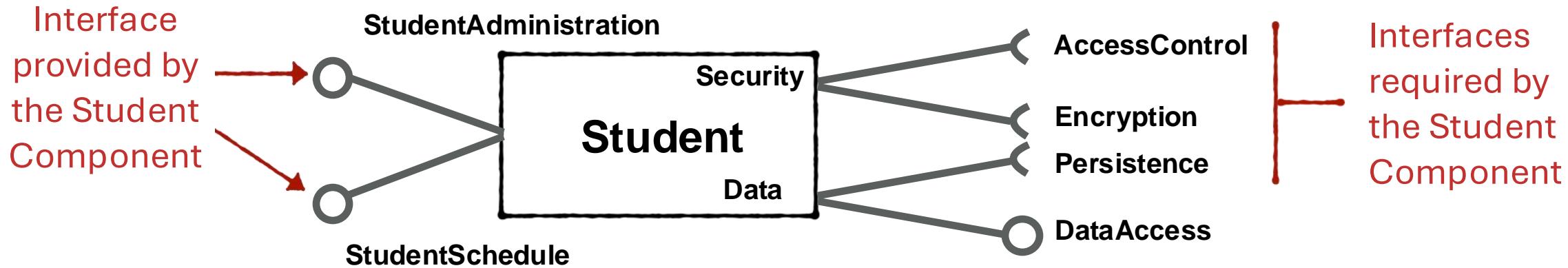
- Indicating the different components in your system
- Showing the interfaces each component exposes (by name only)
- Showing who uses these interfaces

# Component diagrams in a nutshell



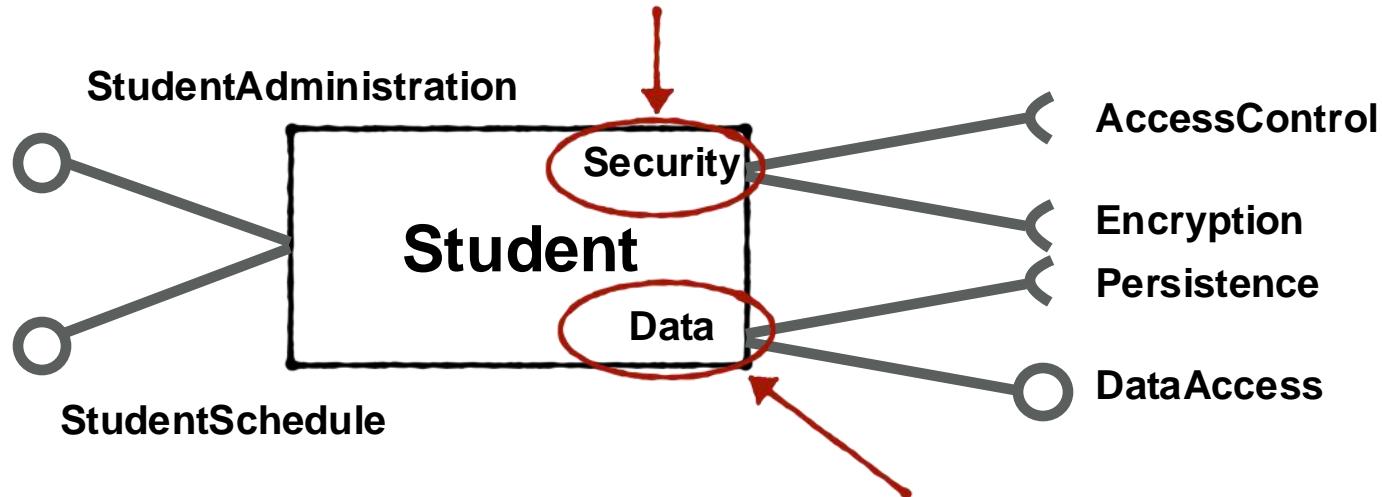
Component diagram for a simplified Seminar Management System

# Interfaces and ports



- The **lollipop notation** denotes the main interface of each component (rectangle)
  - The **sockets** (-c) denote the interfaces the component **requires (input)**
  - The **lollipops** (-o) denote the interfaces a given component **provides (output)**

# Interfaces and ports



- In UML, we can cluster balls and sockets into **ports**, which can be **named** (e.g., Security and Data in the example above)

# Guidelines for creating Component Diagrams

- **Keep components cohesive.** A component should implement a single, related set of functionality.
- **Merge a component into its only client.** If you have a component that is a server to only one other component, you may decide to combine the two components.
- **Highly coupled** classes belong in the same component.
- **Minimize the size** of the message flow between components.



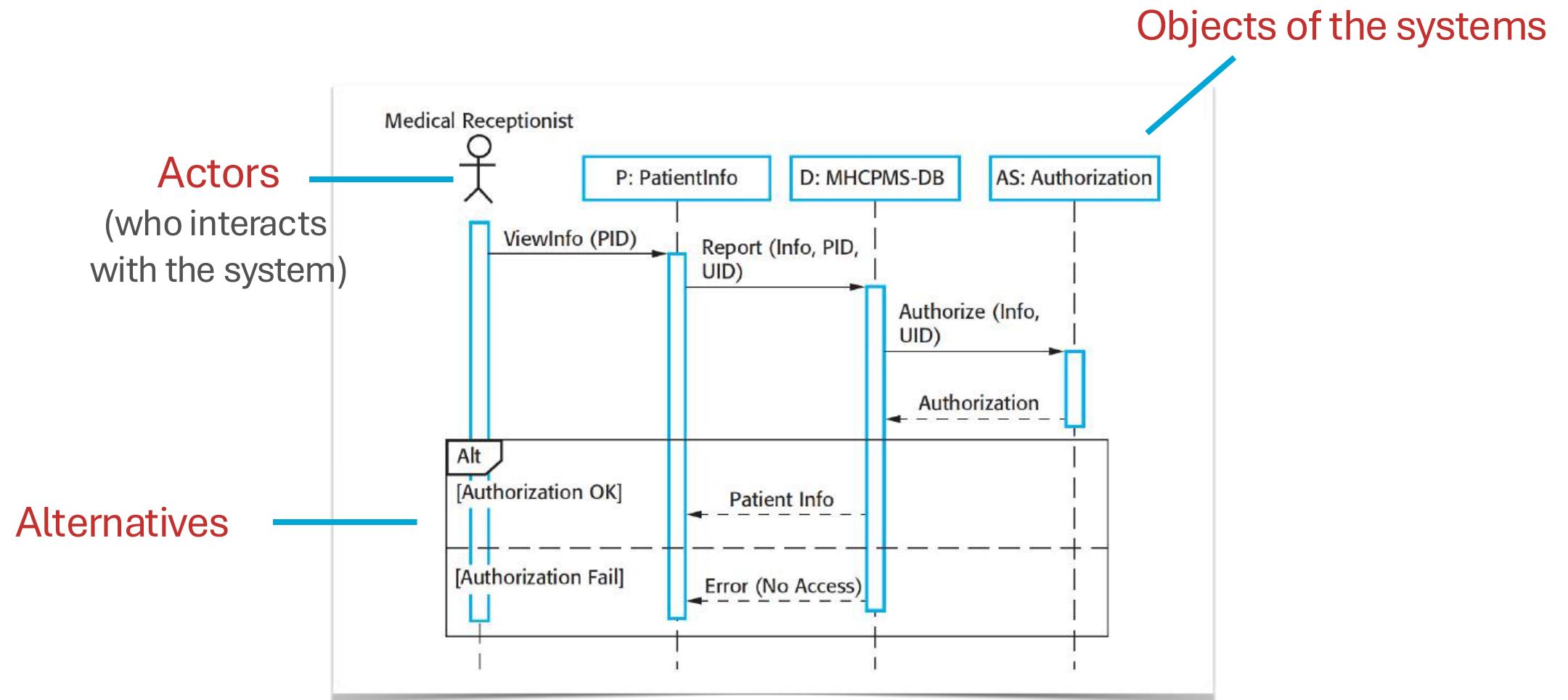
# UML Component Diagram

- Let's create a component diagram together
  - WebLab



**Should you always have a detailed  
architectural description?**

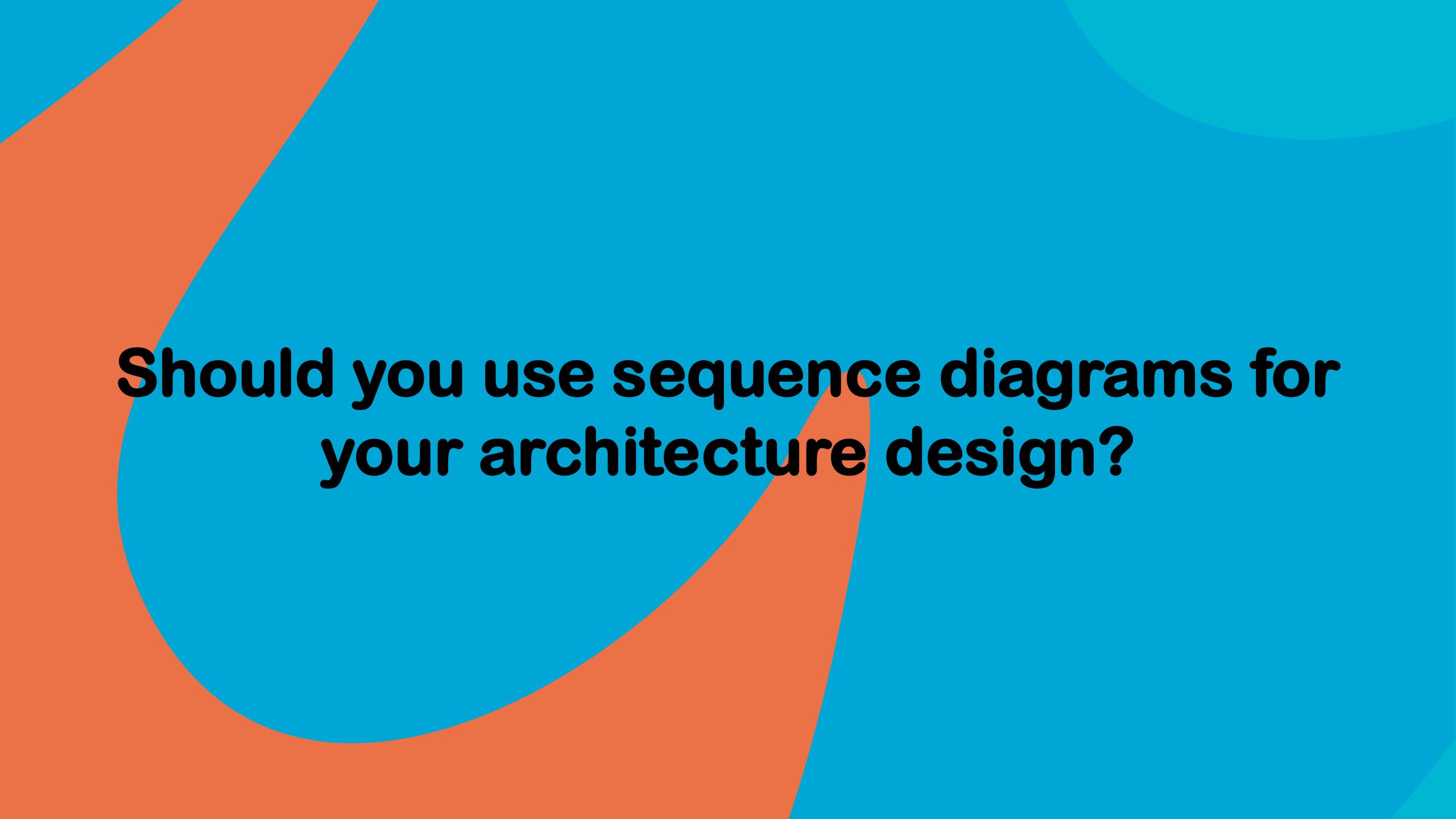
# UML Sequence Diagram



“Software Engineering”  
Ian Sommerville (Figure 5.6)

# UML Sequence Diagram

- Shows the sequence for a particular action in the system
- (Semi-)Detailed description of a particular scenario
- Highlights interactions between actors and the components of your system

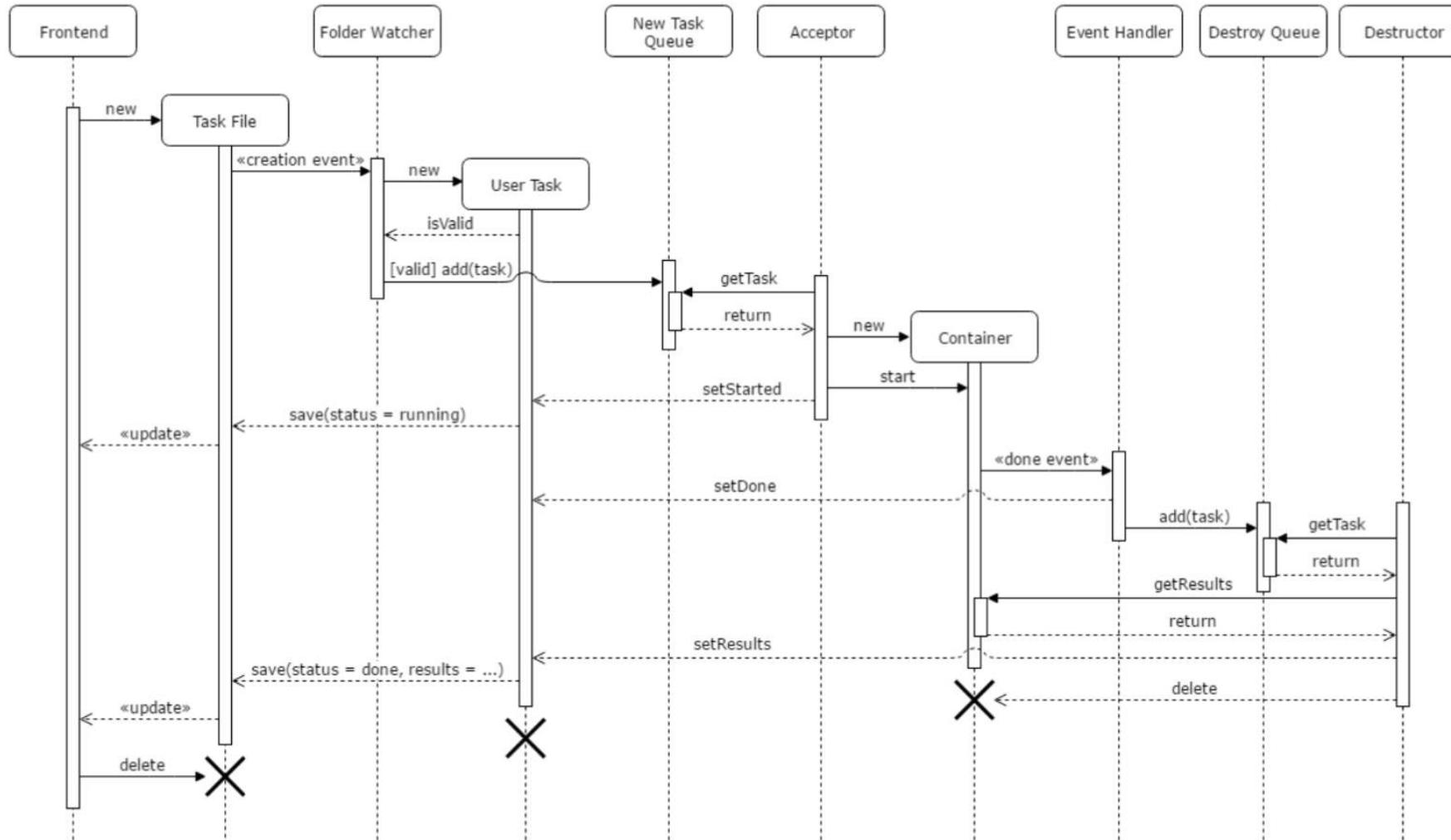


# **Should you use sequence diagrams for your architecture design?**

# UML Sequence Diagram

- Sequence Diagrams (at a high level of abstraction) can be used to support requirements engineering and high-level design
- Sequence Diagrams (at a lower level of abstraction) can be useful for documentation purposes for developers

# UML Sequence Diagram

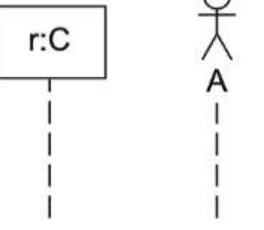
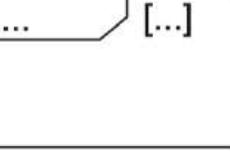


Low level or  
high level?

# How detailed should I know UML?

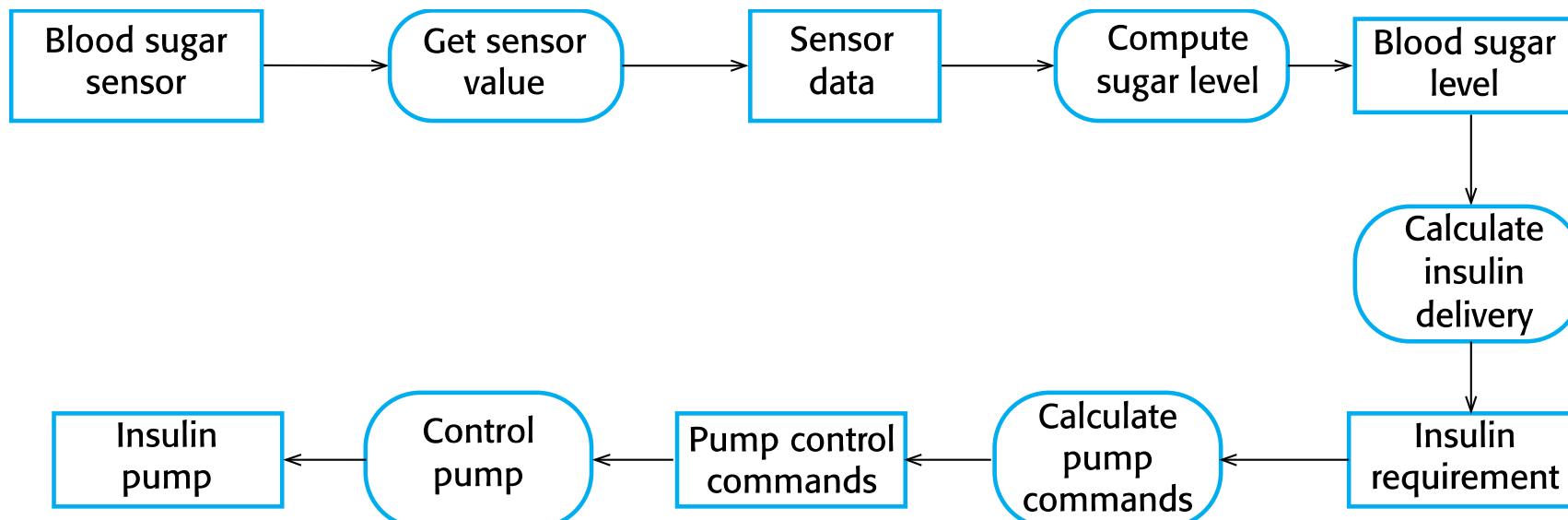
- Be able to **read and create** sequence diagrams
  - Lifeline, Destruction
  - Synchronous message, Response message
  - Combined fragments: Alt, Loop, Break
  - We will not use other elements

# How detailed should I know UML?

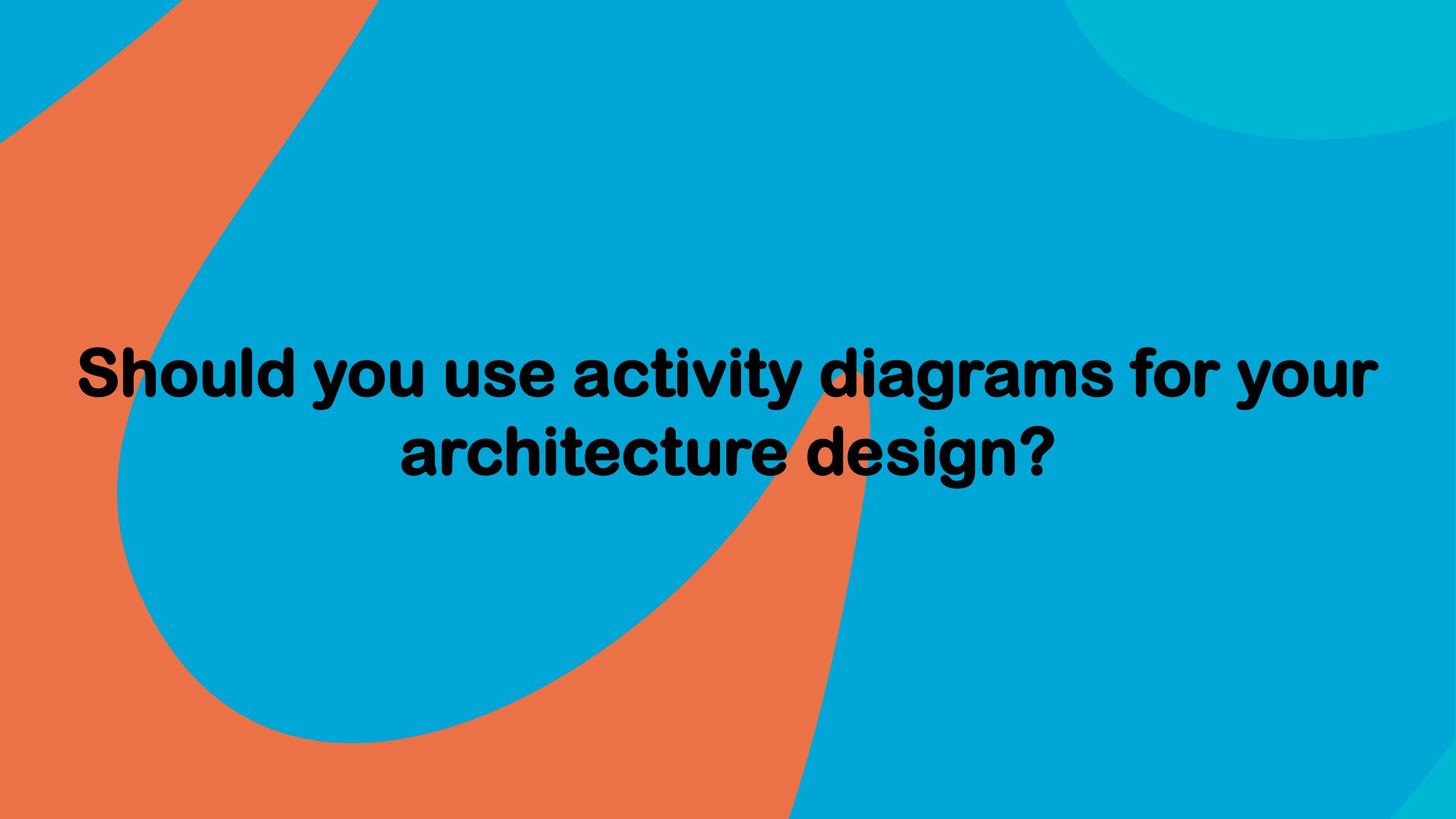
Name	Notation	Description
Lifeline		Interaction partners involved in the communication
Destruction event		Time at which an interaction partner ceases to exist
Combined fragment		Control constructs
Synchronous message		Sender waits for a response message
Response message		Response to a synchronous message

# UML Activity Diagram

- Show the activities involved in a process or data processing
- In some cases better alternative to sequence diagram



“Software  
Engineering”  
Ian Sommerville  
(Figure 5.14)

The background features a large, rounded teal shape on the left and a smaller, curved orange shape on the right, set against a white background.

# Should you use activity diagrams for your architecture design?

# UML Activity Diagram

- More useful for requirements elicitation and/or implementation (tomorrow) than for designing architecture

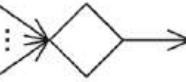
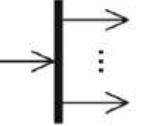
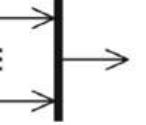
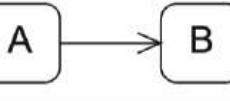
# Differences Sequence Diagram and Activity Diagram

- Sequence diagram highlights objects
- Activity diagram highlights functions

# How detailed should I know UML?

- Be able to **read** activity diagrams
  - Action, Edge, Object node
  - Initial node, Activity Final node, Decision node, Merge node, Parallelization node, Synchronization node
  - We will not use other elements

# How detailed should I know UML?

Name	Notation	Description
Action node	Action	Actions are atomic, i.e., they cannot be broken down further
Initial node	●	Start of the execution of an activity
Activity final node	○	End of ALL execution paths of an activity
Decision node		Splitting of one execution path into two or more alternative execution paths
Merge node		Merging of two or more alternative execution paths into one execution path
Parallelization node		Splitting of one execution path into two or more concurrent execution paths
Synchronization node		Merging of two or more concurrent execution paths into one execution path
Edge		Connection between the nodes of an activity
Object node	Object	Contains data and objects that are created, changed, and read

# UML Class Diagram

- Graphical notation to design and visualize classes in OO systems
- Components
  - Classes
  - Attributes
  - Methods
  - Relationships
- Focus on behavior

More on class diagrams tomorrow

# C4 Model

# C4 Model

- Why?
  - Understanding UML Diagrams requires understanding UML
  - Most things you want to explain don't need most of UML
  - Boxes-and-Lines are commonly used, but make it seem that many software dev teams have lost the ability to communicate visually. (Their words, not ours)

# C4 Model

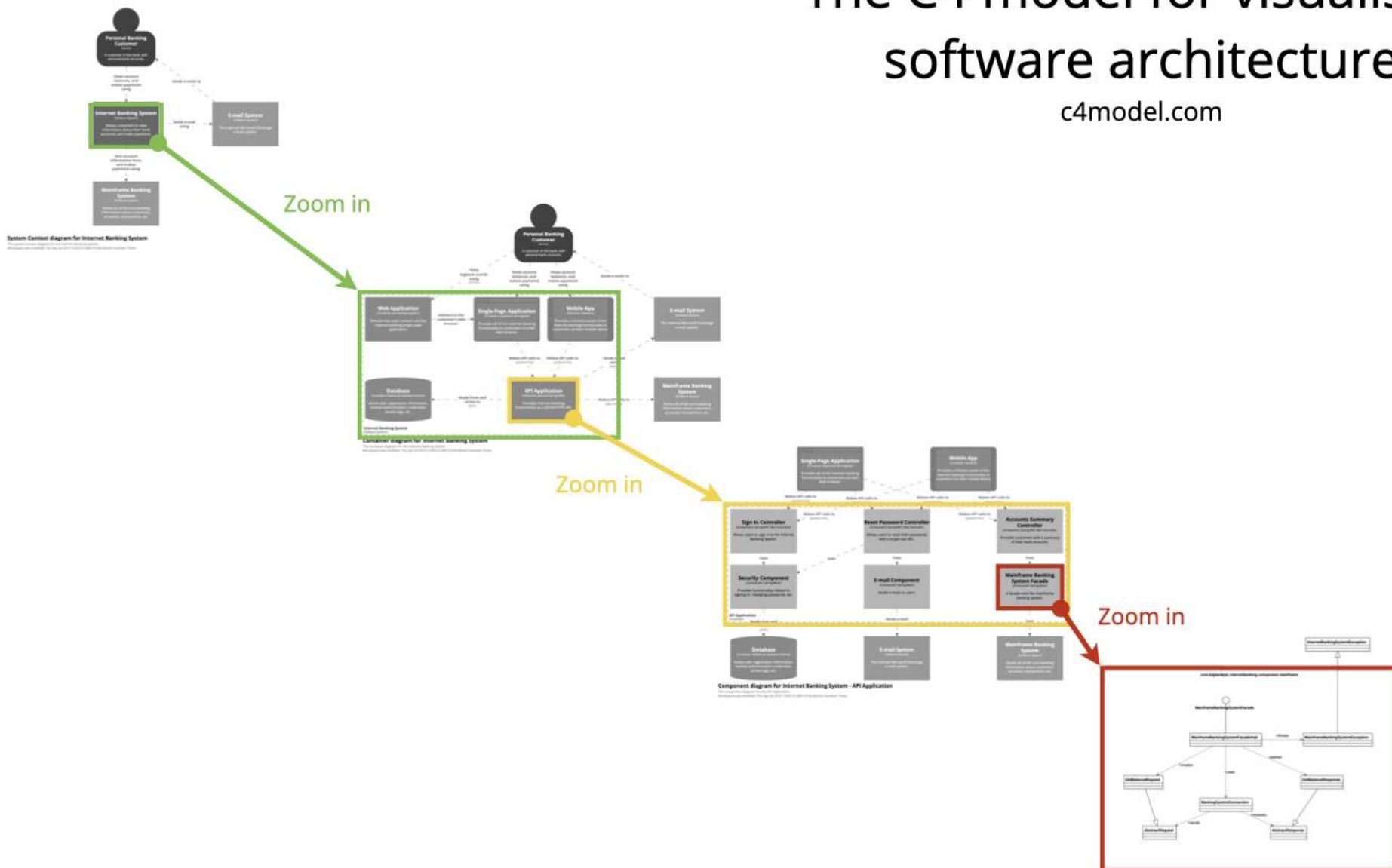
- General idea
  - Maps at different zoom levels
  - More text
  - Legend / Key / Guide of used symbols/lines/...
- Basically
  - Set of good practices for making diagrams of software systems
  - Recommendations for which diagrams to draw



Legenda	
Land	
Bebouwd	
Gebouw	
Plein	
Water	
Hoofdweg	
Weg	
Bezienswaardigheid	
Museum	
Parkeerplaats	
Boot	
Pier	
Ziekenhuis	
Taxistandplaats	
Bushalte	

# The C4 model for visualising software architecture

c4model.com



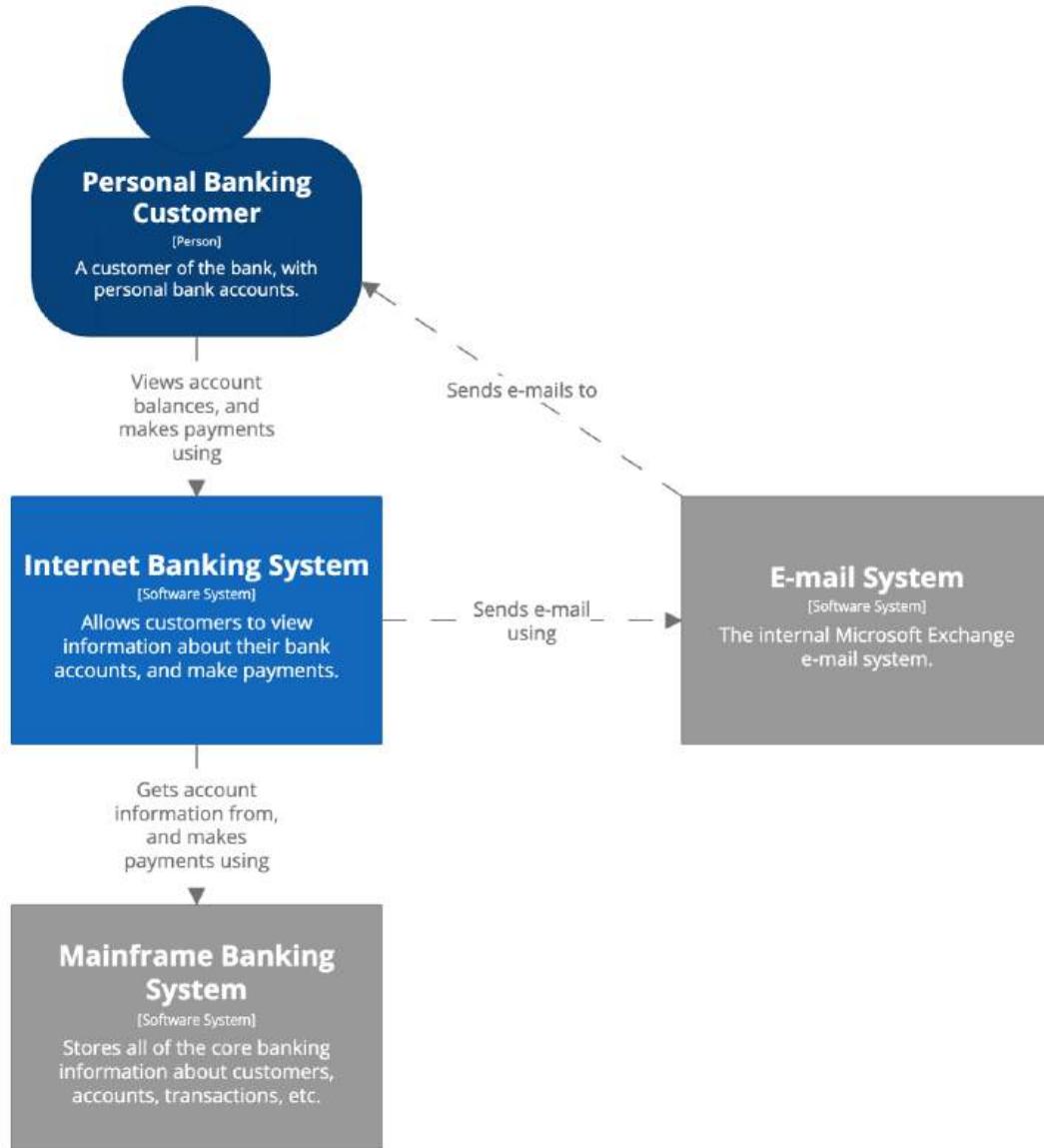
Level 1  
Context

Level 2  
Containers

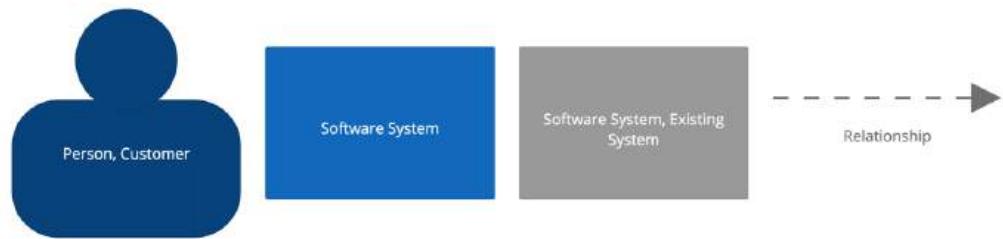
Level 3  
Components

Level 4  
Code

# C4 – (1) Context Diagram

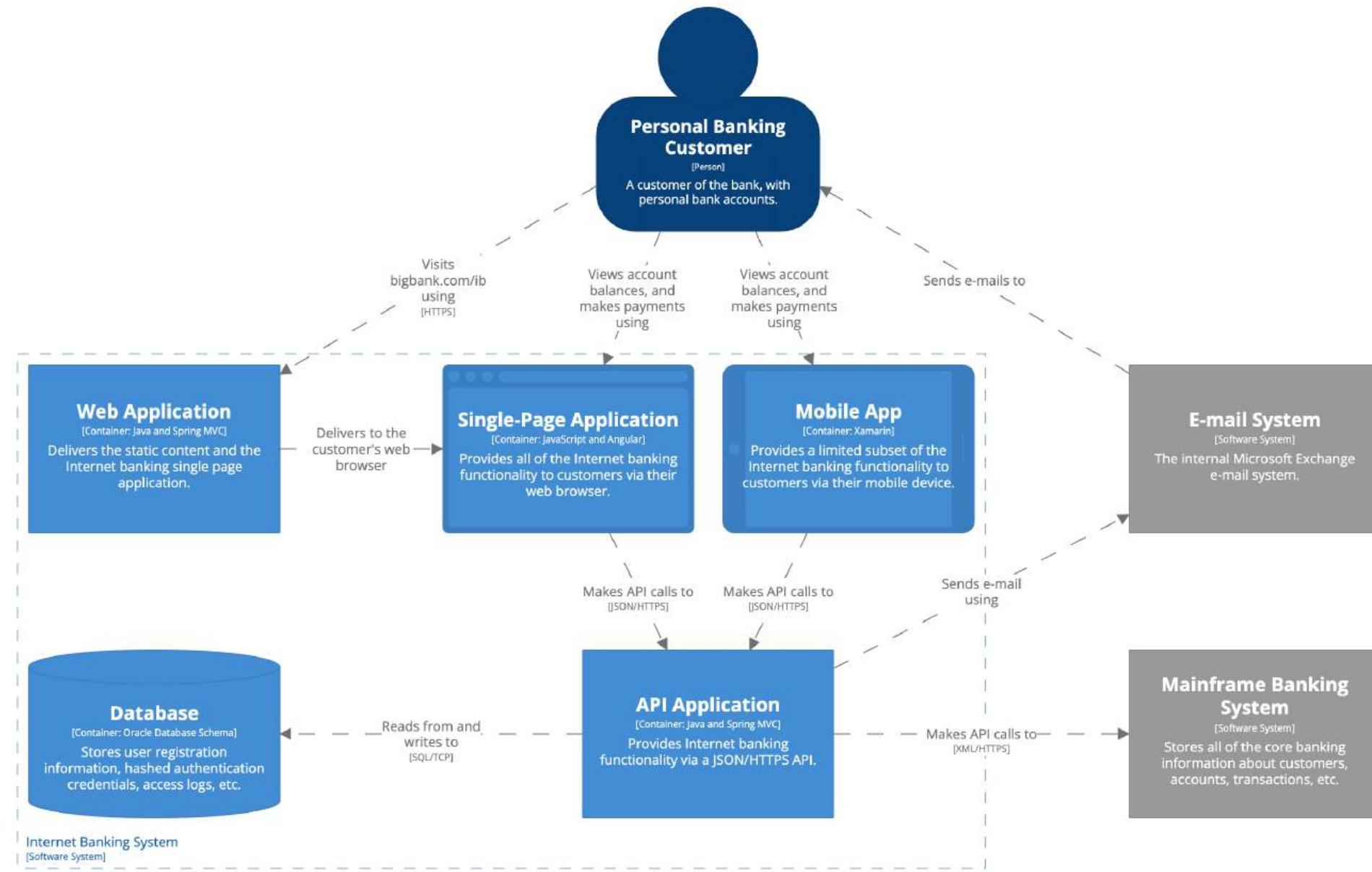


- Your system in center
- Surround with users and other systems it interacts with
- Focus on big-picture



All images on the C4 Model are from  
<https://c4model.com>

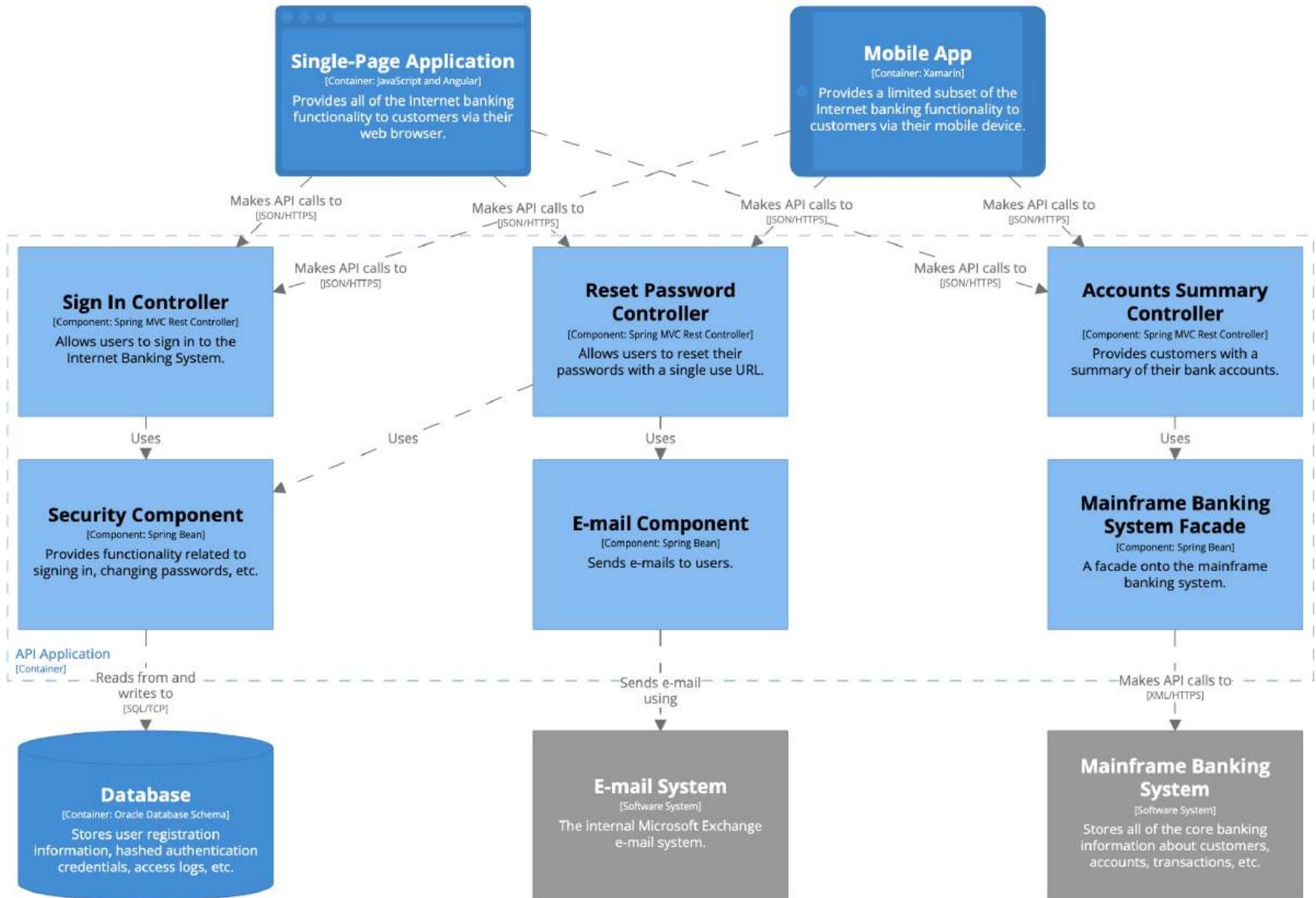
# C4 – (2) Container Diagram



# C4 – (2) Container Diagram

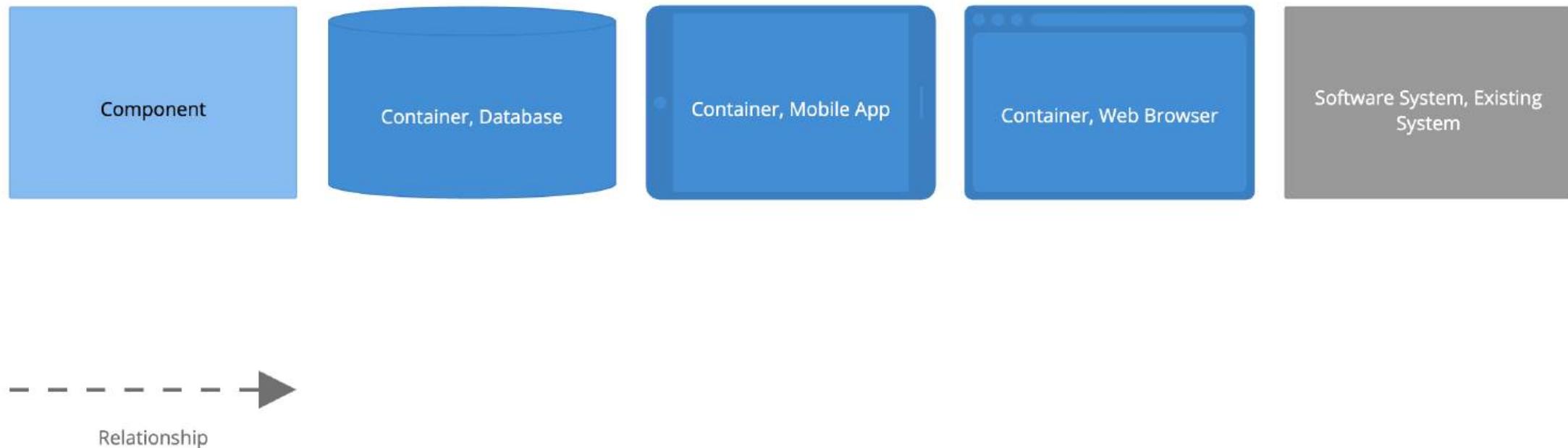
- Container (has nothing to do with Docker here) is e.g. a server-side web app, single-page app, desktop app, mobile app, database schema, file system, etc.
- Container is a separately runnable/deployable unit that executes code or stores data.
- Diagram shows high-level shape of the software architecture and how responsibilities are distributed

# C4 – (3) Component Diagram



# C4 – (3) Component Diagram

- Like the UML component diagram
- Intended for Software architects and devs



# C4 – (4) Code diagram

- Use a UML Class Diagram or Entity Relationship Diagram (CSE1505 IDM)
- Stated as optional
- Auto-generate using tooling

# C4 – Further Reading

- Have a look at the website

<https://c4model.com>

# General Takeaways

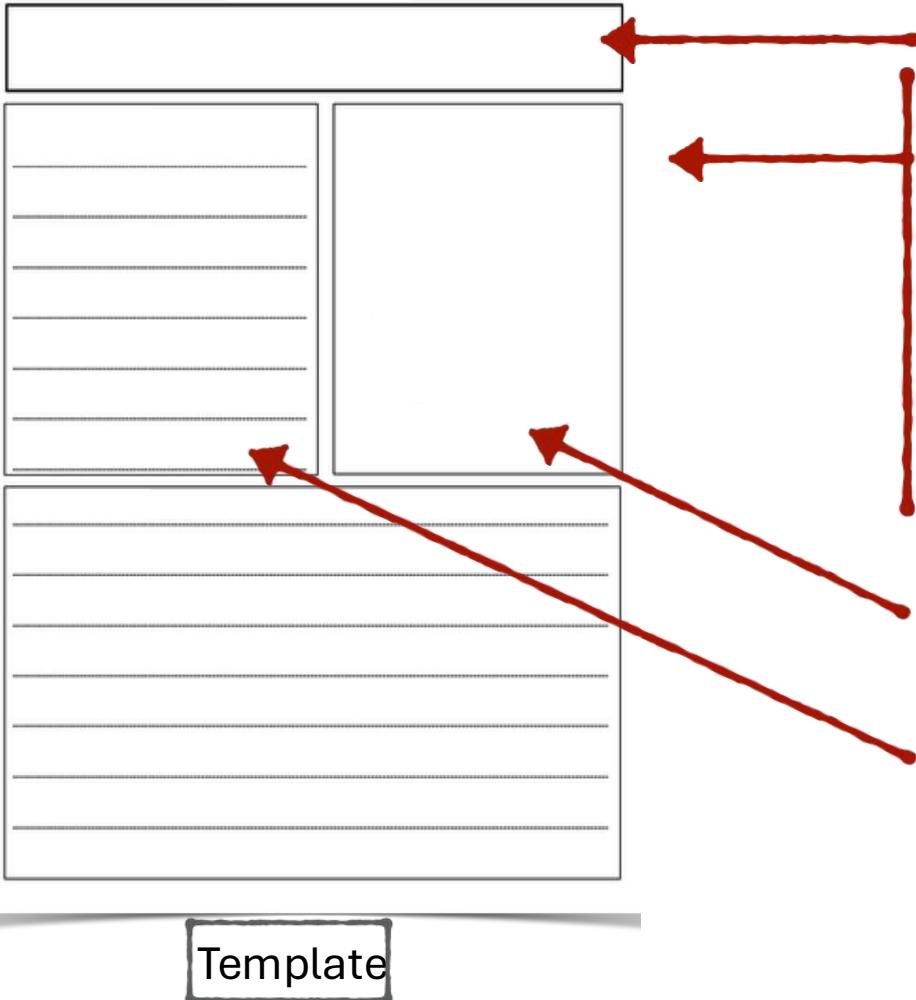
- Different diagrams are useful for different situations
- Larger scope / more critical software? → more diagrams
- Primary point is effective communication, more important than precise way how you do it
- For the exam you should be able to read and create all the covered types of diagrams

# Architectural Patterns

# Architectural Pattern

- General, reusable solution to a commonly occurring problem in SA within a given context.
- Describes overall structure of (a subcomponent of) the system.

# Architectural Patterns



- Name or pattern identifier
- Context: a recurring, common situation that gives rise to a problem
- Problem: The pattern descriptions usually outline the problem and its variants. The description of the problem often influences the quality attribute to meet (e.g., performance)
- Topological layout of the components
- Ingredients of the solution: element types, interaction mechanism, semantic constraints.
- Solution: an architectural resolution to the problem. The solution describes the architectural structures that solve the problem, including how to trade-off the conflicting factors (e.g., performance vs. testability)

# Architectural Patterns

- Different patterns address different quality attributes.
- Not all attributes can be addressed by one single pattern
  - Layered → Maintainability (tradeoff performance)
  - Peer-to-peer → Performance (tradeoff data integrity)
  - Pipe-and-filter → reusability (tradeoff performance)
- Choosing an architectural pattern over another is about making trade-offs between different quality attributes.

# Quality attributes

- Performance: How fast is the system response?
- Usability: How easy-to-learn is the system?
- Maintainability: How easy is to make changes?
- Testability: How easily can we verify that the system works correctly?
- Reusability: Can we reuse software components?
- Availability: Is the system always available when needed?
- Integrity: Can unauthorised users change the data?

# Quality attributes

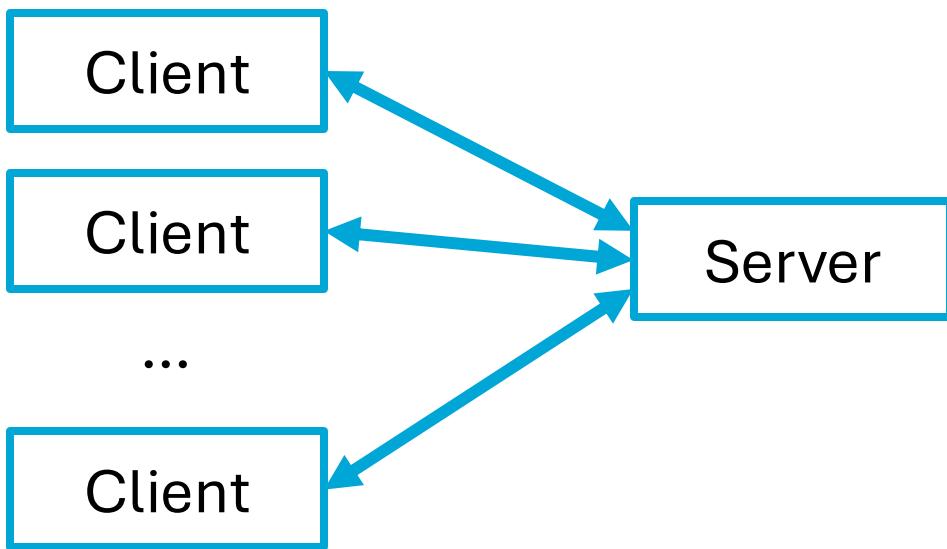
- Interoperability: Is the system well-connected to other systems?
- Scalability: How many users can we handle?
- Robustness: How does our system respond to unexpected conditions?
- Reliability: How frequent are the failures (if any)?
- Recoverability: Does the system recover from failure quickly?
- ...

# Architectural Patterns

- Client and Server
- Layered Architecture
- Model-View-Controller (MVC)
- Pipe-and-Filter Pattern
- Microservices
- Publish-Subscribe Pattern
- Event-Driven Architecture

# Client and Server

- Server hosts, delivers and manages most resources and services to be consumed by the client. Multiple client computers connect to a central server over a network (e.g. the internet).



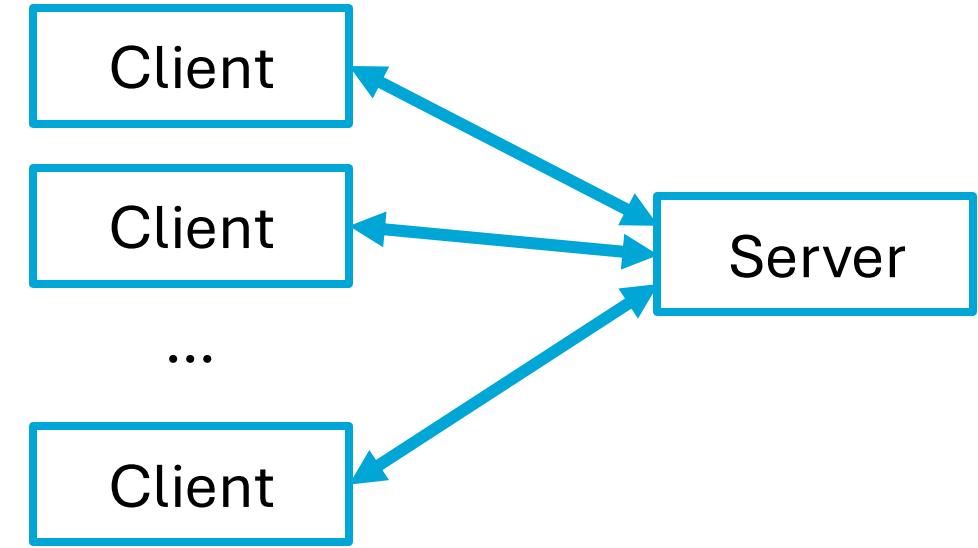
The server does not know the number or identities of clients

Clients know the server's identity

Connectors are RPC-based network interaction protocols  
(RPC = Remote Procedure Call)

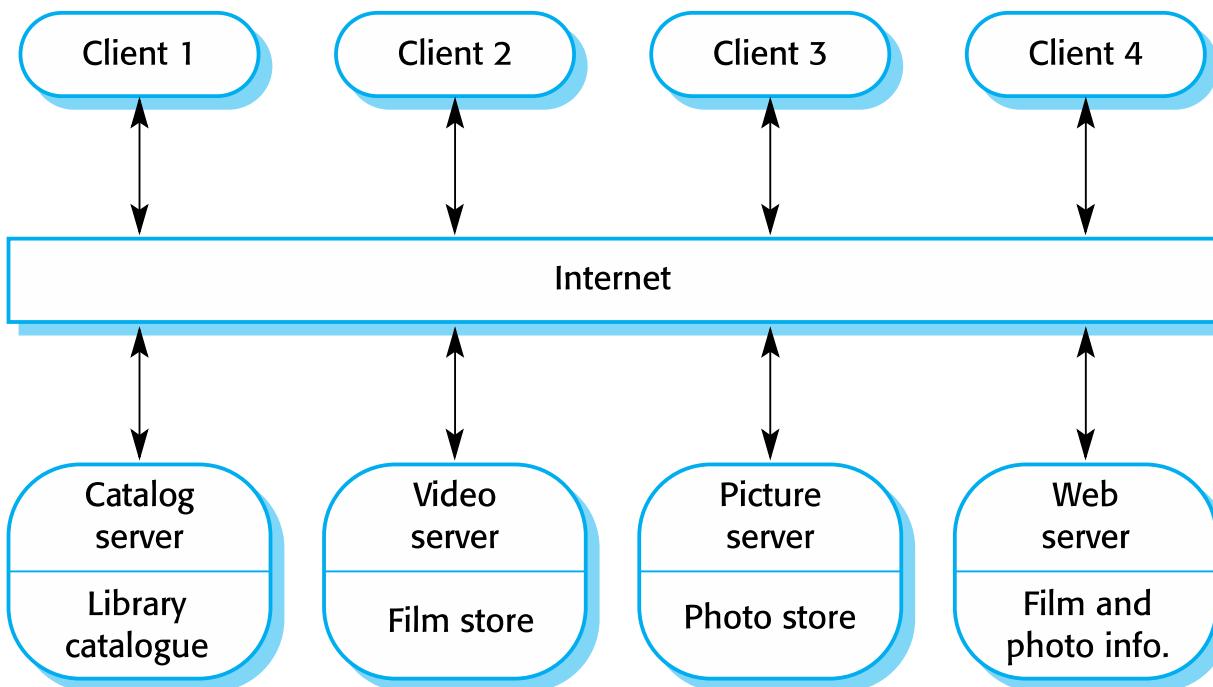
# Client and Server

- Client and Server are different processes with well-defined interface
- Client and Server have different roles
  - Server provides the service; it has the business logic
  - Client (usually) cannot be fully trusted



# Client and Server

- Systems can consist of multiple servers for different tasks



## Example: Film service

- One server serves the general site with information of how to load other elements
- Another server provides thumbnails
- Another server provides the video streaming

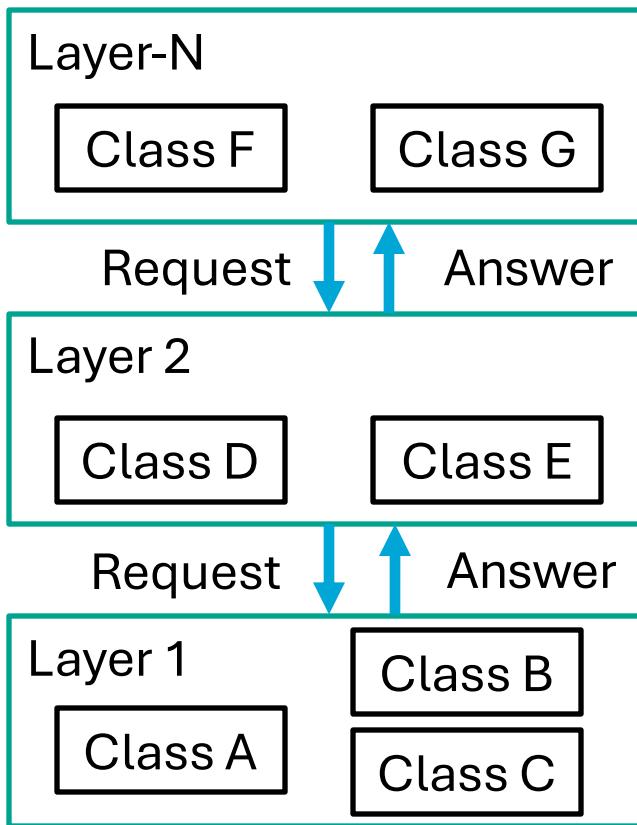
Sommerville, “Software Engineering”, Figure 6.11

# Client and Server

- Advantages
  - Scalability: often easy to scale horizontally by deploying more servers (with load balancers)
  - Reusability: General functionality can be available to all clients and does not need to be implemented by all services
- Disadvantages
  - Reliability: Each service is a single point of failure
  - Performance: may be unpredictable (network influence)

# Layered Architecture

This pattern is used for systems that can be decomposed into groups of layers, each of which is at a particular level of abstraction. Each layer provides services to the next higher layer and consumes services from the lower layer (least knowledge principle).

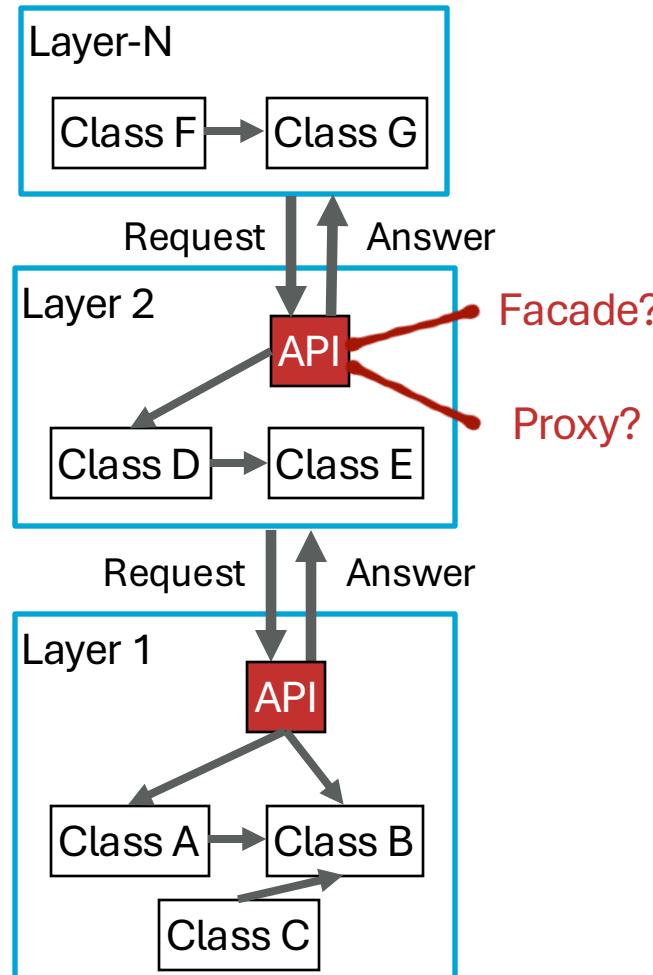


Each component exposes an interface to be used by the layer above

Each component sees the layer below (in the hierarchy) as one single opaque layer (virtual machine)

For example, Layer 2 provides an API for Layer 1 and it uses the API of the layer 2

# Layered Architecture



Every module (e.g., class or package) must be allocated to one and only one layer.

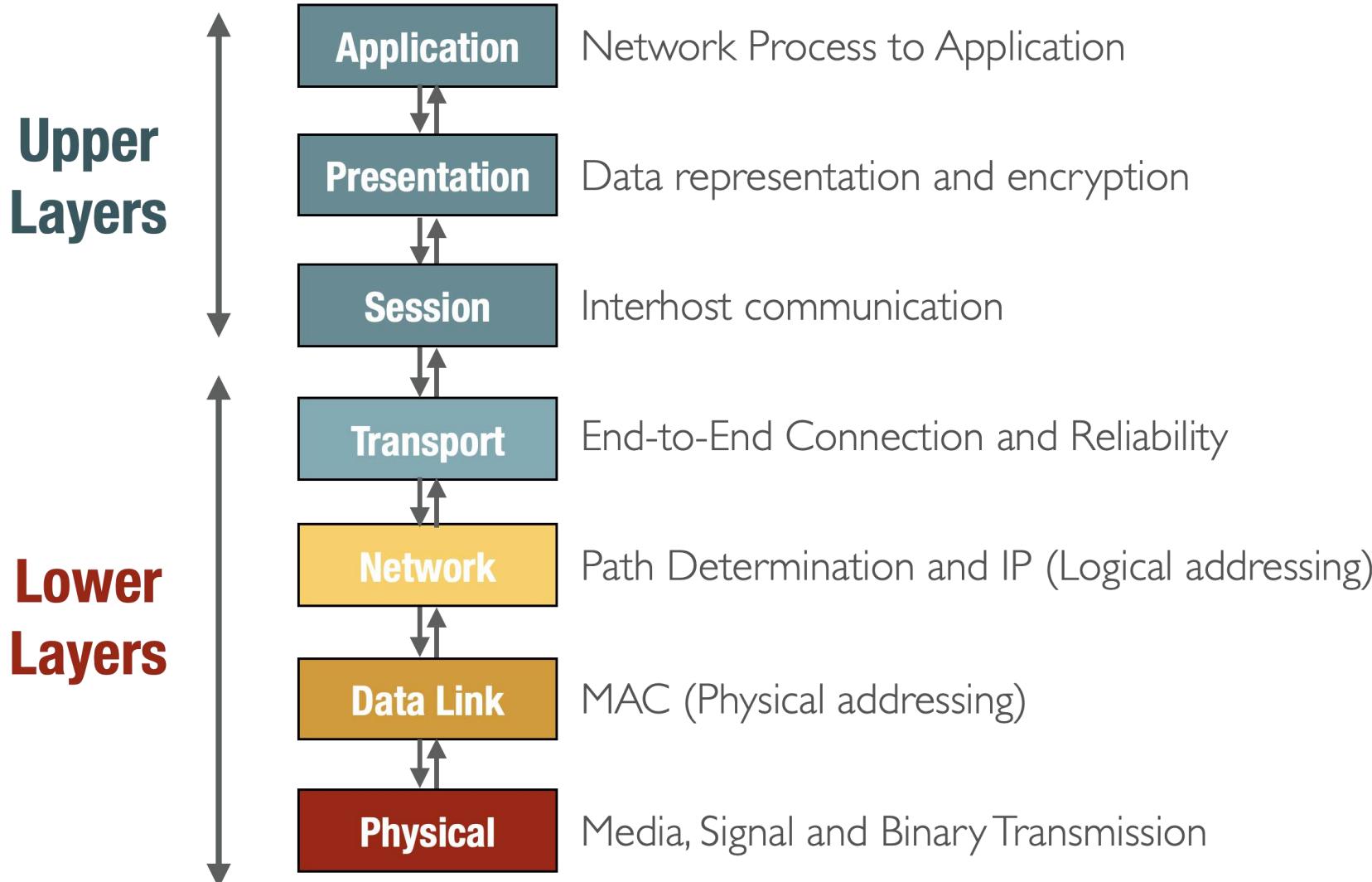
The **API** to the adjacent (higher) layer can be implemented using the Facade or the Proxy design patterns.

Using a **Proxy** or a **Facade** depends on the context (do we need to simplify the interface?)

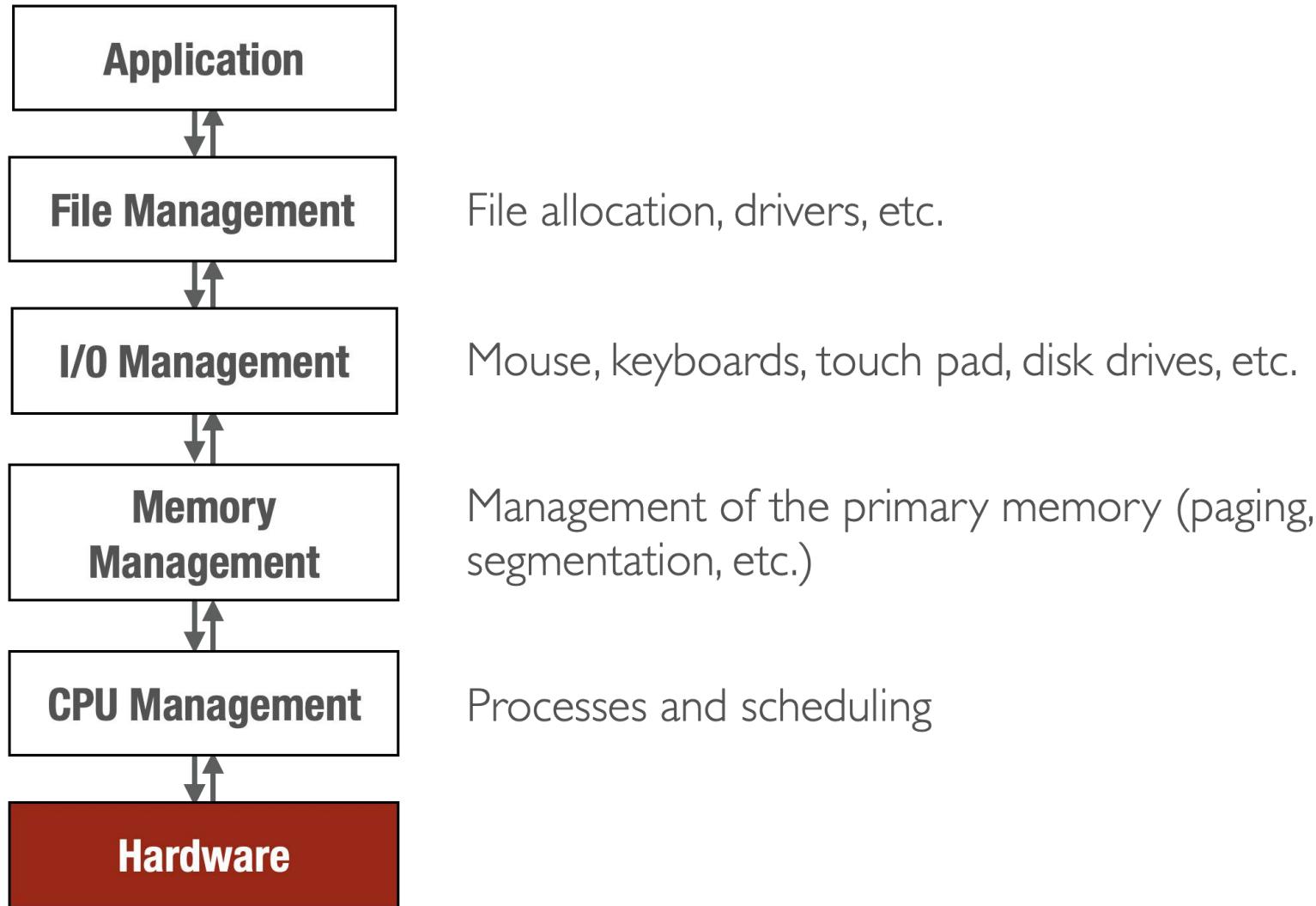
If one of the layer is a legacy system, we could also use the **Adapter pattern** to manage the APIs

Proxy, Facade, and Adapter are design patterns! We will see them in future lectures

# Example: Computer Network



# Example: Operating System



# Layered Architecture – Three-tier

- Usually 3 or 4 layers (any number is allowed)
  - Example: separate GUI from Logic

**Presentation Layer**

HTML page, GUI, user interface, ...

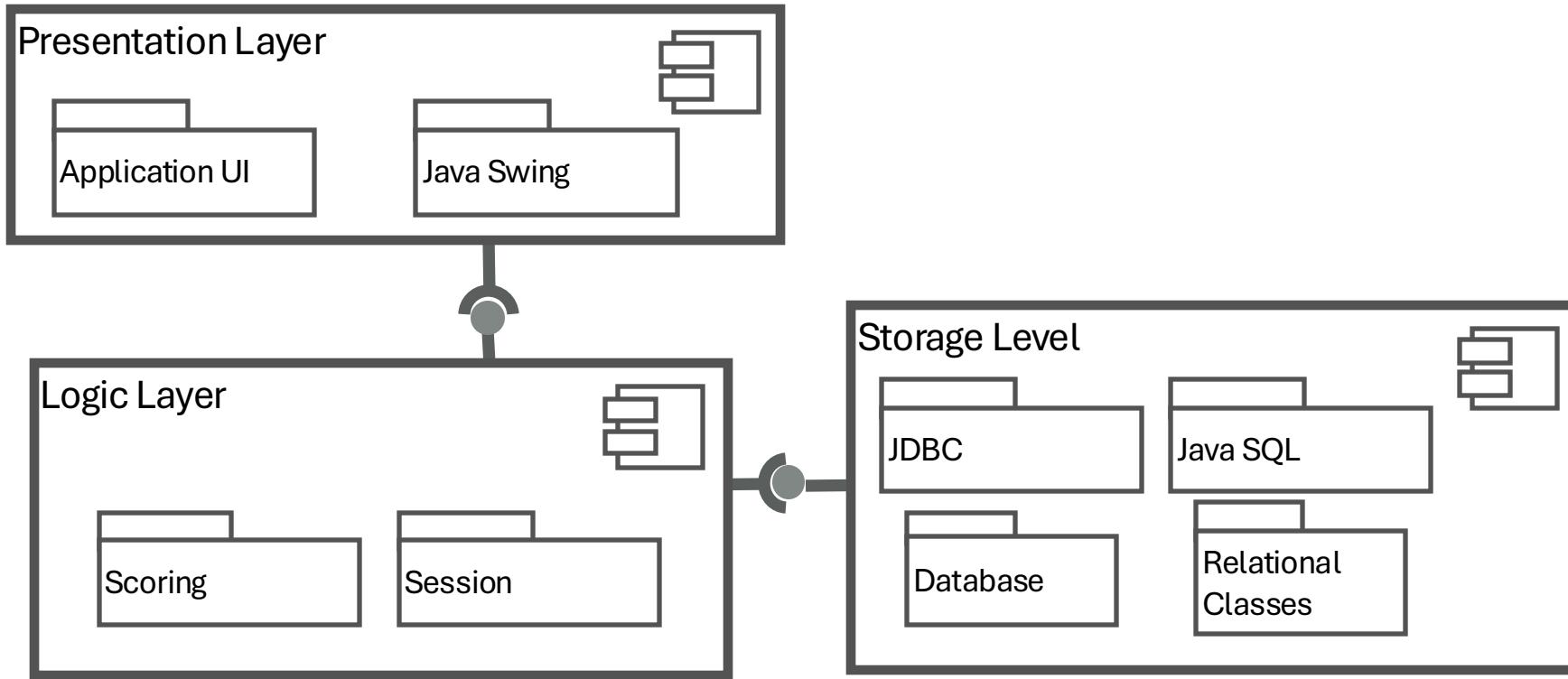
**Business Layer**

Logic that is the real point of the system

**Storage Layer**

Communication with databases, messaging systems, transaction managers, other packages

# Example in UML



**What are the main advantages (Quality Attributes) of the Layered Architecture?**

# Layered Architecture

- Advantages
  - Testability: we can test each layer in isolation (using mocks and stubs)
  - Maintainability: a change in the behaviour of a layer has no effect on the layers below it
  - Flexibility: we can exchange one layer (e.g., transport layer) with any alternative layer as long as the new and old layers implement the same interface
- Disadvantages
  - Higher complexity: additional abstraction increases the overall complexity
  - Lower performance: one request may pass through all layers in a chain of responsibility to be properly handled by the system

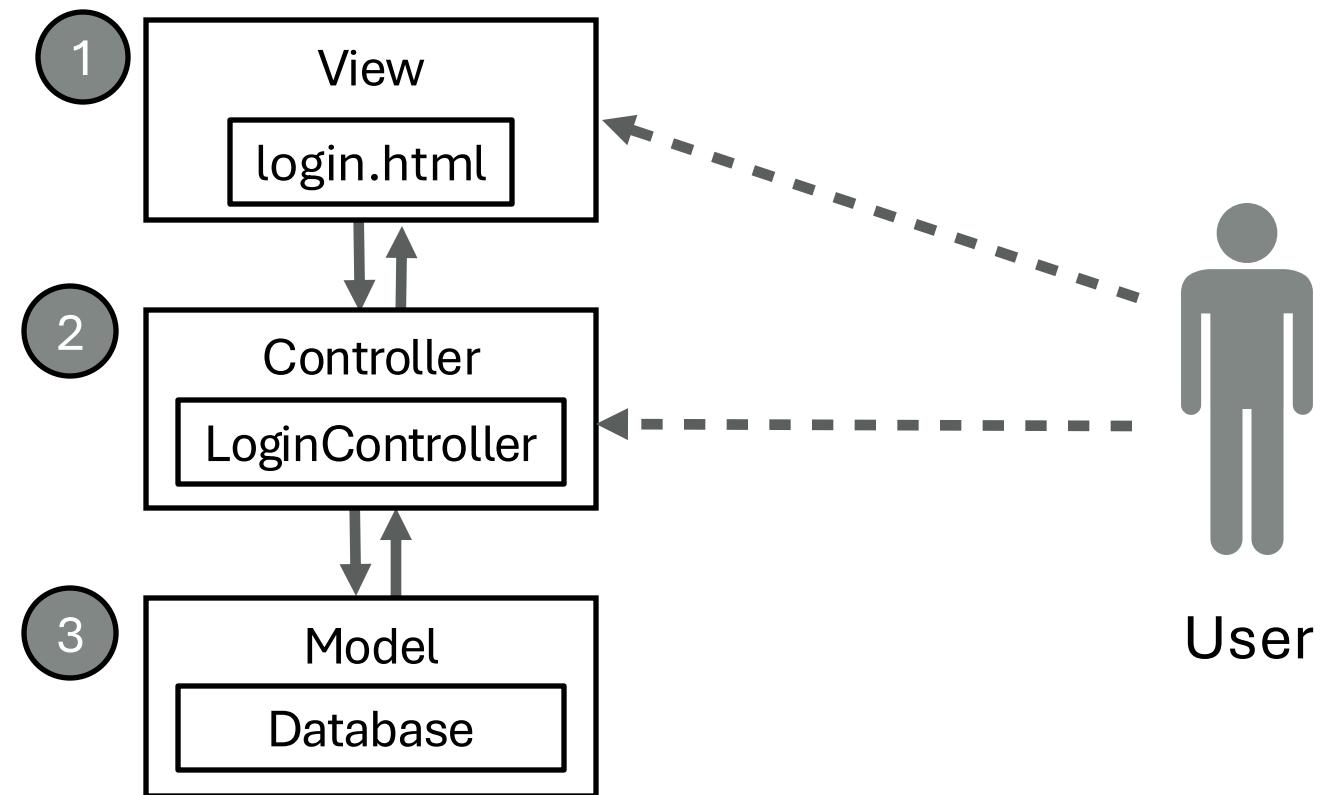
# Model-View-Controller (MVC)

- MVC is basically a three-layered architecture
- Commonly used in web development

The user presses a button in the GUI (e.g., HTML)

The control receives the request and forwards it to the model (e.g., DB)

The model verifies the request (e.g., querying the DB)



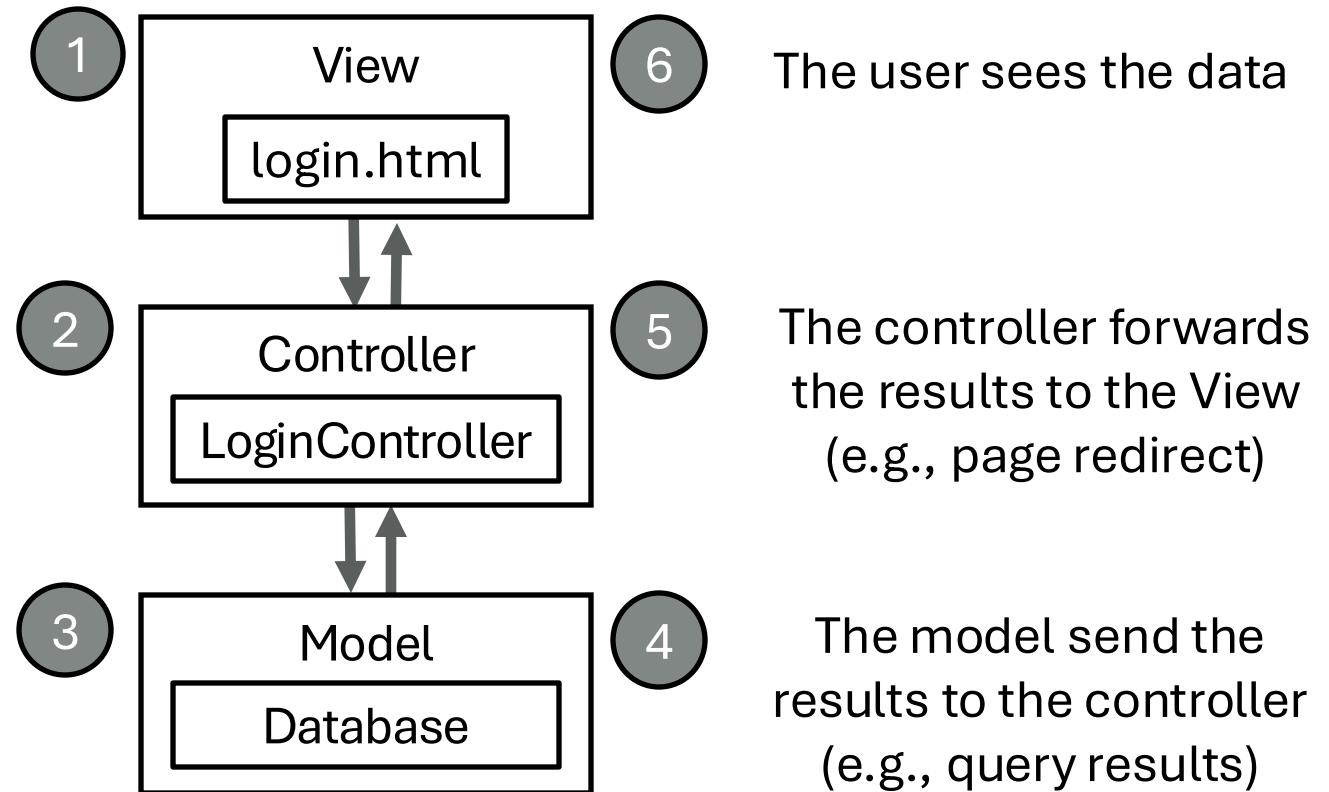
# Model-View-Controller (MVC)

- MVC is basically a three-layered architecture
- Commonly used in web development

The user presses a button in the GUI (e.g., HTML)

The control receives the request and forwards it to the model (e.g., DB)

The model verifies the request (e.g., querying the DB)

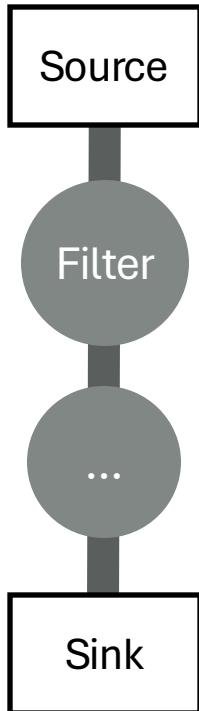


# Model-View-Controller (MVC)

- Advantages
  - Maintainability: you can make changes to one component without affecting the others (e.g. data can change independent of representation)
  - Reusability: components can be easily reused (e.g. same view for different models)
  - Flexibility: different representations of the same data
  - Testability: possible to test each layer separately (with mocks and stubs)
- Disadvantages
  - Complexity: Additional code and complexity, especially when model and interactions are simple
  - Maintainability: Often easy to “escape” from MVC and do things in wrong layers (e.g. views triggering database queries directly)

# Pipe-and-Filter Pattern

The Pipe-Filter pattern provides a structure for systems that produce a **stream of data**. Each processing step is encapsulated in a filter component while data is passed through pipes.

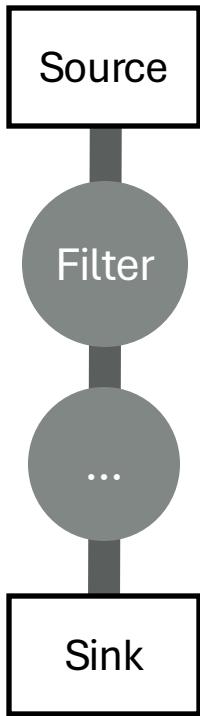


**Pipe:** A connector, which transports data from one filter to the next, preserving the order

**Filter:** a components that reads data, transforms it, and return the transformed data

The steps are connected by the data flow through the system: the output of one step is the input for the next one

# Pipe-and-Filter Pattern



**Filters** must define expected input and produced output (proper API and documentation needed)

**Pipes** have a single input and output, and do not alter the data in transit

Some variants of this pattern also include the **source** (where the data comes from) and the **sink** (the component receiving the final data)

This pattern is often used for **data intensive** applications, buffering or for synchronisation.

# Example: Machine Learning Pipeline

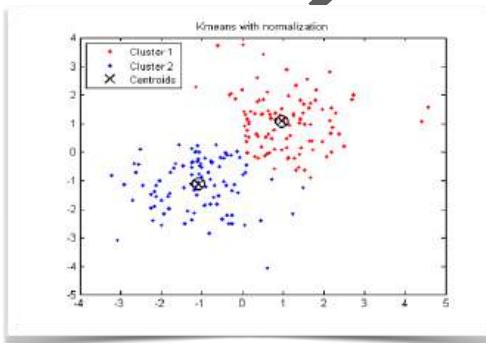


Data Pre-processing

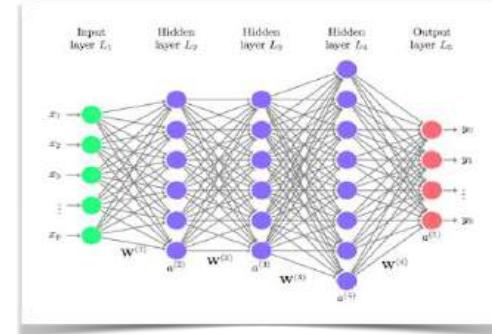
ML Training

Decision Making

Sink



Data normalisation  
and scaling

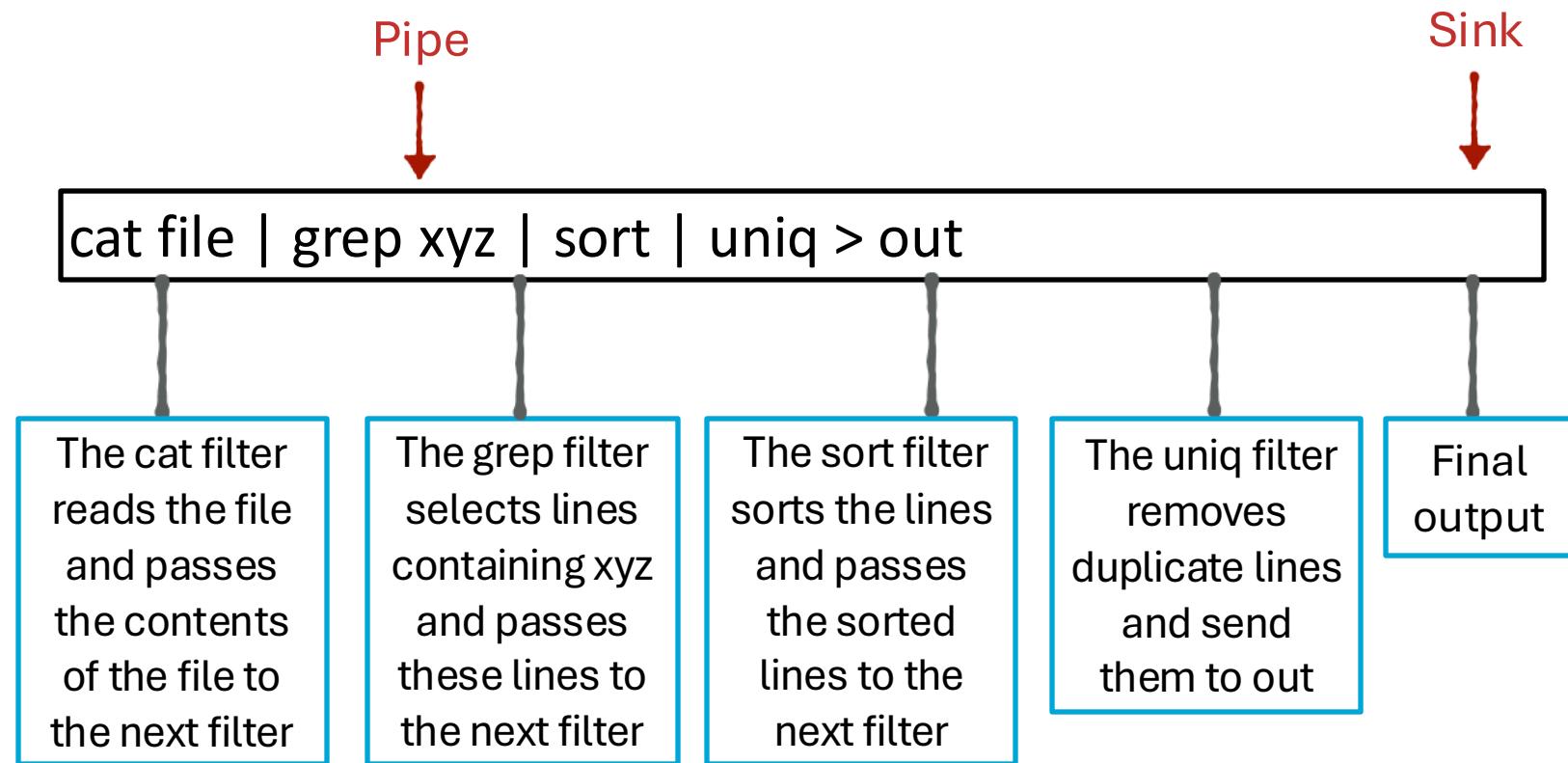


Deep Learning



Break or accelerate?

# Example: Unix Shell Commands



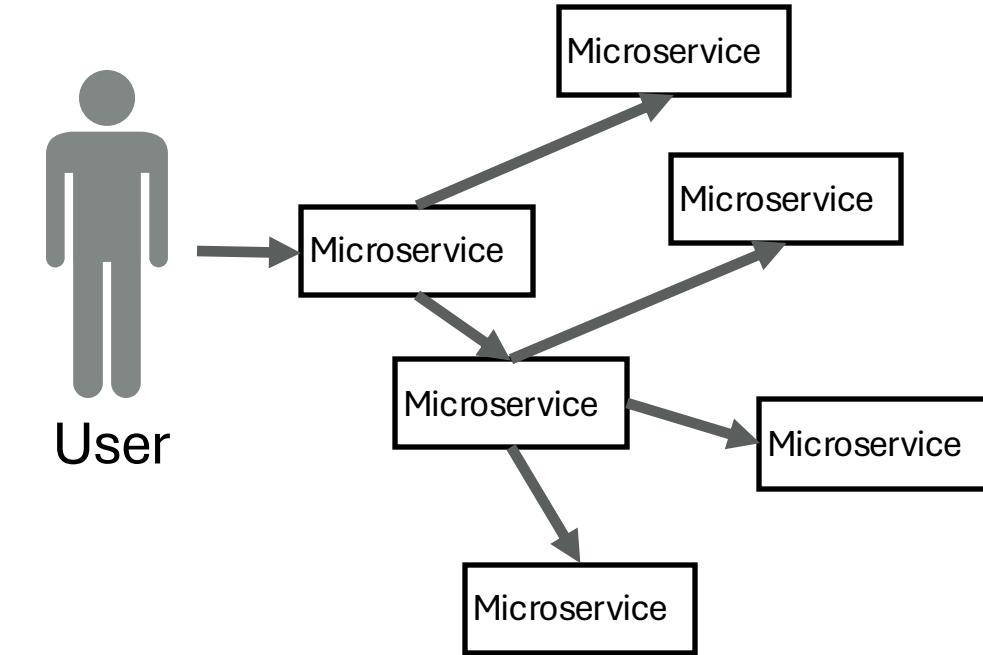
**What are the main advantages (Quality Attributes) of the Pipe-and-Filter pattern?**

# Pipe-and-Filter Pattern

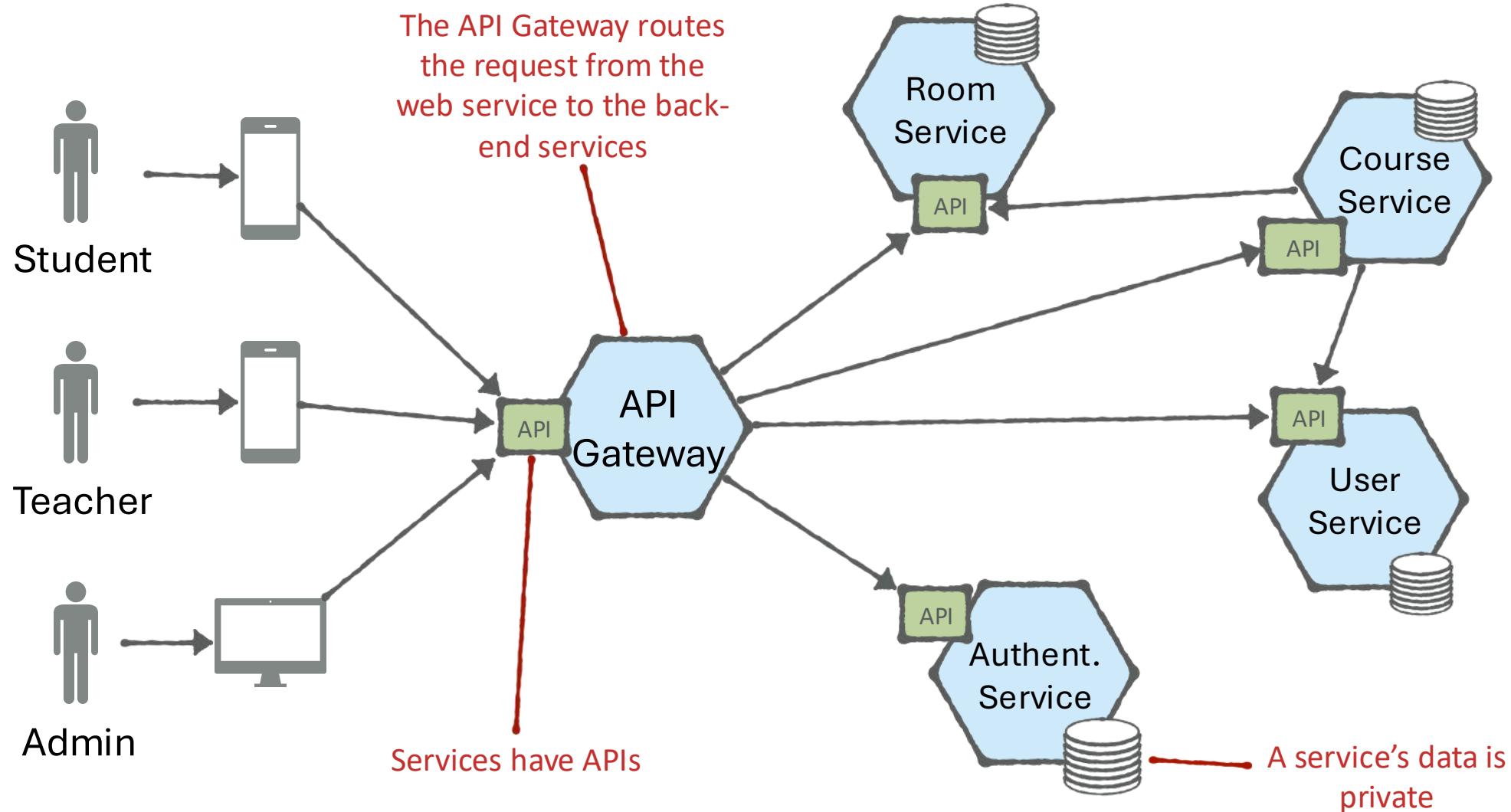
- Advantages
  - Reusability: possible to build different pipelines by recombining a given set of filters
  - Extensibility: it is very easy to add a new filter
  - Testability: filters can be tested separately
  - Concurrency: input and output consist of streams; filters start computing when they receive data (async)
- Disadvantages
  - Performance: large overhead due to data transformation (and depending on pipeline length)
  - Low responsiveness (latency): not useful for interactive applications

# Microservices Architecture

- Microservices re-arrange an application as a collections of **loosely** coupled services.
- Each microservice can be implemented in different programming languages, database, hardware and software environment, etc.

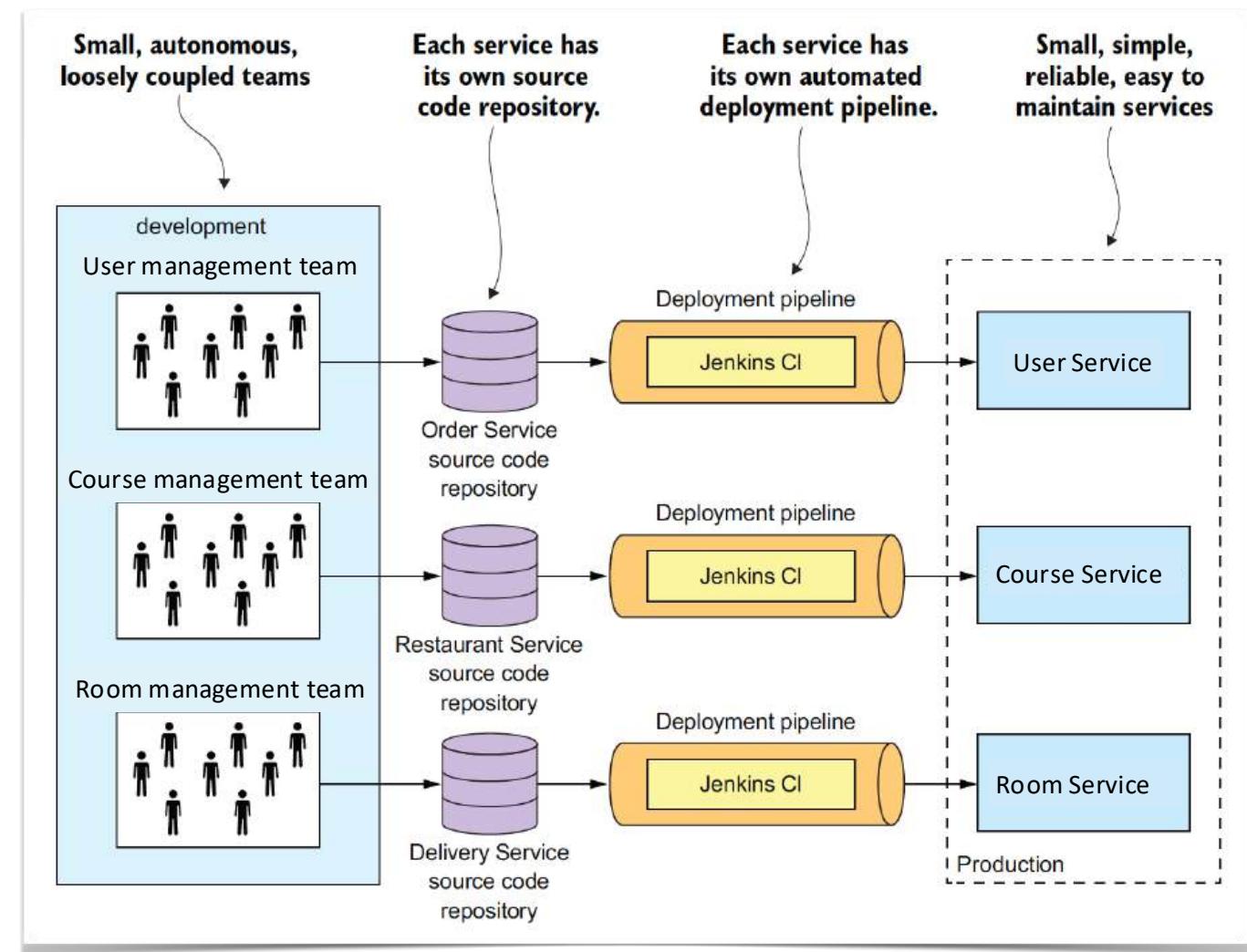


# Microservice Architecture Example – Course Management



# Benefits of Microservices

- Services are **small** and **easily maintained**.
- Services are **independently deployable**.
- Services are **independently scalable**.
- The microservice architecture enables teams to be **autonomous**.



# Microservices Architecture

- Advantages
  - **Modularity**: application is divided into small microservices that are easy to maintain, test, understand, etc.
  - **Scalability**: since microservices are implemented and deployed independently of each other, they can be monitored and scaled independently
  - **Integration**: microservices can be integrated with legacy systems
  - **Distributed development**: different teams can develop different microservices independently from one another
- Disadvantages
  - **Network overhead**: all communications between microservices happen in the network. Thus, there is higher network latency and message processing time than in-process calls within a monolithic architecture
  - **Deployment** is more complicated
  - **Security**: the more the data in the network the larger the attack surface
  - **Moving responsibilities** between services is more difficult



# **When should you use a microservices architecture?**

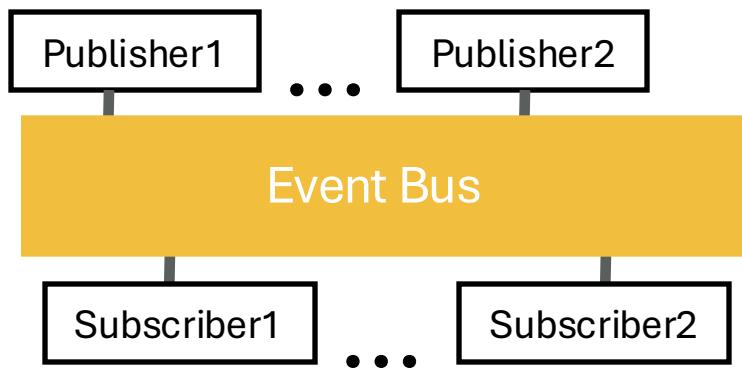
# Microservices Architecture

- When to use it
  - Large and complex system
  - Need for scaling up and down
  - Working with multiple software development teams
  - Scale of > 500,000 users
  - Heterogeneous tech stack
- When not to use it
  - Small or simple application
  - Intensive communication between services
  - Inadequate team size or no skills in distributed systems
  - Lack of experience with microservices
  - Early-stage projects or rapid prototyping
  - Data consistency requirements

It's usually better to grow  
into a microservice  
application (as a business)

# Publish-Subscribe Pattern

- Publishers and subscribers are independent and unaware of each other
- Multiple consumers subscribe to events published by various producers



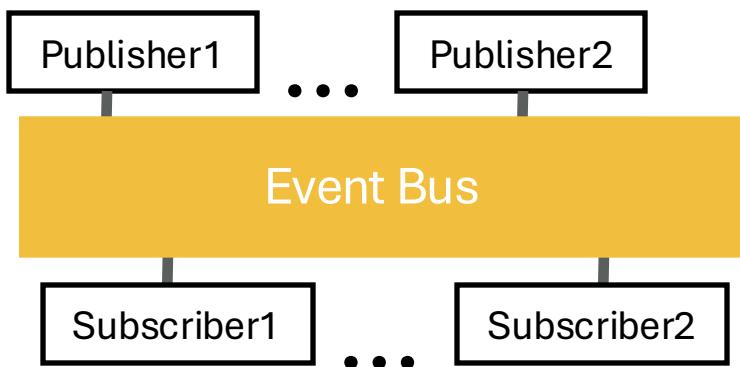
**Publisher:** any component that publishes an event. Specific events published should be documented

**Subscriber:** any component that subscribes to an event

**Event Bus:** responsible for registering component subscriptions and delivering published events (e.g., network protocol)

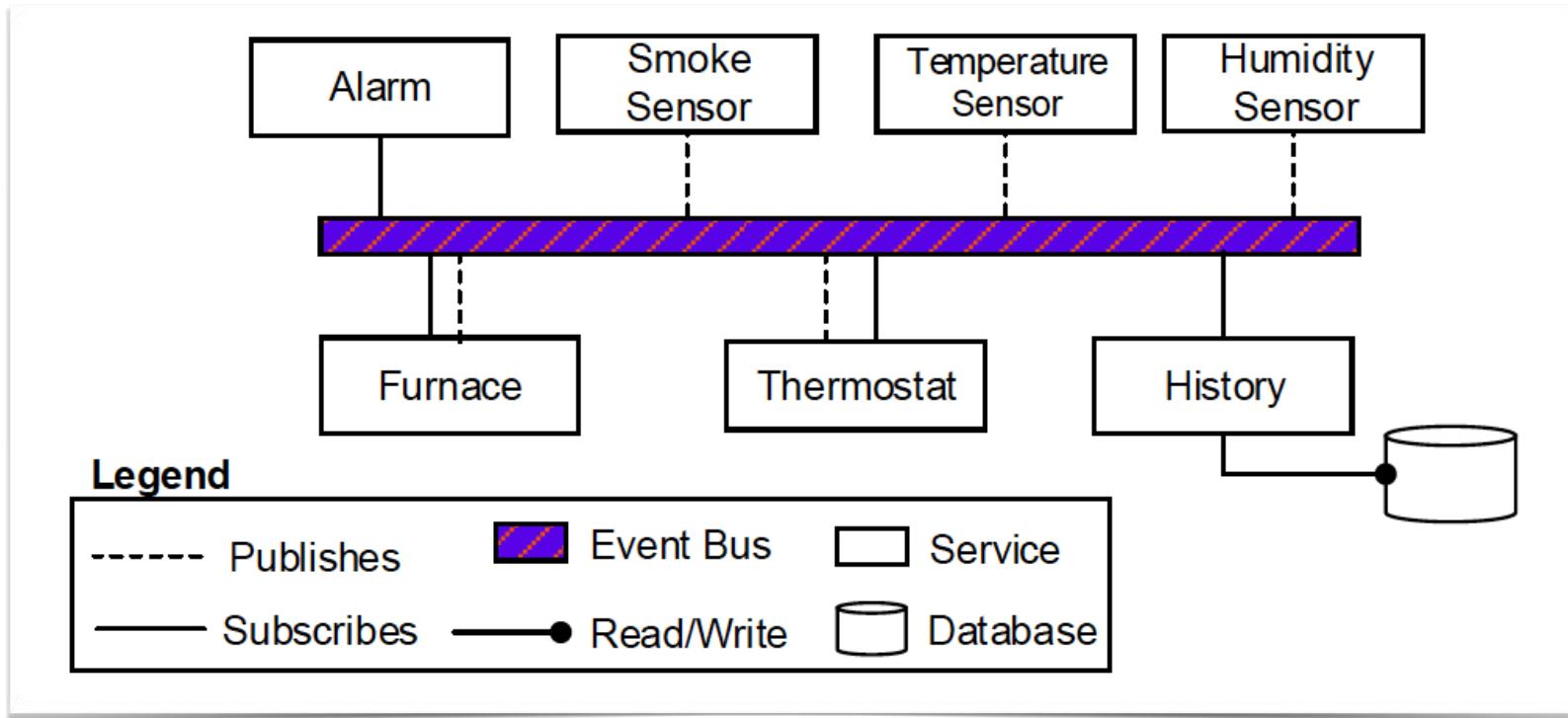
# Publish-Subscribe Pattern

- Publishers and subscribers don't know of each others existence
- Subscribers receive only a subset of published messages (the event bus filters)



- Filtering can be done by
  - **Topic:** publisher indicates topic and topics can be subscribed to
  - **Content:** based on constraints on attributes or content of message

# Publish-Subscribe Pattern



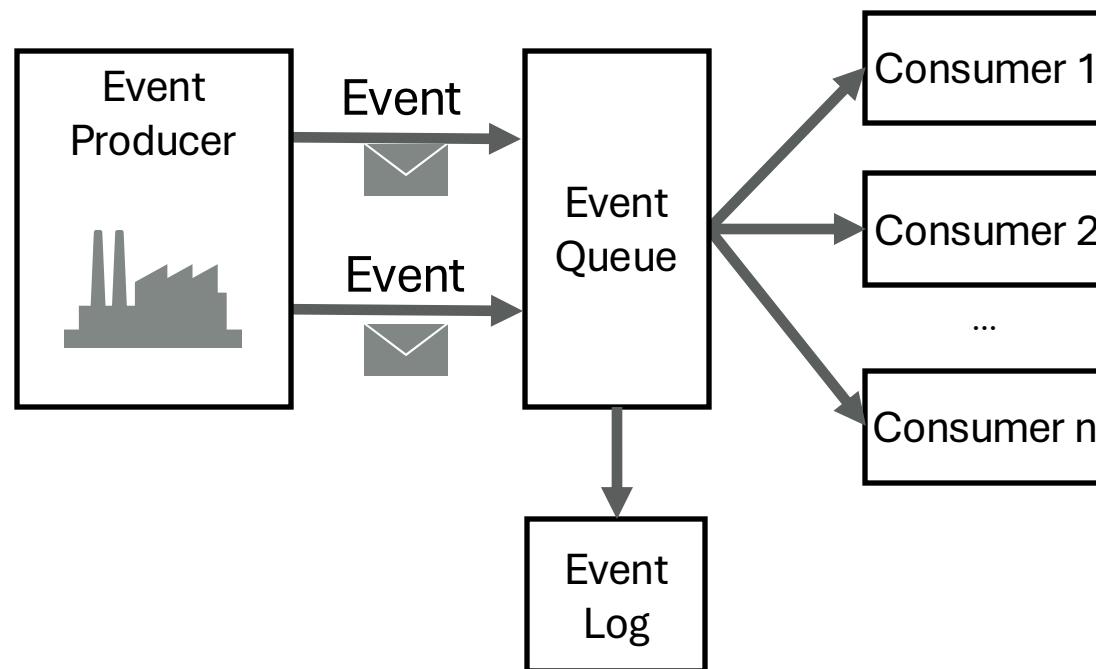
**What are the main advantages (Quality Attributes) of the Publish-Subscribe pattern?**

# Publish-Subscribe Pattern

- Advantages
  - Reusability: services can be reused to make many applications
  - Testability: publishers and subscribers can be tested separately
  - Scalability: we can have as many publishers or subscribers as we want
  - Separation of concerns
  - Loose coupling: publishers and subscribers are loosely coupled, and don't even need to know each other
- Disadvantages
  - Scalability (bottleneck): all messages travel through same event bus
  - Security: all messages are sent through the same event bus
  - Availability: did the subscriber receive the requested data?

# Event-Driven Architecture

The event-driven architecture (EDA) consists of event **producers** (or agents), event **consumers** (or sinks), and event **channels**.

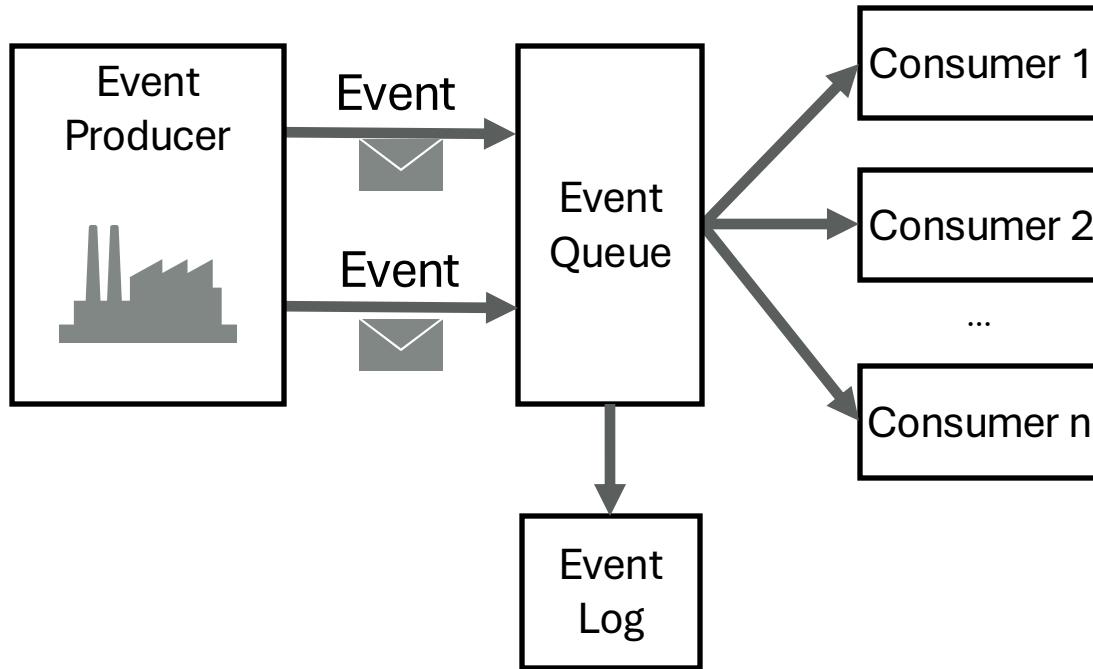


Events are the inputs to EDA. Events trigger transitions from one state to another one in our system

Producers have the responsibility to detect, gather, and transfer events.

Sinks (consumers) have the responsibility of applying a reaction as soon as an event is presented

# Event-Driven Architecture



This pattern is a popular **distributed asynchronous** architecture

Events are transmitted thought event **channels** (i.e., the network)

An event channel can be a **TCP/IP** connection, or any type of an **input file** (flat, XML format, e-mail, etc.)

The event queue (or event processing engine) identifies an event, and then selects the appropriate reaction/service

# Implementing an EDA

EDAs can be implemented using **existing frameworks** (integration hubs), which already provide the interfaces for events, brokers, and channels (and communication protocols).

So, you can focus on implementing your events and the **business logic** (the processors/consumers)

Some examples:

- Event mediator implementation: Spring Integration, Apache Camel, or Mule ESB
- Event flows can be implemented in Java or using a DSL (domain-specific language)
- In Apache ODE uses XML format to describes the data and steps required for processing an initial event



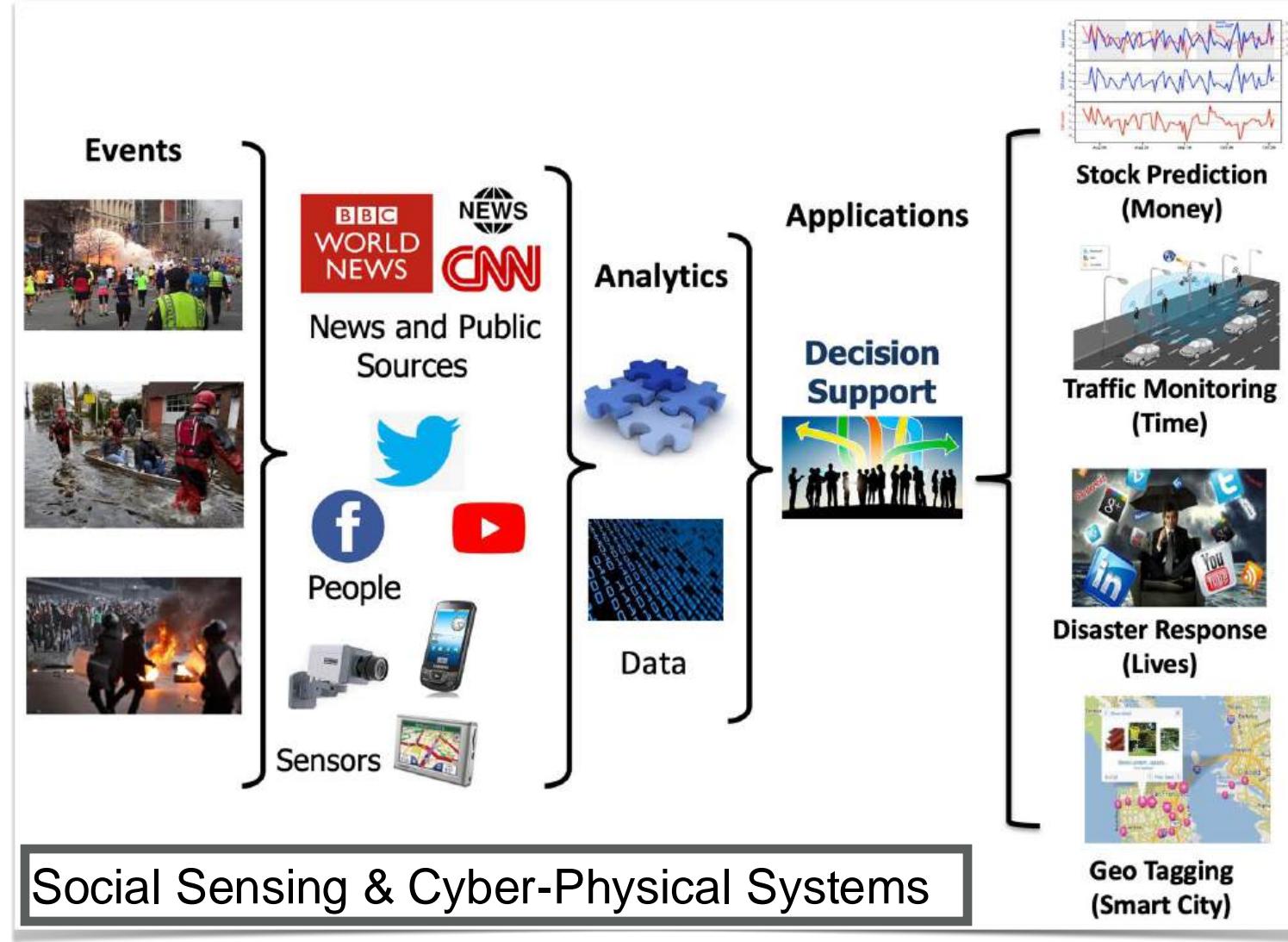
# Event-Driven Architecture

- Advantages
  - **Real-time**: very useful for real-time systems (e.g. alarm management in the EWI building, satellites, etc.)
  - **Loose coupling**: the event itself doesn't know about the consequences of its cause
- Disadvantages
  - **Increased complexity**: we need to implement the events, the emssages and the sinks (handlers)
  - **Security**: the event queue can become the main attack point (buffer overflow, DoS)
  - **Reliability**: potential loss of events/states

# EDA Example: WebLab

- Weblab uses an event driven architecture
- Disconnected frontend and backends
- Tasks are distributed to different backends based on their type
- There can be multiple backends of the same type

# Event-Driven Architecture Example



# The CAP Theorem

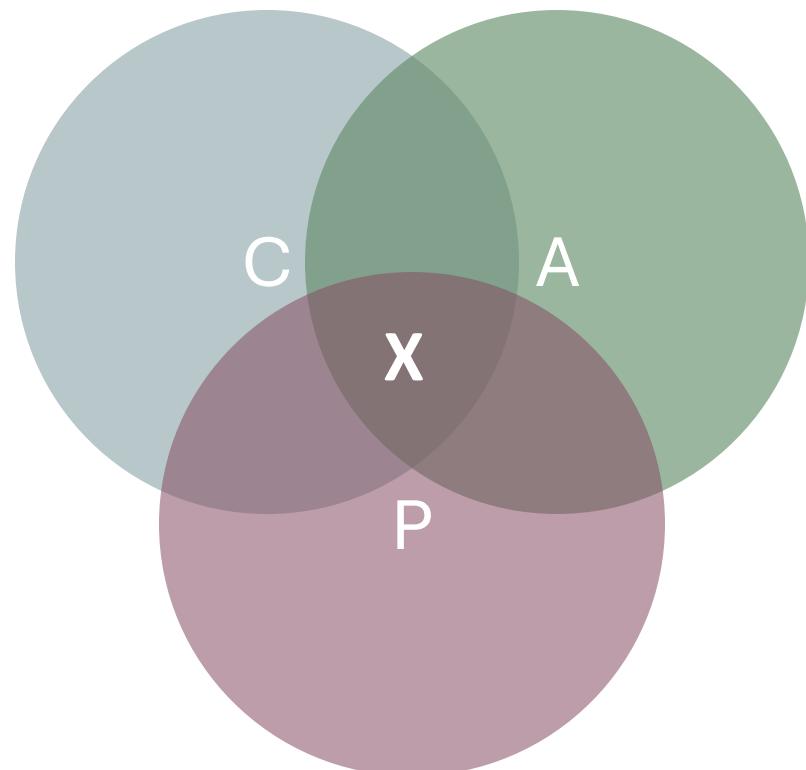
# The CAP Theorem

- Conjectured by Prof. Eric Brewer at PODC (Principle of Distributed Computing) 2000 keynote
- Described the trade-offs involved in distributed systems
- It is impossible for a web service to provide following three guarantees at the same time:
  - Consistency
  - Availability
  - Partition-tolerance

# The CAP Theorem

- **Consistency**: All nodes should see the same data at the same time
- **Availability**: Node failures do not prevent survivors from continuing to operate
- **Partition-tolerance**: The system continues to operate despite network partitions
- The **theorem**: “A distributed system **can satisfy any two of these guarantees at the same time but not all three**”.

# The CAP Theorem



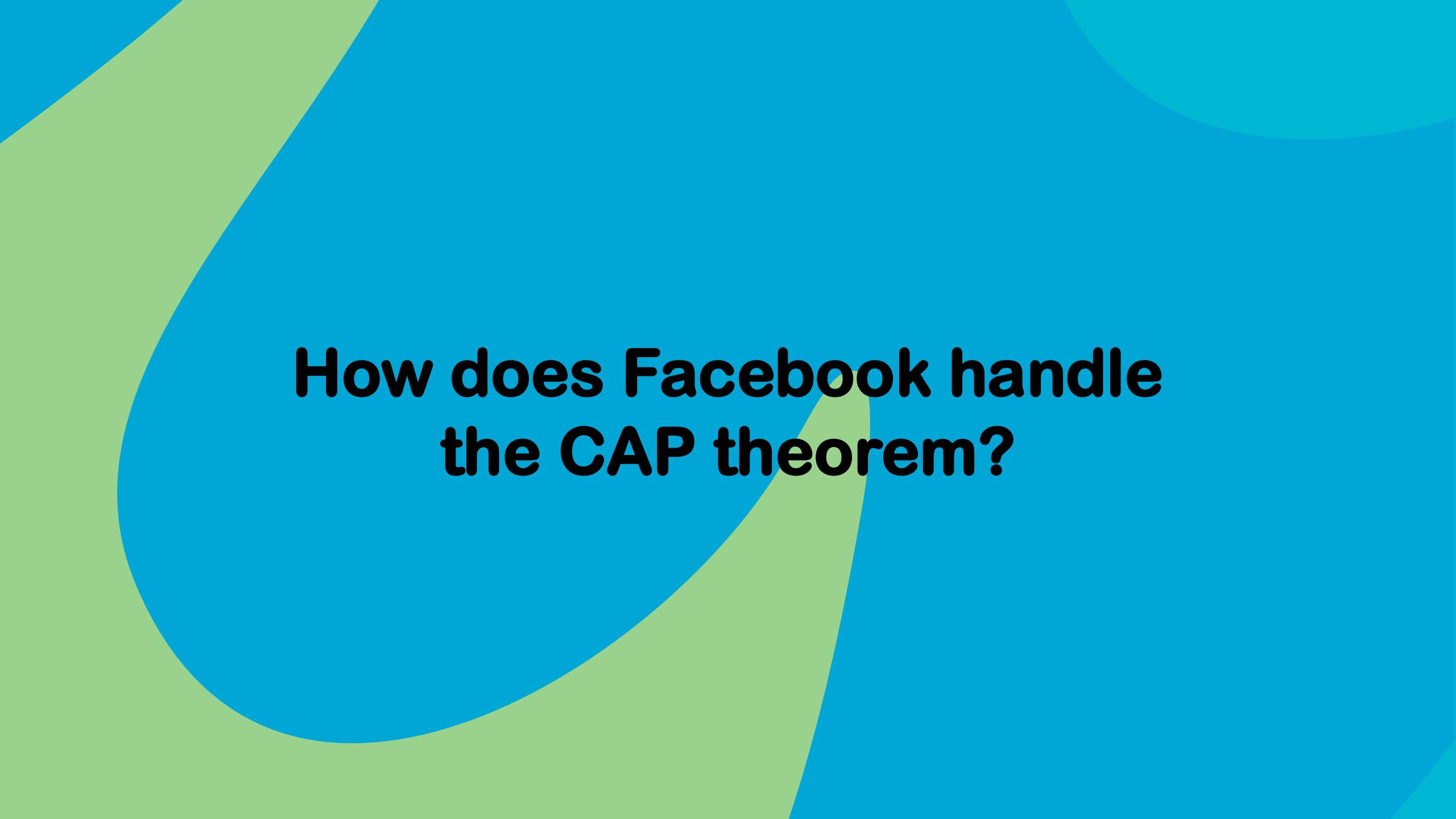
The **theorem**: “A distributed system **can** satisfy any **two** of these guarantees at the same time but not all **three**”.

The intersection between all **three** elements **is not possible**

We need to discover what is the best trade-offs for the system we have to develop

# The CAP theorem: why is it important?

- The future of databases is distributed (Big Data Trend, etc.)
- CAP theorem describes the trade-offs involved in distributed systems
- A proper understanding of CAP theorem is essential to making decisions about the future of distributed database design
- Misunderstanding can lead to erroneous or inappropriate design choices



# **How does Facebook handle the CAP theorem?**

# (Eventual) Consistency: A Facebook Example

- Bob finds an interesting story and shares with Alice by posting on her Facebook wall
- Bob asks Alice to check it out
- Alice logs in her account, checks her Facebook wall but finds:
  - Nothing is there!



# (Eventual) Consistency: A Facebook Example

- Bob tells Alice to wait a bit and check out later
- Alice waits for a minute or so and checks back:
  - She finds the story Bob shared with her!





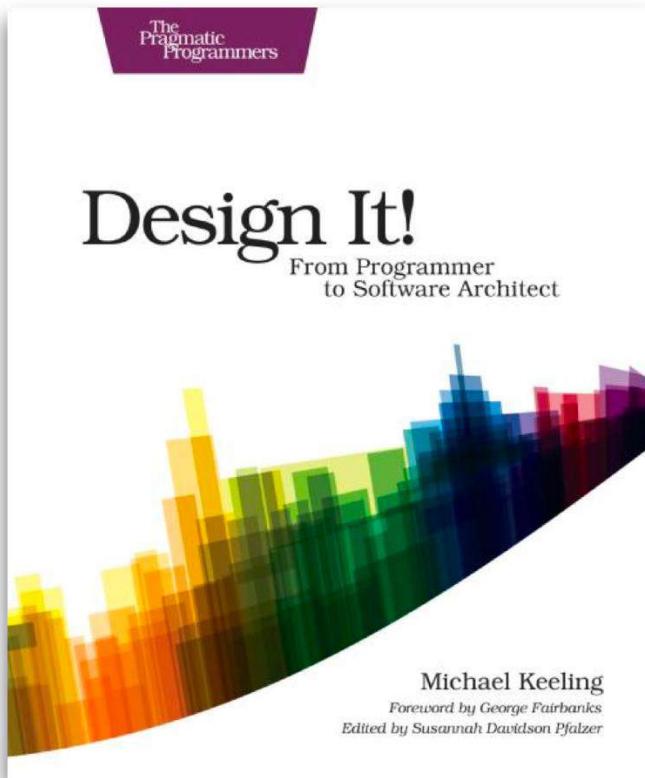
**Choose an Architecture  
before it chooses you!**

# Alert



- Do not overreact seeing all these patterns!
- Choose the architecture that fits best your problem
- Look at the trade-offs!
- Changing the architecture is very expensive if not done in early stages
- Document your architecture with UML (component diagrams)

# Further Reading



- “Design It! From Programmer to Software Architect”, Michael Keeling
- <https://tudelft.on.worldcat.org/oclc/1015687142>
- Chapter 7

# Question from WebLab

- I've given some detailed answers on WebLab to some of you, look at the grade/discussion

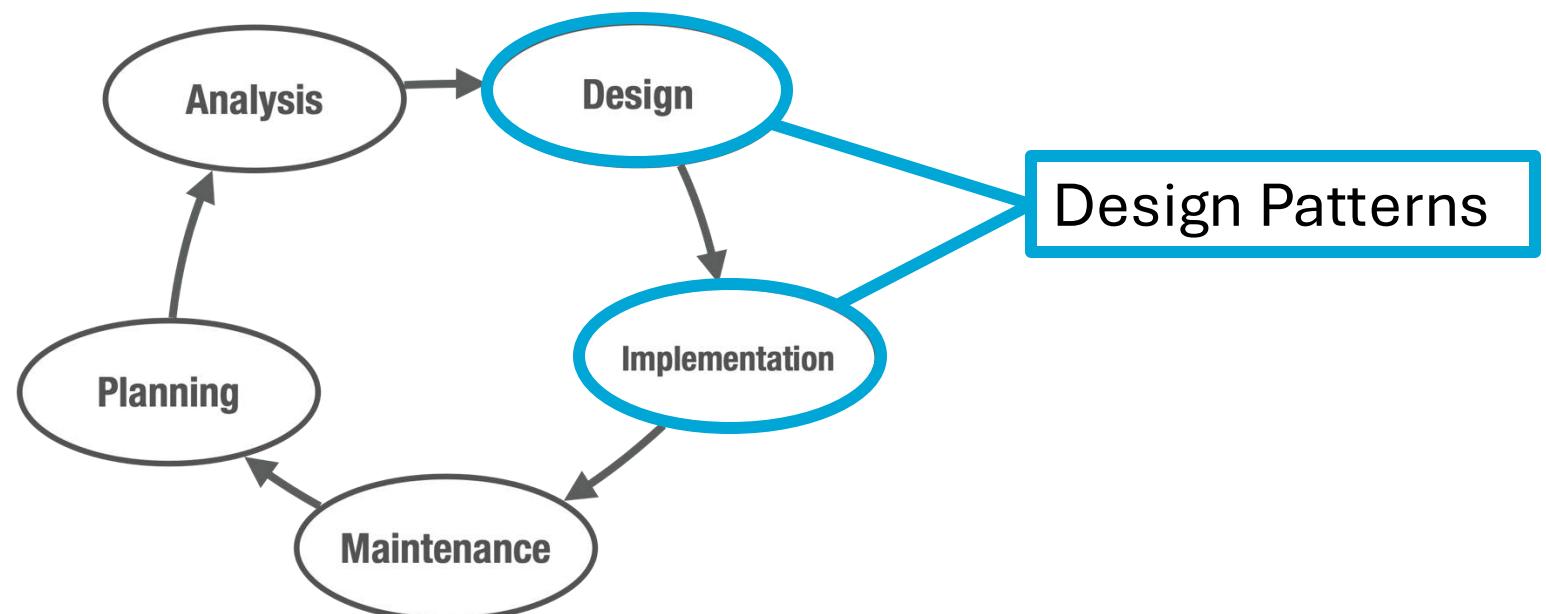
# Other Questions

# Summary

- Software Architecture is the organization of a system (the important parts of a system)
- There exist many different architectural patterns that can be used in an architecture.
- Presenting architecture can be done with various UML diagrams
- When presenting SA, you should use various views depending on whom you want to present it to.

# Next Time

- Moving down towards implementation

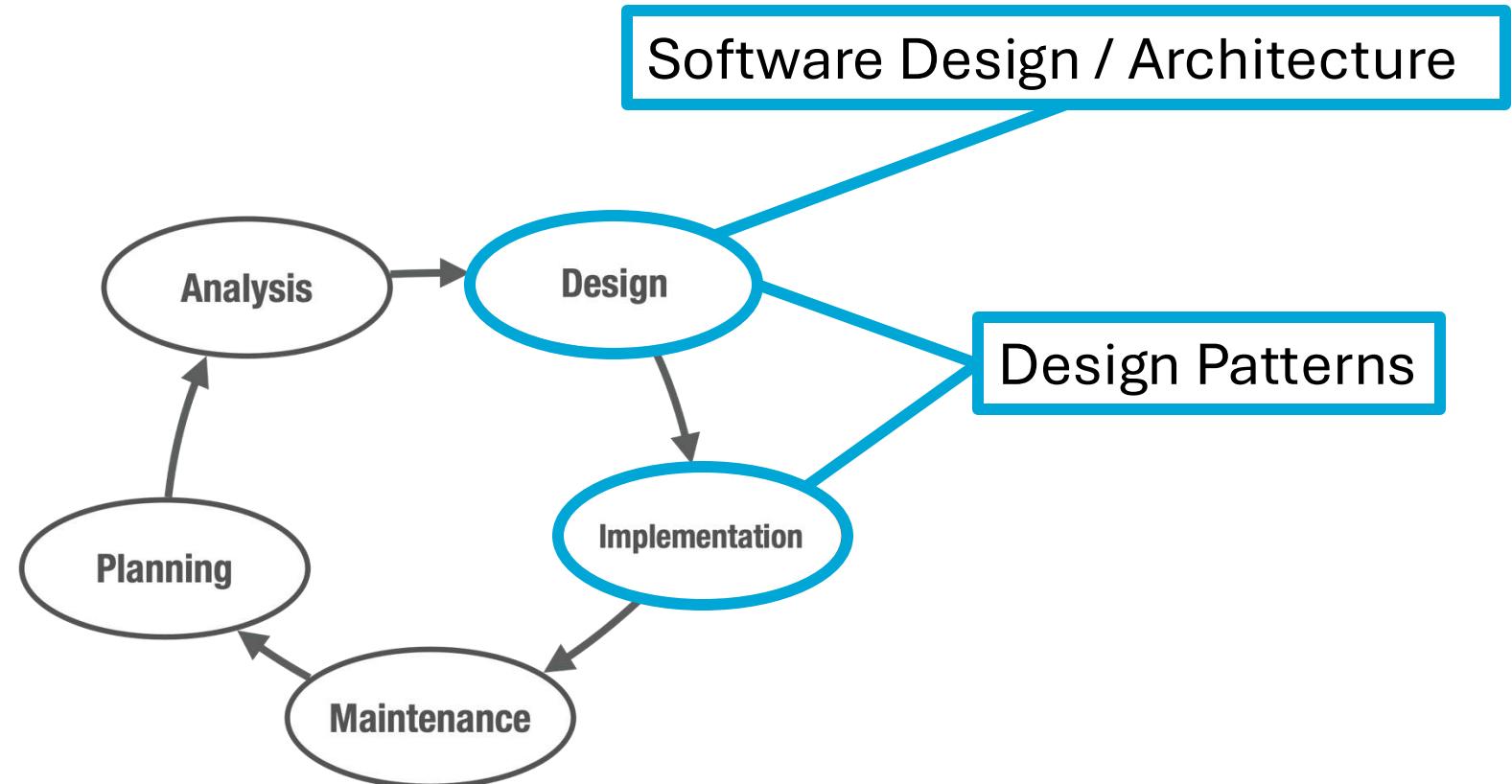


# **CSE2115 Software Engineering Methods**

**Design Patterns**

**Taico Aerts & Arie van Deursen**

# Software Development Lifecycle



# Learning Objectives

- Describe common design patterns and their tradeoffs
- Decide which design pattern to use given a scenario
- Identify common design patterns from UML Class Diagrams
- Draw UML Class Diagrams

# Further Reading



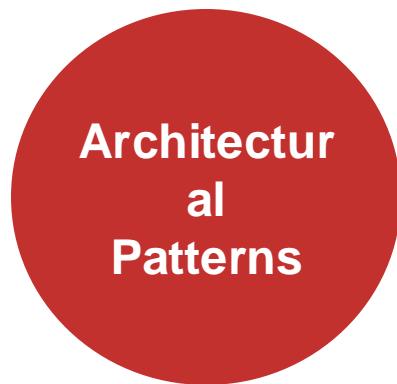
- Refactoring Guru
- <https://refactoring.guru>
- Highly recommended!

# Last Time

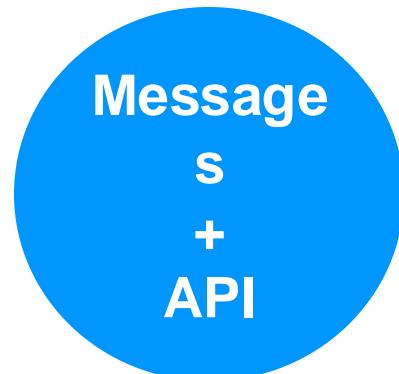
# Software Architecture

Software architecture is how the **components** of a software system are organized and assembled. How they **communicate** with each other. And the **constraints** that rule the whole system.

Software architecture has three main parts:



How components are organized



How components interact with one another

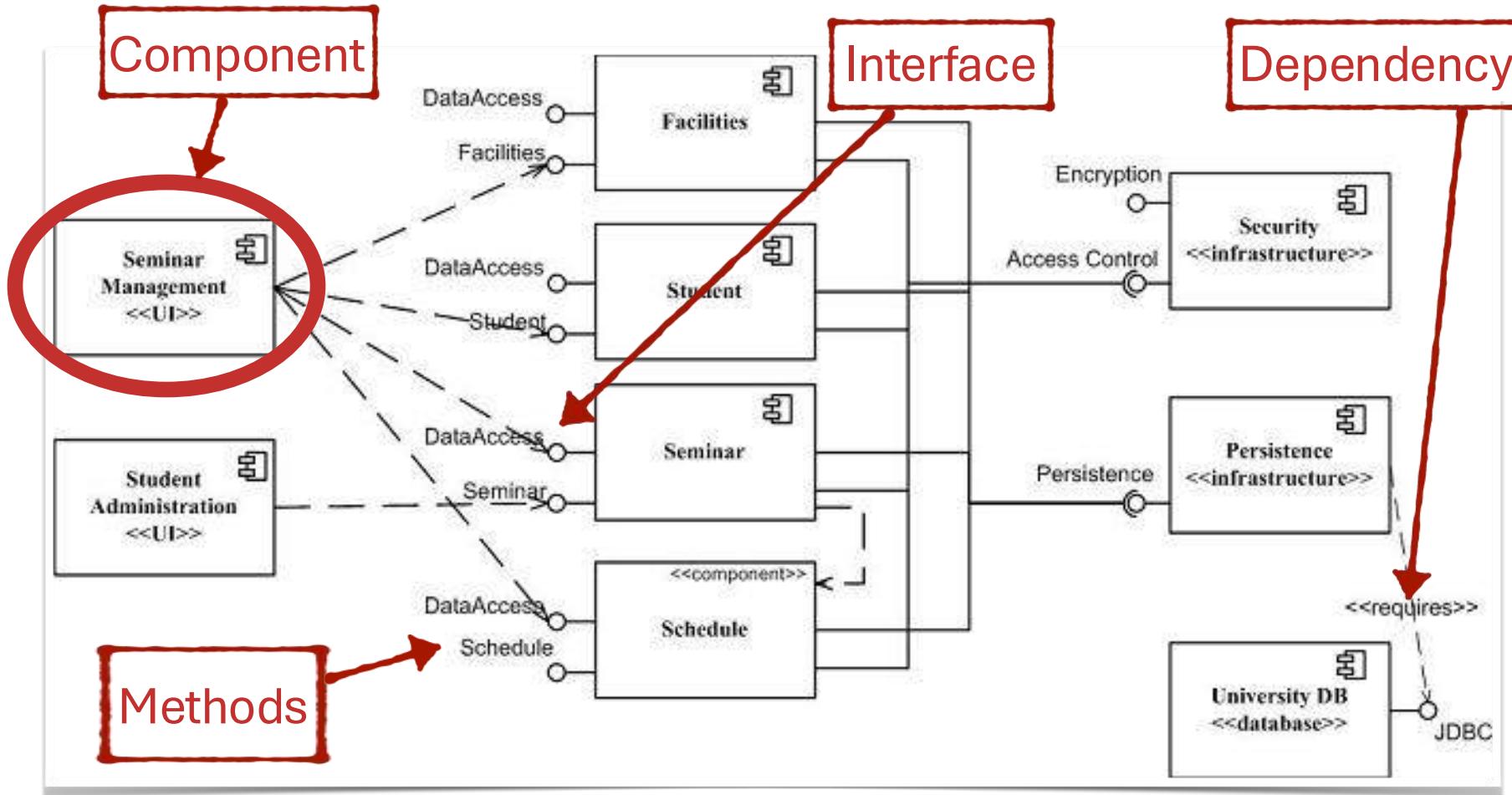


Constraints + non-functional Req.

# Architectural Views

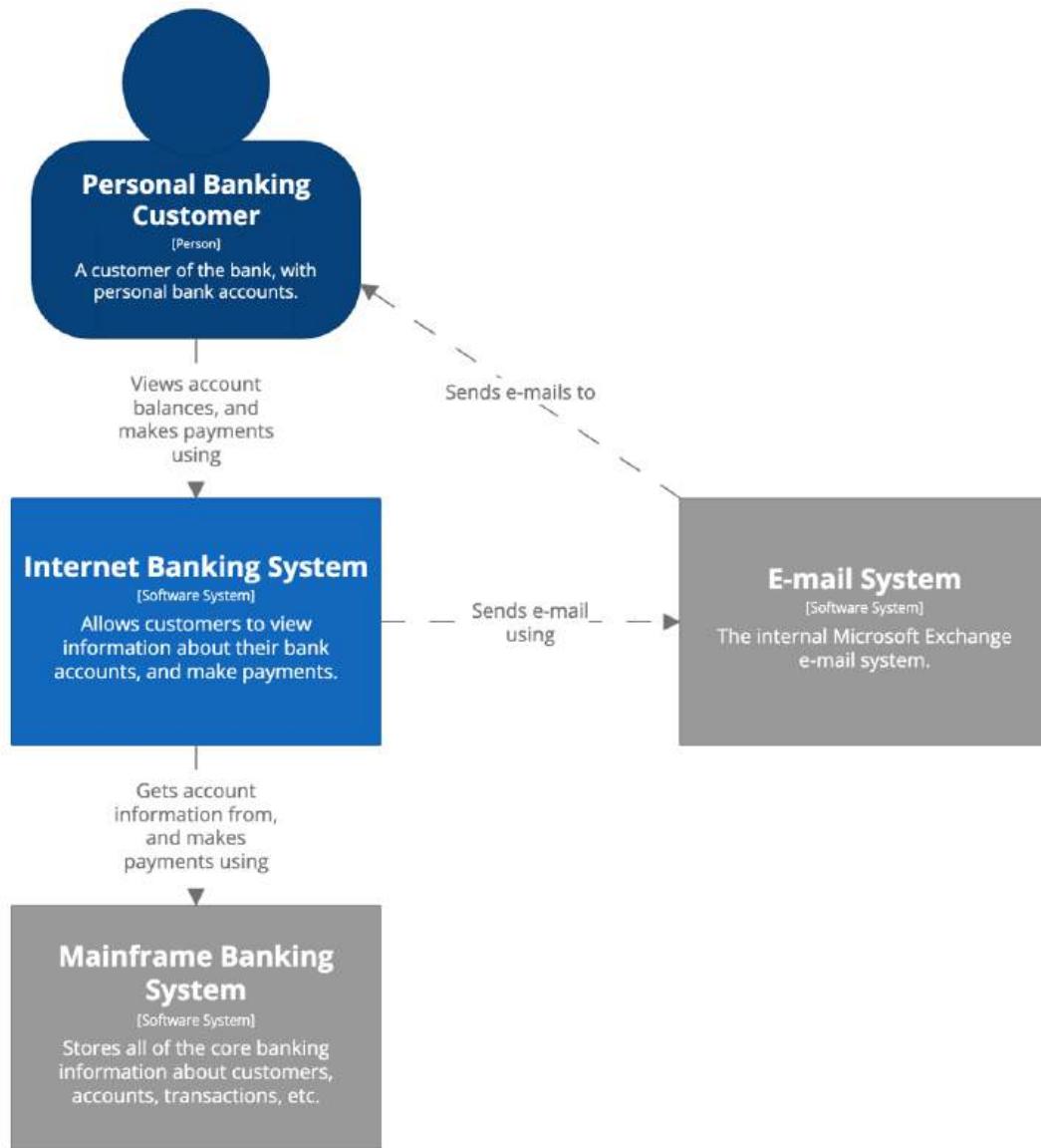
- Context View
  - (C4) Context Diagram
- Functional View
  - Component Diagrams, (Sequence Diagrams)
- Information View
  - Class Diagrams, ER Diagrams, Activity Diagrams
- Concurrency View
- Development View
- Deployment View
- Operational View

# Component diagrams in a nutshell

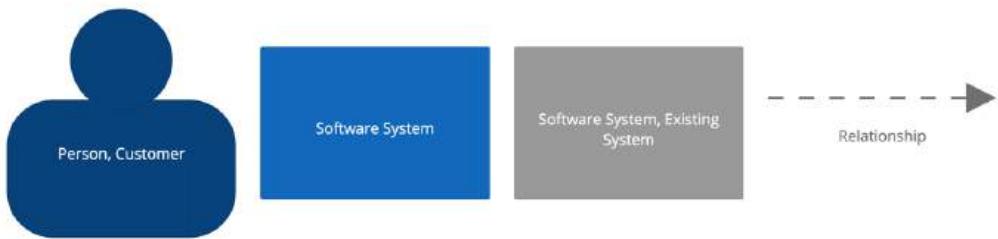


Component diagram for a simplified Seminar Management System

# C4 – (1) Context Diagram

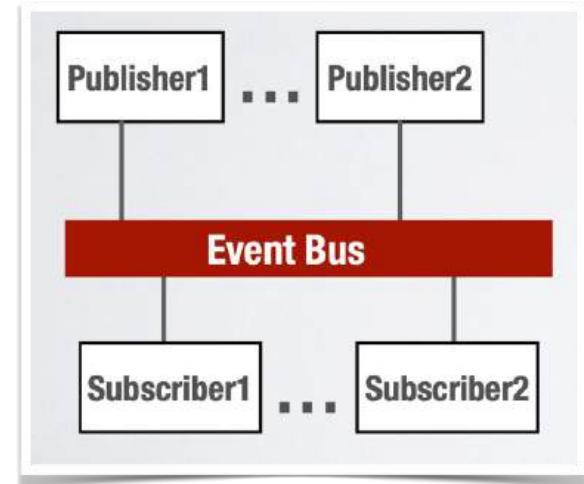
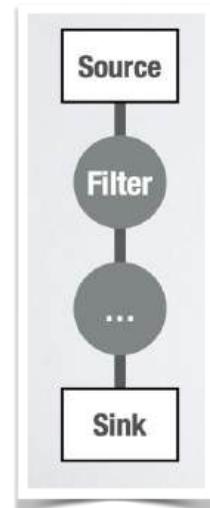
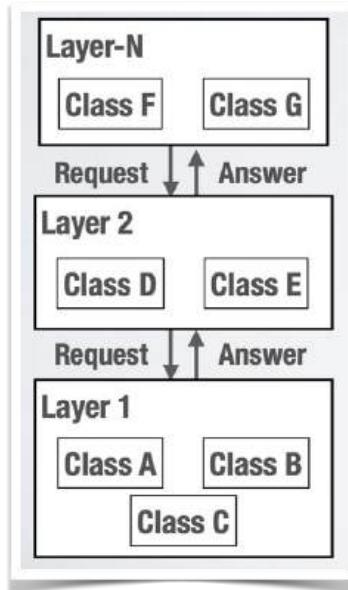
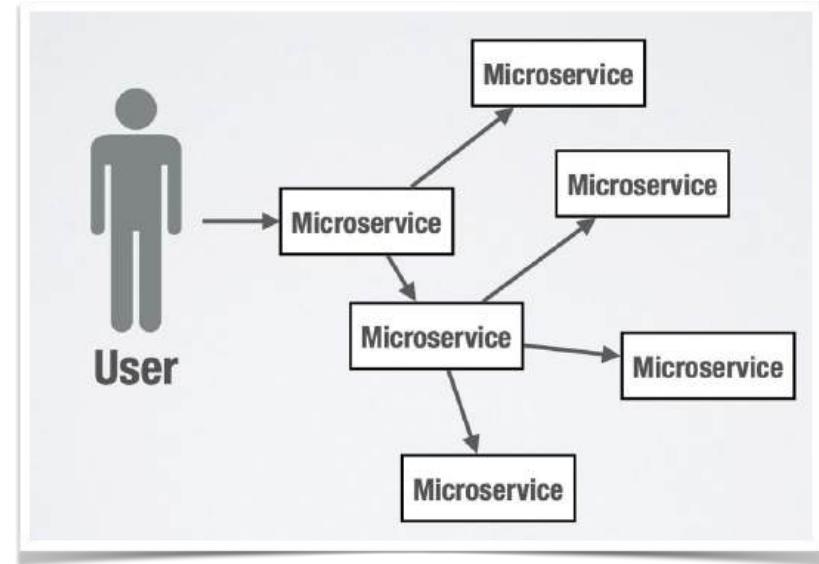
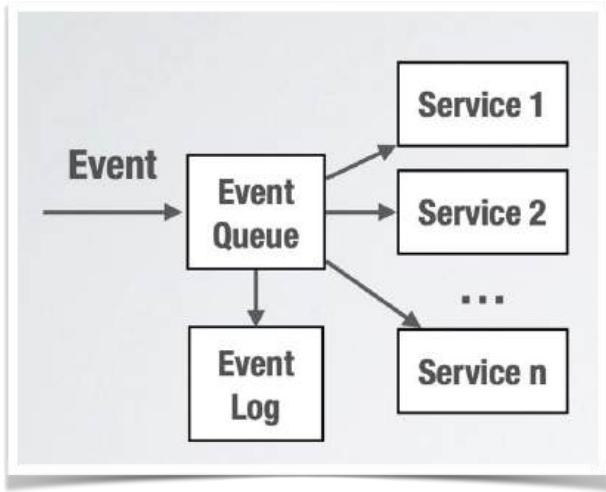


- Your system in center
- Surround with users and other systems it interacts with
- Focus on big-picture



All images on the C4 Model are from  
<https://c4model.com>

# Architectural Patterns

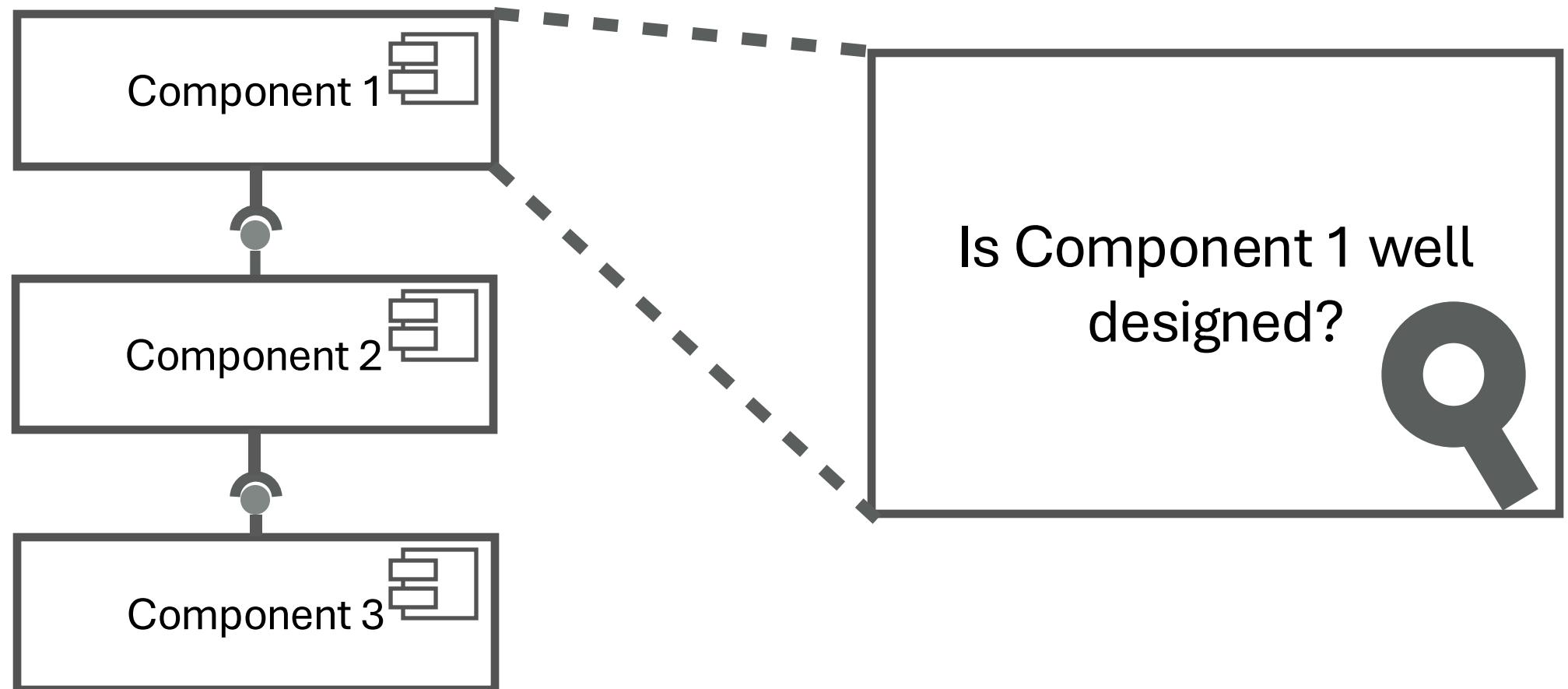


# Quality attributes

- **Performance:** How fast is the system response?
- **Usability:** How easy-to-learn is the system?
- **Maintainability:** How easy is to make changes?
- **Testability:** How easily can we verify that the system works correctly?
- **Reusability:** Can we reuse software components?
- **Availability:** Is the system always available when needed?
- **Integrity:** Can unauthorised users change the data?

# **From High-Level to Low-Level Design**

# From High-Level to Low-Level Design





Before further discussing program design, we need to introduce a proper a notation for conceptual modeling of the structure of a program

# Class Diagrams



# UML Class Diagram

- Graphical notation to design and visualize classes in OO systems
- Components
  - Classes
  - Attributes
  - Methods (or operations)
  - Relationships
- Class diagrams tend to become data models.
  - Class diagrams focus on behaviour.  
Class diagram != ER diagram

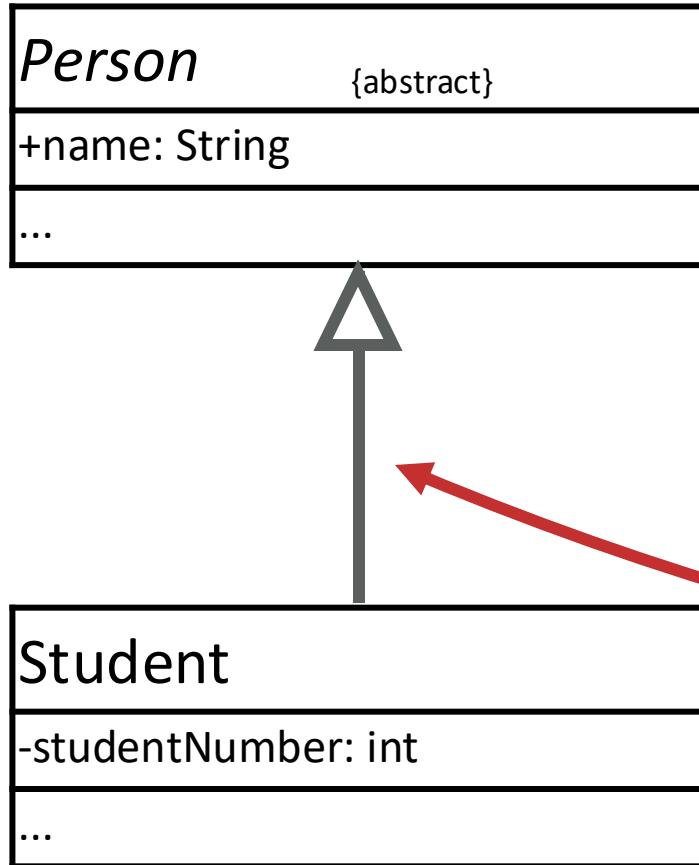
# UML Class Diagram



+ public  
- private  
# protected  
~ package

Visibility

# UML Class Diagram

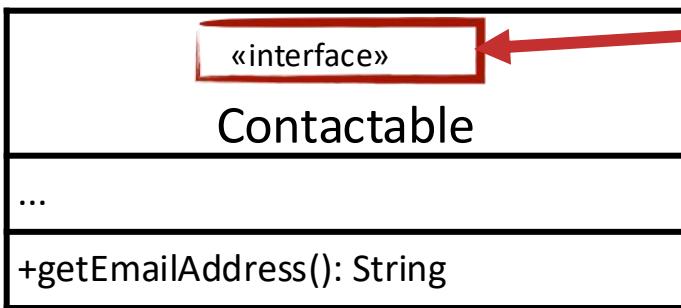


```
abstract class Person {  
    public String name;  
}
```

```
class Student extends Person {  
    private int studentNumber;
```

```
}
```

Extension represented by solid line with hollow arrow head



“Stereotype” to represent interfaces

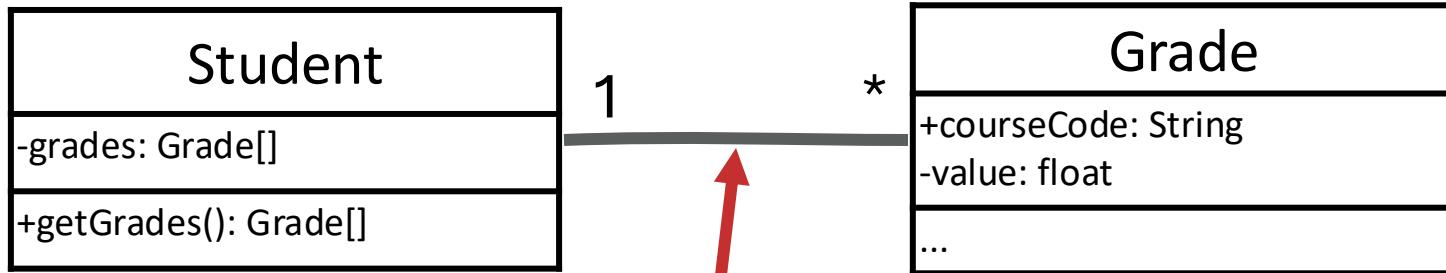
```
public interface Contactable {  
    String getEmailAddress();  
}
```

Realisation of interface represented  
by dashed line with hollow head

```
abstract class Person implements Contactable {  
    public String name;  
}
```

```
class Student extends Person {  
    private int studentNumber;  
}
```

# UML Class Diagram



Associations are represented using a solid line between classes.

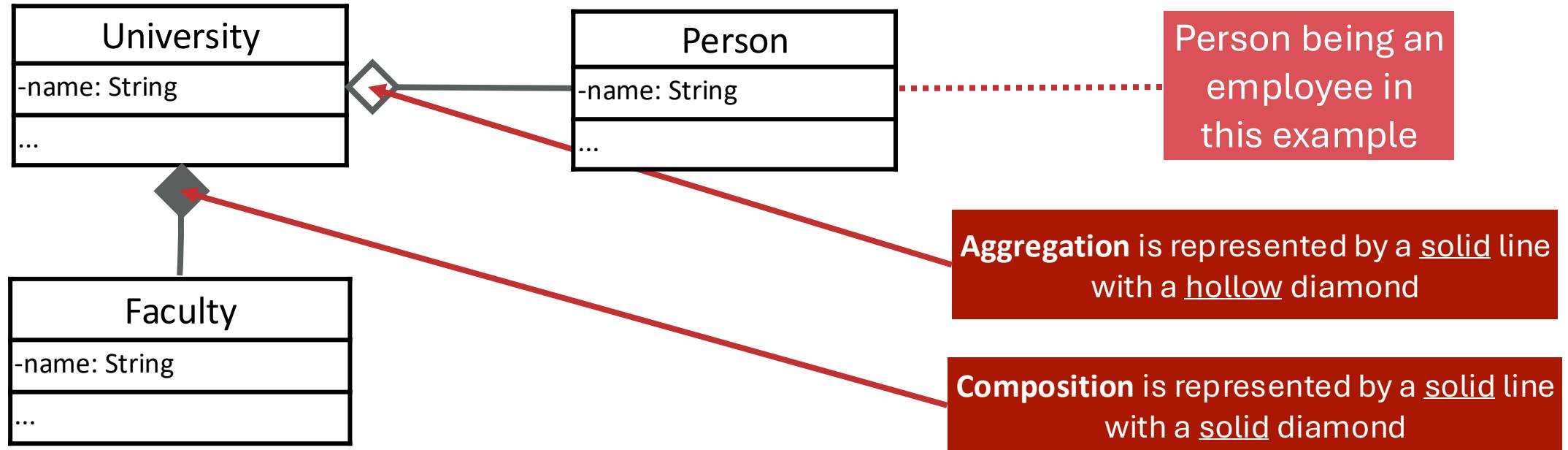
```
class Student {
    private Grade[] grades;

    public Grade[] getGrades() {
        return this.grades;
    }
}
```

Types of associations between **A** and **B**

- **A** sends message to **B**
- **A** creates instance of **B**
- **A** has attribute of type **B** (or collection of)
- **A** receives a message with **B**

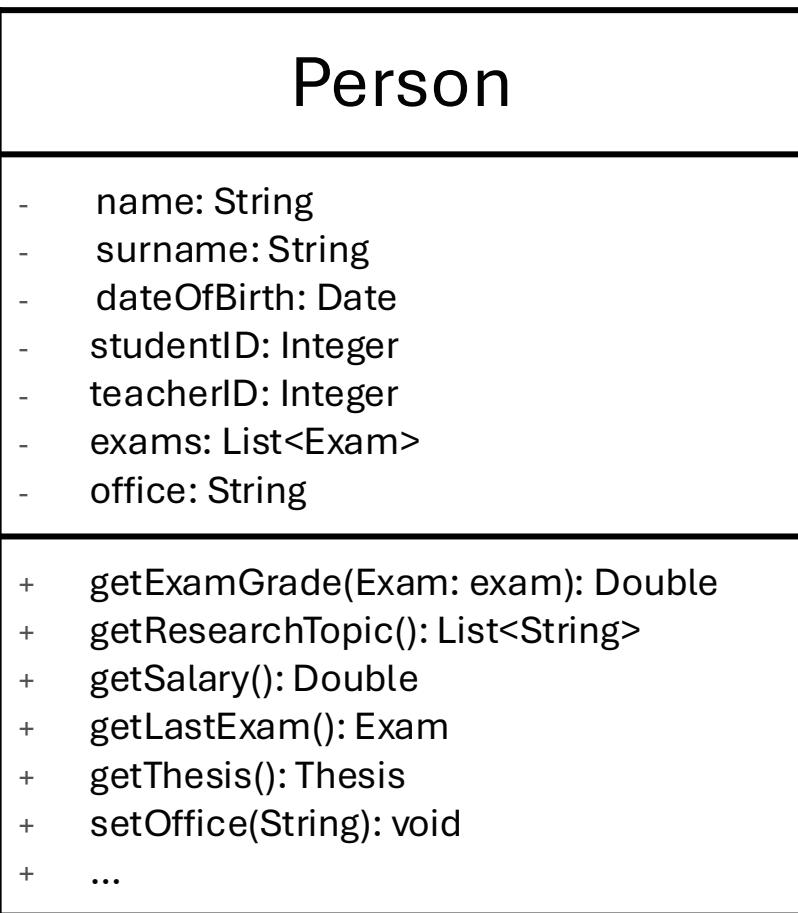
# Aggregation and Composition



- **Aggregation**: A contains B, but B can live without A
- **Composition**: A manages lifecycle of B, B cannot exist without A
- When the university goes out of business then its faculties go too. The Persons (employees/students) can move to another uni.

# **Is our Software Well-Designed?**

# Example 1



```
public class Person {  
    private String name;  
    private String surname;  
    private Date dateOfBirth;  
    private Integer studentID;  
    private Integer teacherID;  
    private LinkedList<Exam> exams;  
    private int salaryScale;  
    // more attributes  
  
    public Person(){...}  
  
    public double getExamGrade(Exam exam) throws Exception {  
        int index = this.exams.indexOf(exam);  
        if (index == 1)  
            throw new Exception("Exam not found!");  
        return exams.get(index).getGrade();  
    }  
  
    public double getSalary(){  
        switch (this.salaryScale) {  
            case 1:  
                return 10;  
            // ....  
            default:  
                return 0;  
        }  
    }  
  
    public Exam getLastExam(){  
        return this.exams.getLast();  
    }  
    // ....  
}
```

# Example 1

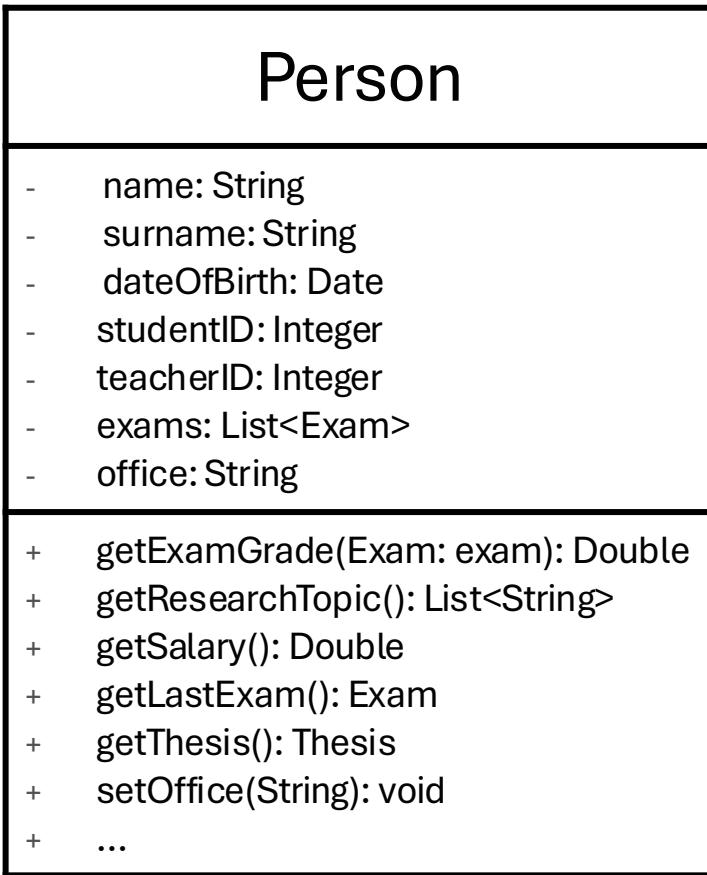
Person	
-	name: String
-	surname: String
-	dateOfBirth: Date
-	studentID: Integer
-	teacherID: Integer
-	exams: List<Exam>
-	office: String
+	getExamGrade(Exam: exam): Double
+	getResearchTopic(): List<String>
+	getSalary(): Double
+	getLastExam(): Exam
+	getThesis(): Thesis
+	setOffice(String): void
+	...

Is the code/diagram  
well designed?

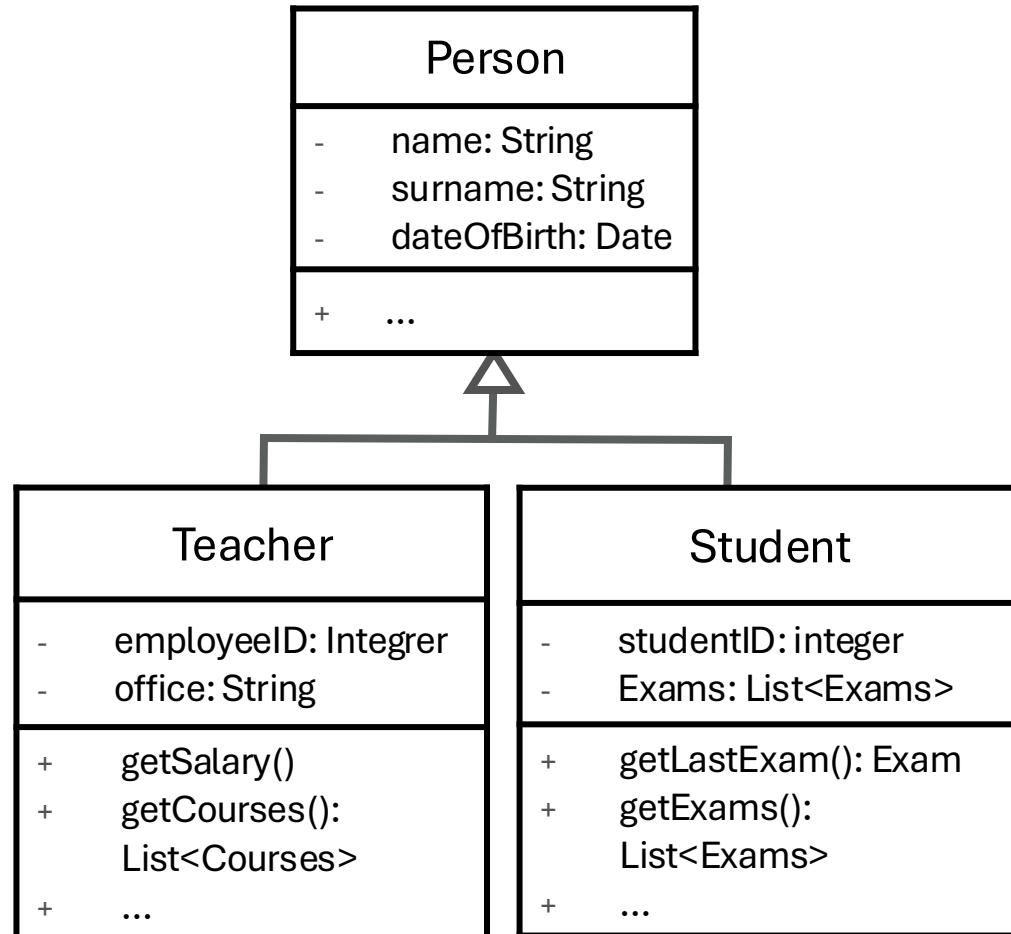
Why?

# Example 1

## Solution #1 (Low Cohesion)



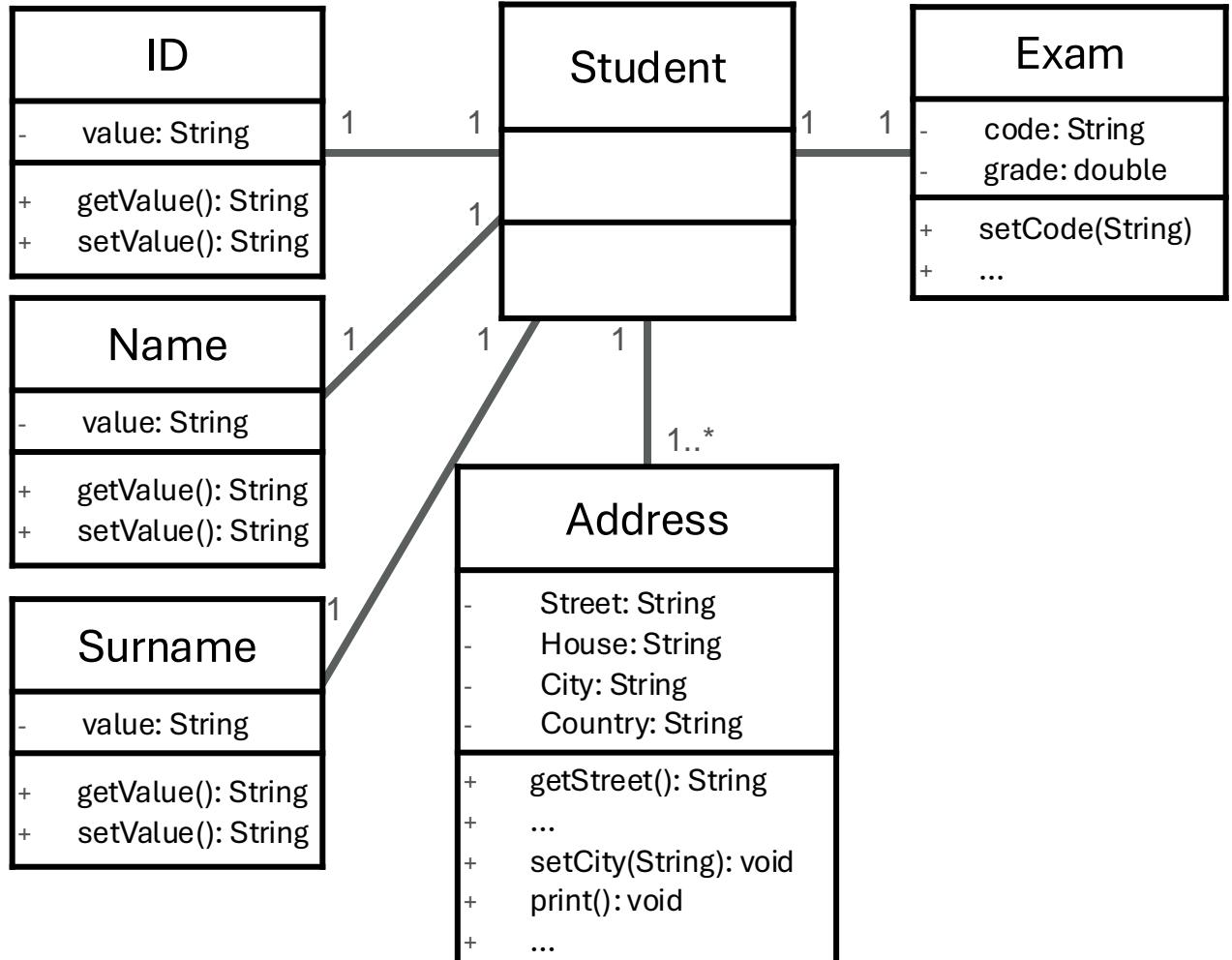
## Solution #2 (High Cohesion)



# Cohesion

- Cohesion is the degree to which methods inside a module belong together (intra-module complexity)
- A module can be a class (e.g., in Java), a function (e.g., in C), or higher-level software component (e.g., package, maven module)
- High Cohesion: classes (or modules) should be independent as far as possible
- High cohesion simplifies modification (all relevant code in one place)

# Example 2



Is the code/diagram  
well designed?

Why?

# Coupling

- Coupling is the degree to which one class (or module) relies on (calls or uses methods of) another class (inter-module complexity)
- Low Coupling: a module should implement one single, clear, well-defined task
- High Coupling leads to lower maintainability: a change in one unit affects other units implicitly
- Duplicated code often is an indication that a better solution (in the design and implementation) is possible

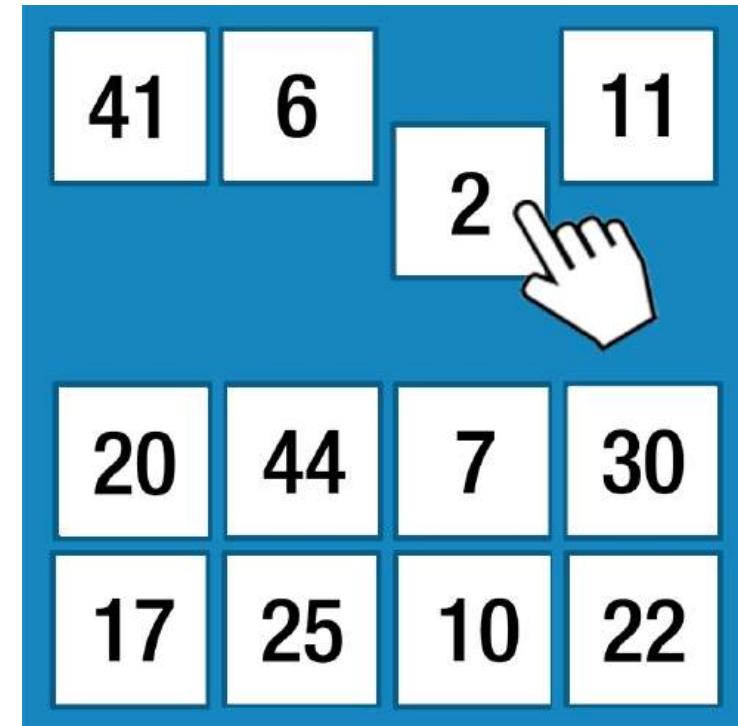
# Cohesion vs. Coupling



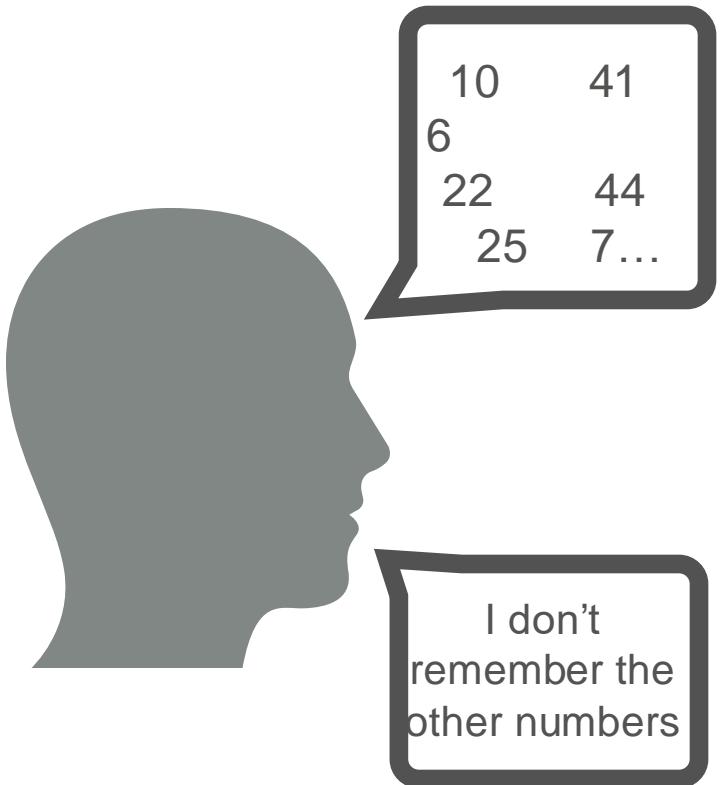
- High cohesion facilitates reuse (well-defined modules)
- Low coupling simplifies modification (all relevant code in one place)
- Higher cohesion can lead to higher coupling and vice versa
- Action: split or combine modules

# Does Design Complexity Really Matter?

- Numbers will appear on the right
- Try to memorize as many as you can



# Does Design complexity Matter?



(Including developers)

The average person can hold 5-7 concepts in the short-term memory.

*"The Magical Number Seven, Plus or Minus Two"*,  
George Miller, 1956

When we program, we should target simple design with low complexity (better cohesion, lower coupling)

When the design complexity exceeds what the developers can handle bugs can occur more often

# Does Design complexity Matter?

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 29, NO. 4, APRIL 2003 297

## Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects

Ramanath Subramanyam and M.S. Krishnam

**Abstract**—To produce high quality object-oriented (OO) applications, a strong emphasis on design aspects, especially during the early phases of software development, is necessary. Design metrics play an important role in helping developers understand design aspects of software and, hence, improve software quality and developer productivity. In this paper, we provide empirical evidence supporting the role of OO design complexity metrics, specifically a subset of the Chidamber and Kemerer suite, in determining software defects. Our results, based on industry data from software developed in two popular programming languages used in OO development, indicate that, even after controlling for the size of the software, these metrics are significantly associated with defects. In addition, we find that the effects of these metrics on defects vary across the samples from two programming languages—C++ and Java. We believe that these results have significant implications for designing high-quality software products using the OO approach.

**Index Terms**—Object-oriented design, software metrics validation, object-oriented languages, C++, Java.

### 1 INTRODUCTION

THE object-oriented (OO) approach to software development promises better management of system complexity and a likely improvement in project outcomes such as quality and project cycle time [8]. Research on metrics for OO software development is limited and empirical evidence linking the OO methodology and project outcomes is scarce. Recent work in the field has also addressed the need for research to better understand the determinants of software quality and other project outcomes such as productivity and cycle-time in OO software development [4], [21]. Of these outcomes, the importance of detection and removal of defects prior to customer delivery has received increased attention due to its potential role in influencing customer satisfaction [27] and the overall negative economic implications of shipping defective software products [32]. Hence, researchers have proposed several approaches to reduce defects in software (for e.g., see [1], [33], [39]). Suggested solutions include improvement of clarity in software design, effective use of process and product metrics, achievement of consistency and maturity in the development process, training of software development teams on tracking in-process defects, and promotion of practices such as peer reviews and causal defect analyses.

Empirical evidence in support of the effectiveness of the above approaches has also been presented [33], [34]. For example, Krishnam et al. empirically show that higher upfront investment in design helps in controlling costs as well as in improving quality [34]. It has also been shown that a number of software size-related metrics such as lines of code and McCabe's cyclomatic complexity are associated with defects and maintenance changes in a software system [1], [33]. Similarly, prior research has also shown that measures of testing effectiveness and test coverage can significantly explain defects [39]. However, most of these studies are primarily based on data from software developed using traditional software development methods and our understanding of the applicability of these approaches and metrics in OO development settings is limited.

Design complexity has been conjectured to play a strong role in the quality of the resulting software system in OO development environments [8]. Prior research on software metrics for OO systems suggests that structural properties of software components influence the cognitive complexity for the individuals (e.g., developers, testers) involved in their development [12]. This cognitive complexity is likely to affect other aspects of these components, such as fault-proneness and maintainability [12]. Design complexity in traditional development methods involves the modeling of information flow in the application. Hence, graph-theoretic measures [36] and information-content driven measures [30] were used for representing design complexity. In the OO environment, certain integral design concepts such as inheritance, coupling, and cohesion<sup>1</sup> have been argued to significantly affect complexity. Hence, OO design complexity measures proposed in literature have captured these design concepts [19], [20].

One of the first suites of OO design measures was proposed by Chidamber and Kemerer [19], [20] (henceforth, CK). The authors of this suite of metrics claim that

<sup>1</sup> Inheritance represents the degree of reuse of methods and attributes via the inheritance hierarchy. Coupling is a measure of interdependencies among the objects, while cohesion is the degree of conceptual consistency within an object.

The authors are with the University of Michigan Business School, Tappan Street, Ann Arbor, MI 48109. E-mail: ramanath.mkrishna@umich.edu.  
Manuscript received 8 Feb. 2001; revised 12 June 2002; accepted 28 Aug. 2002.  
Recommended for acceptance by C. Kremer.  
For information on obtaining reprints of this article, please send e-mail to [tael@computer.org](mailto:tael@computer.org), and reference IEEE/CS Log Number 11359.

0888-360X/03/\$17.00 © 2003 IEEE  
Published by the IEEE Computer Society

“Our results, based on industry data from software developed in two popular programming languages [Java and C++] used in OO development, indicate that [...] these metrics are significantly associated with defects.”

CK metrics are often used to measure source code complexity (more details in other lectures for this course)

# Design Patterns

# Design Patterns

Alexander, 1977

Design patterns are general, reusable solutions to problems that commonly occur in software design

Jia, 2003

Design patterns form best practices that software engineers should use to solve common (recurring) problems when designing a software system

Wang, 2003

A design pattern is a recipe for solving a certain type of design problem that captures the high-level objects, their interactions, and their behaviors.

# Design Patterns

Alexander, 1977

Design patterns are general, reusable solutions to problems that commonly occur in software design

Jia, 2003

Design patterns form best practices that software engineers should use to solve common (recurring) problems when designing a software system

Wang, 2003

A design pattern is a recipe for solving a certain type of design problem that captures the high-level objects, their interactions, and their behaviors.

# Architectural Patterns

Higher-level structure of the system (broader scope)

How components are organized and assembled

Architectural pattern can influence the design patterns we can/will use

# Design Patterns

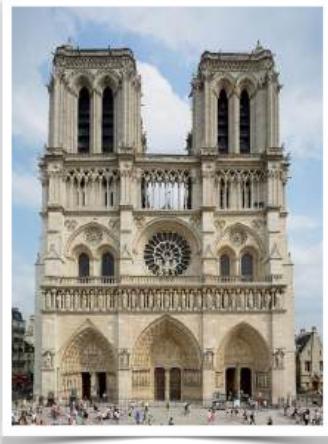
Lower-level scope

How individual components are built internally

Design patterns solve localized problems

# Architectural Patterns

Gothic Architecture

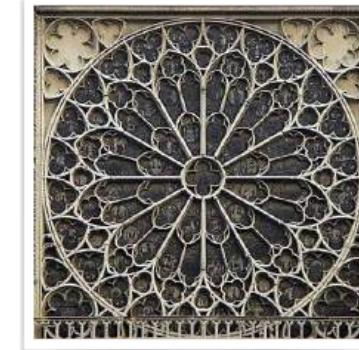


Baroque Architecture



# Design Patterns

Gothic Design Patterns



Baroque Design Patterns



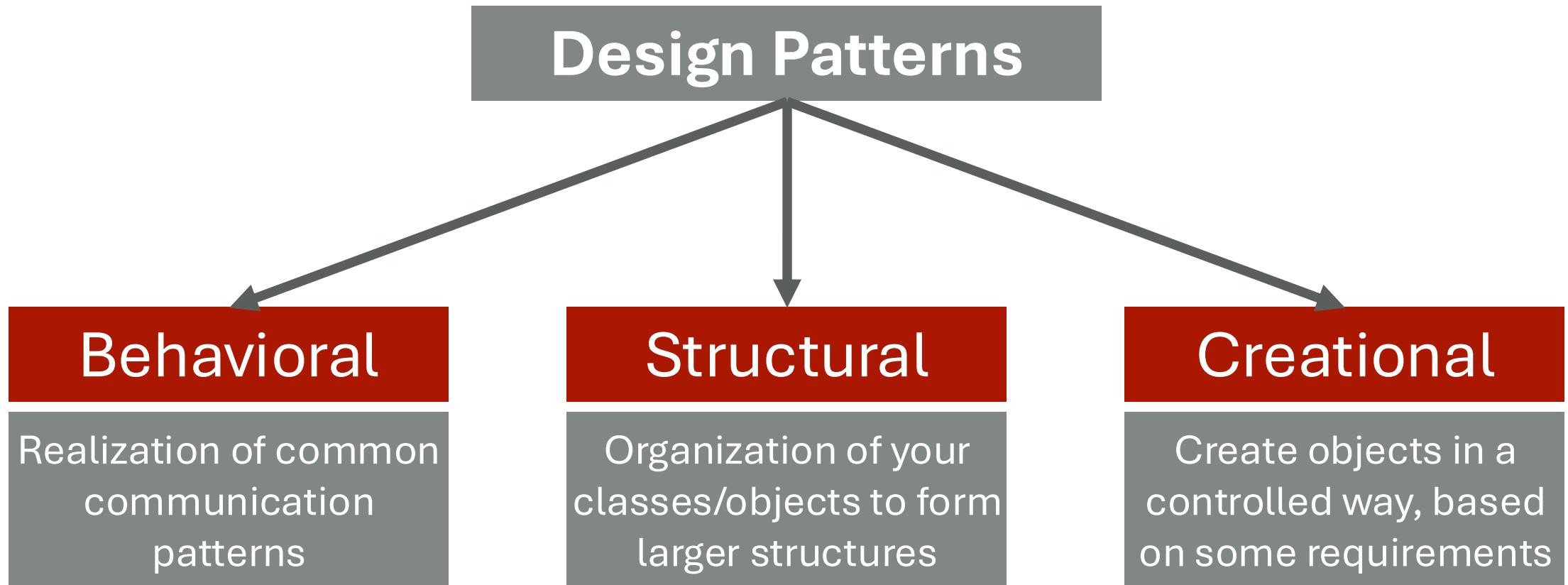
# Design Patterns

- Compare a pattern to a recipe in a cookbook;
  - Prescribed steps to get to the result
  - You are free to make changes/try new things
- Design Patterns consist of
  - Name
  - Intent
  - Target problem
  - Solution
  - Consequences

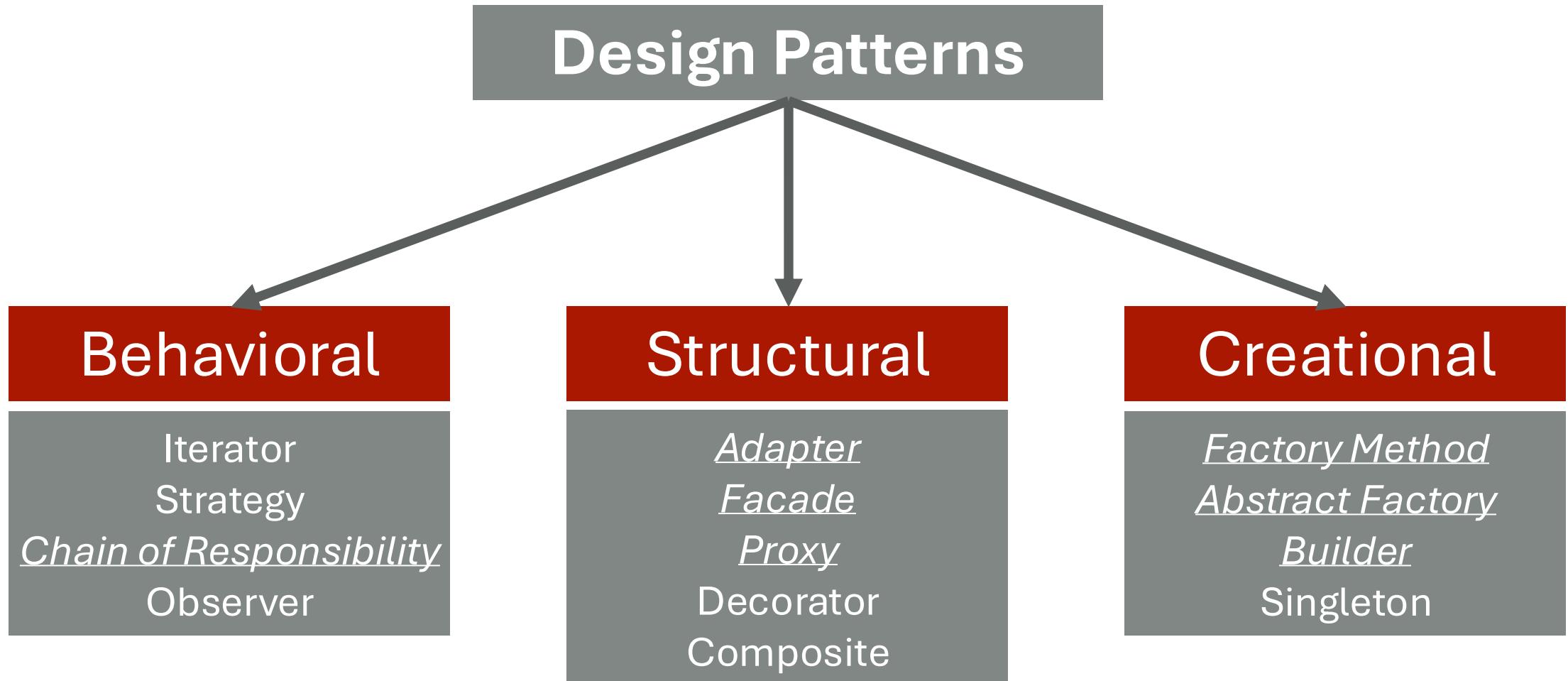


Design Patterns  
Erich Gamma, Richard Helm,  
Ralph Johnson, John Vlissides

# Design Patterns



# Design Patterns



# Disclaimers

- There are 23 design patterns in the original book by Gamma et al.
- We will study only 13 patterns using scenarios (problem examples), which are applicable for most programming language (not only to Java)
- More design patterns have been designed since the original book also for concurrency problems, and program-language specific features



The background features a large, solid blue circle on the left side. Behind it, there are two yellow bands: a wide one at the top and a narrower one at the bottom. The yellow band at the bottom has a subtle curve.

**SOLID**

# The SOLID Principles

The five design principles intended to make **object-oriented designs** more understandable, flexible, and **maintainable**:

- **Single responsibility principle**
- **Open-close principle**
- **Liskov Substitution principle**
- **Interface Segregation principle**
- **Dependency inversion principle**

Every class should have  
only one responsibility

# The SOLID Principles

The five design principles intended to make **object-oriented designs** more understandable, flexible, and **maintainable**:

- Single responsibility principle
- Open-close principle ←
- Liskov Substitution principle
- Interface Segregation principle
- Dependency inversion principle

“Software entities ... should be open for extension, but closed for modification”

# Liskov Substitute Principle

The Liskov Substitution Principle (**LSP**) was introduced by Barbara Liskov in 1987. It states that objects of a superclass should be replaceable with objects of their subclasses without affecting the correctness of the program

## Implications:

- **Subtype Requirement:** Subtypes must be substitutable for their base types.
- **Behavior Preservation:** Derived classes should extend the behavior of the base class without changing its behavior.
- **Inheritance Consistency:** Derived classes should not violate the contracts of the base class.



# The SOLID Principles

The five design principles intended to make **object-oriented designs** more understandable, flexible, and **maintainable**:

- Single responsibility principle
- Open-close principle
- Liskov Substitution principle
- Interface segregation principle
- Dependency inversion principle

“A client should not be forced to depend on interfaces it does not use.”

OR

“it's better to have many specific, smaller interfaces rather than one large, monolithic interface.”



# The SOLID Principles

The five design principles intended to make **object-oriented designs** more understandable, flexible, and **maintainable**:

- Single responsibility principle
- Open-close principle
- Liskov Substitute principle
- Interface segregation principle
- Dependency inversion principle



“Depend upon  
abstractions, not  
concretes”

# Design Patterns and SOLID

- The design patterns are designed to implement the SOLID principles:
- Some examples:
  - The **Strategy** pattern implements the DIP
  - The **Decorator** pattern implements the LSP
  - The **Strategy** pattern implements the open-close principle
- ...

# Exercises

# Which Design Pattern to Apply?

## Scenario:

We are implementing a video game (RPG). We want to model the ability of our super heroes to use different weapons.

A dragon slayer can use different weapons to slay a dragon. The available weapons include:

- A sword to sever the dragon's head
- Shoot the dragon with a magical crossbow
- Cast a spell to disintegrate the dragon
- ...



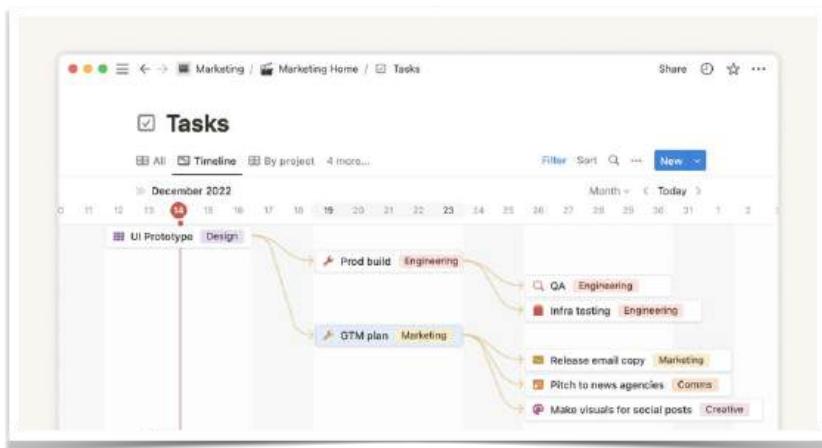
# Showcase

- Iterator + Iterable in modern Java (Java 8 +)  
using Functions

# Which Design Pattern to Apply?

## Scenario:

As part of developing a robust project management software for software development teams, our goal is to create a tool that enables efficient organization, tracking, and management of tasks and subtasks (aka nested tasks). The software should provide a structured approach to mark the status of **tasks** and all its **subtasks** (e.g., done, to be completed, ignored, etc.)



# Reverse Exercise

- Let's consider the **Chain of Responsibility**
- Describe a real-world application or system where the **Chain of Responsibility** pattern can be employed. Illustrate a scenario highlighting the benefits and advantages of using this pattern in that context.



# **Behavioral Patterns**

## **Iterator Pattern**

# Iterator Pattern

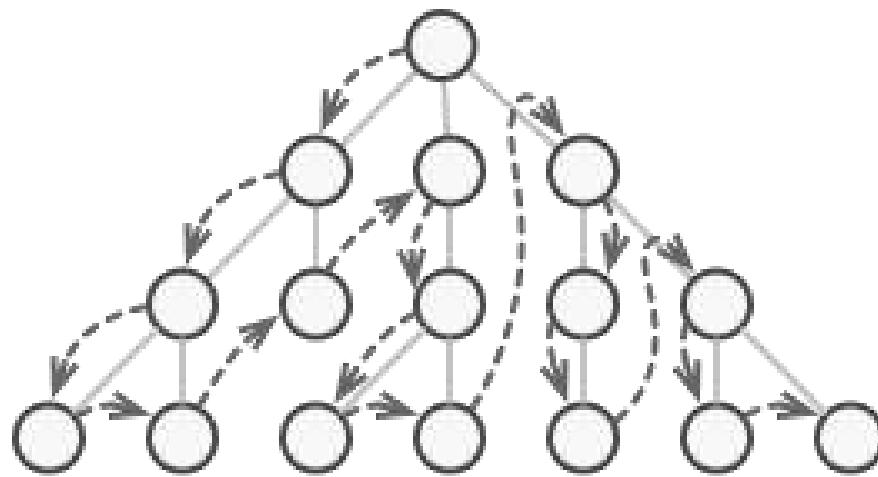
- Intent
  - Traverse elements of a collection without exposing underlying representation (list, stack, tree, ...)
- When
  - Iteration over your collection is necessary
  - One or multiple iteration methods possible

# Iterator Pattern

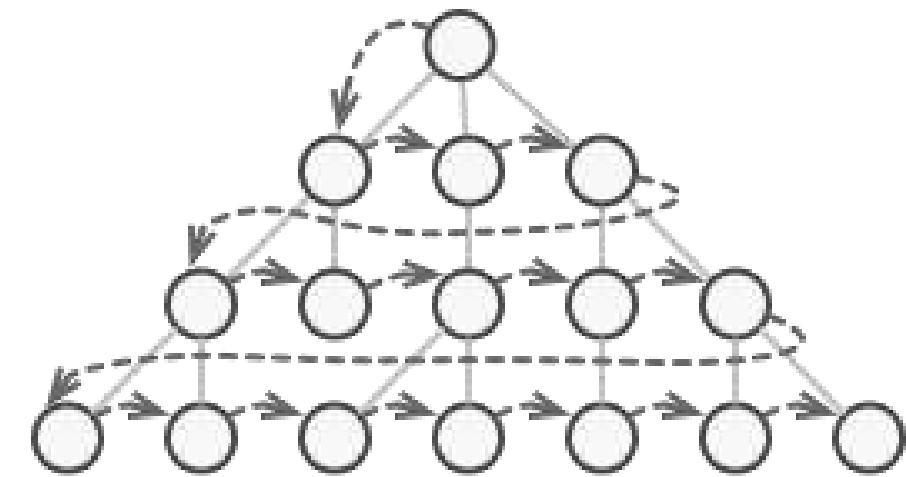
- Problem
  - Collections are common
  - Different internal representations
  - Usually we need a way to iterate over each element
  - Some collections may support multiple different iteration orders

# Iterator Pattern – Problem Example

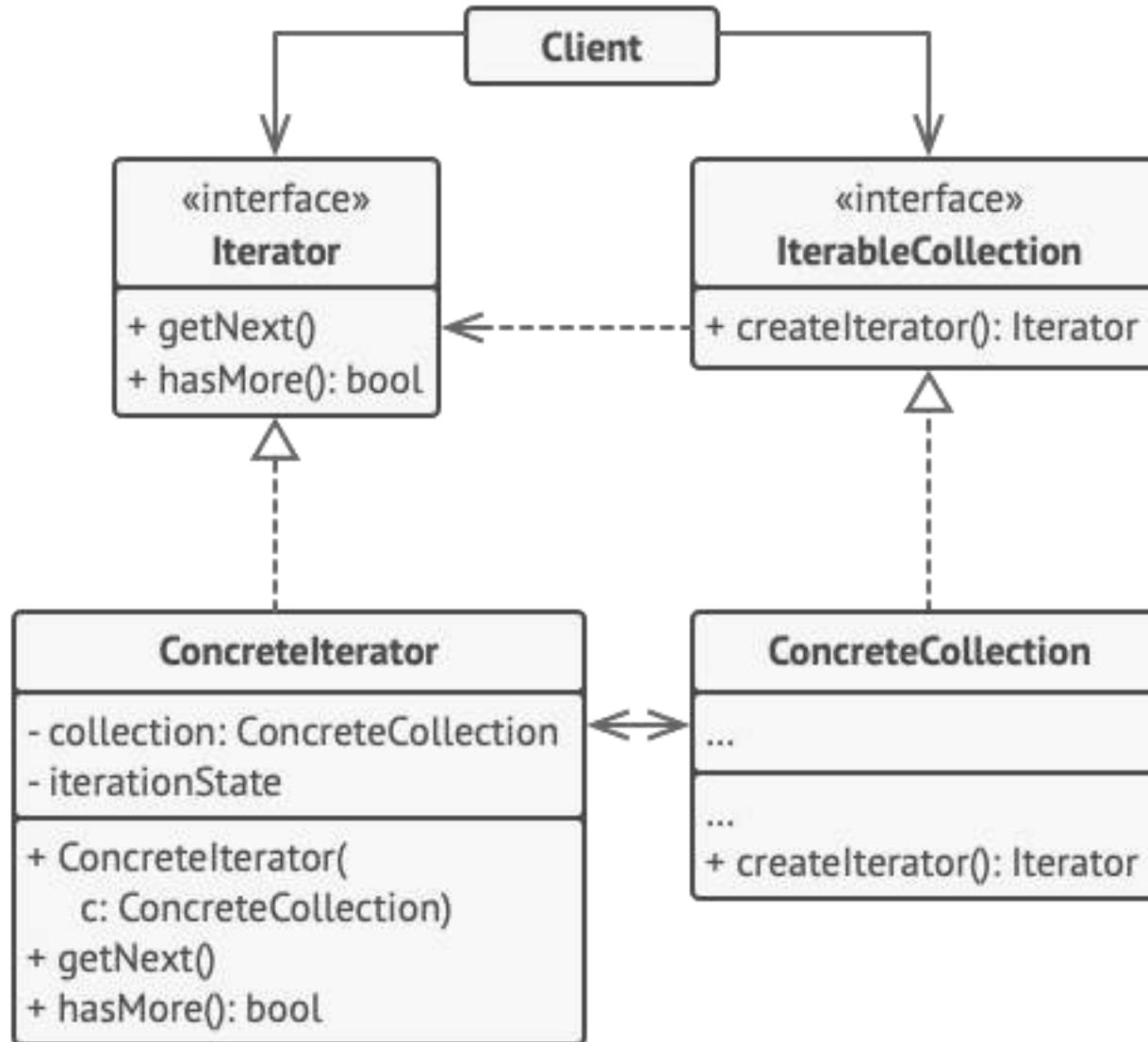
- Trees can be traversed DFS or BFS



Refactoring Guru, “[Iterator](#)”

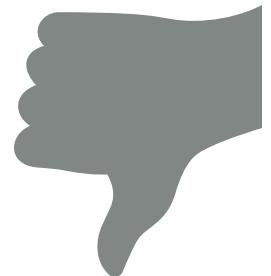


# Iterator Pattern



# Iterator Pattern

- Pros
  - SRP All traversal algorithms into separated classes
  - OCP Possible to add new collections and iterators and pass them to existing code
  - Parallel iteration is possible because each iterator has its own state
  - You can delay an iteration and continue when needed
- Cons
  - Overkill if your app only works with simple collections
  - Can be less efficient than going through elements directly for some specialized collections



# **Behavioral Patterns**

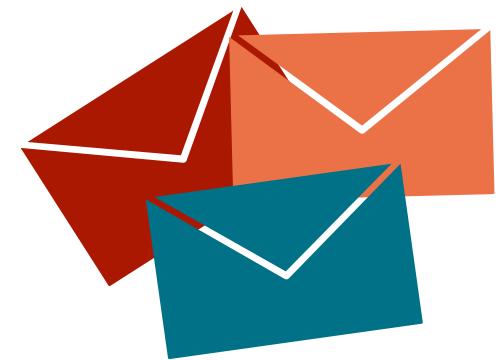
## **Strategy Pattern**

# Strategy Pattern

- Intent
  - Define a family of algorithms
  - Every concrete implementation in a separate class
  - All their objects are interchangeable
  - Algorithm can now vary independent from clients
- When
  - Many related classes differing only in behaviour
  - Need for different variants of an algorithm
  - Class defines behaviours that appear as multiple conditional statements in its operations

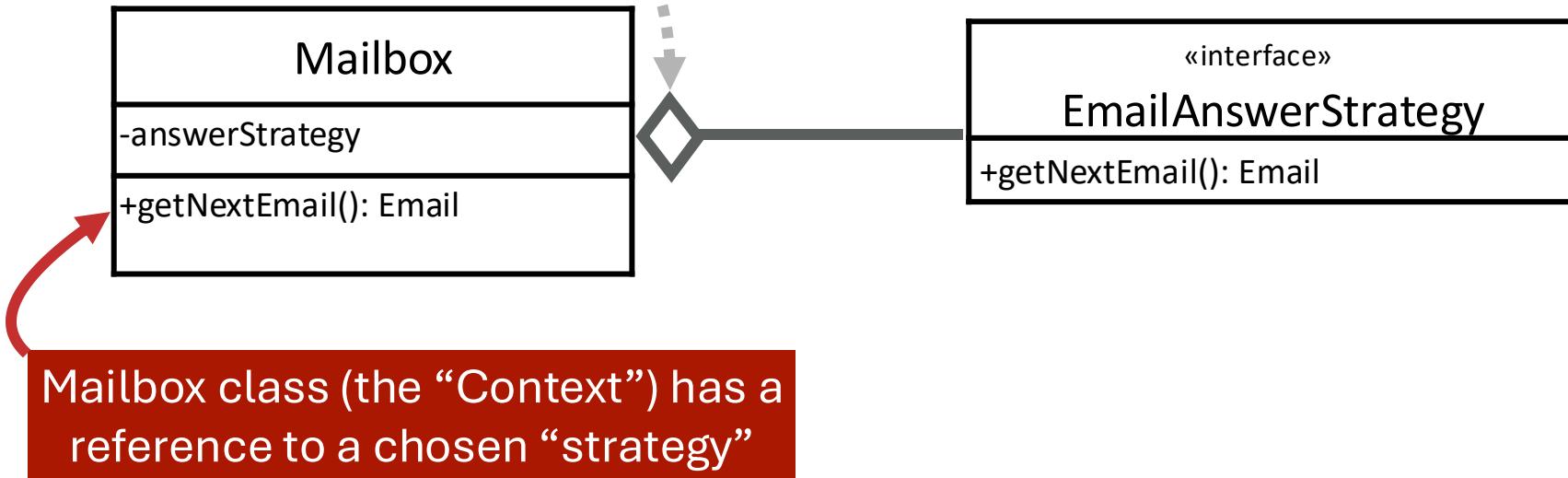
# Strategy Pattern – Problem

- Imagine a mailbox with a lot of mail coming in
- In what order do you process the mails?
  - Chronological?
  - Triage-based? (Assign priority and first process most important)
  - Money-based? (Those who pay more get faster replies)\*
- Strategy may change per year, per course, ...



# Strategy Pattern

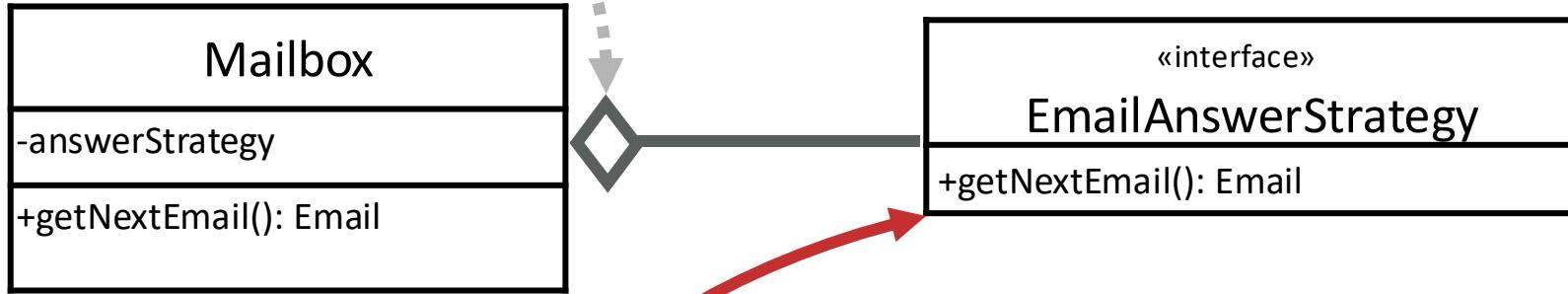
Can also be a composition  
when the context requires it.



UML does not require you to provide *all* details.  
Level of detail depends on reader, purpose, etc.

# Strategy Pattern

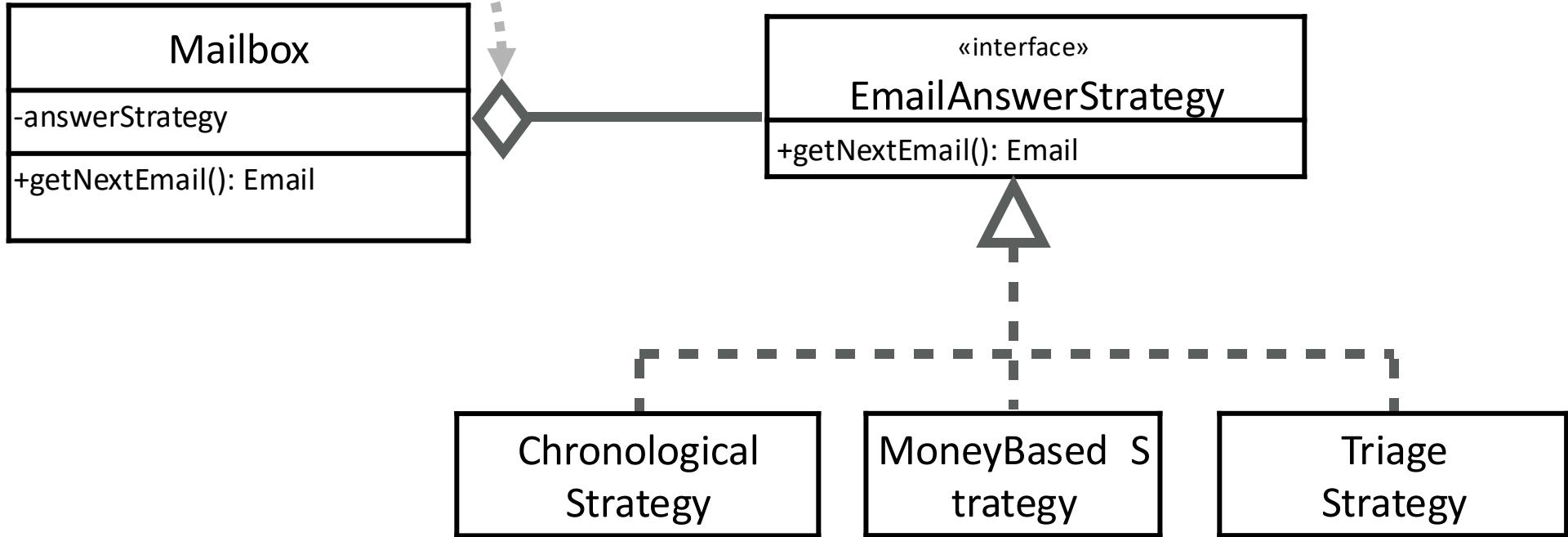
Can also be a composition  
when the context requires it.



The Strategy interface defines all operations that have different behaviors

# Strategy Pattern

Can also be a composition  
when the context requires it.



These classes implement the actual behaviors. They are called “Concrete Strategies”

# Strategy Pattern

- Pros
  - Run-time changing of behaviour within family of algorithms.
  - Uses interfaces as an alternative to sub-classing.
  - Eliminates conditional statements.
  
- Cons
  - Context must be aware of the strategy
  - Potential communication overhead between Strategy and Context
  - Increases number of objects



# **Behavioral Patterns**

## **Chain of Responsibility Pattern**

# Chain of Responsibility (CoR) Pattern

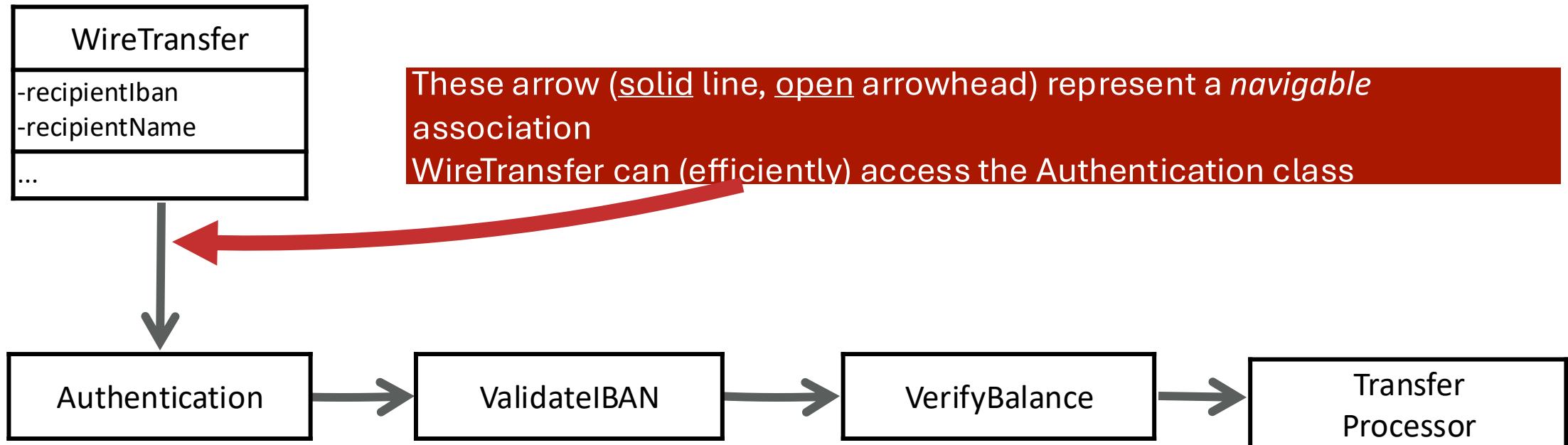
- Intent (A)
    - Pass requests along a chain of handlers
    - If any handler fails/rejects the request, propagation is stopped
  - When
    - More than one object needs to handle a request in sequence.
    - You want to issue a request to one of several objects, without specifying the receiver explicitly.
- There are 2 variants!

# CoR Pattern (A) – Problem

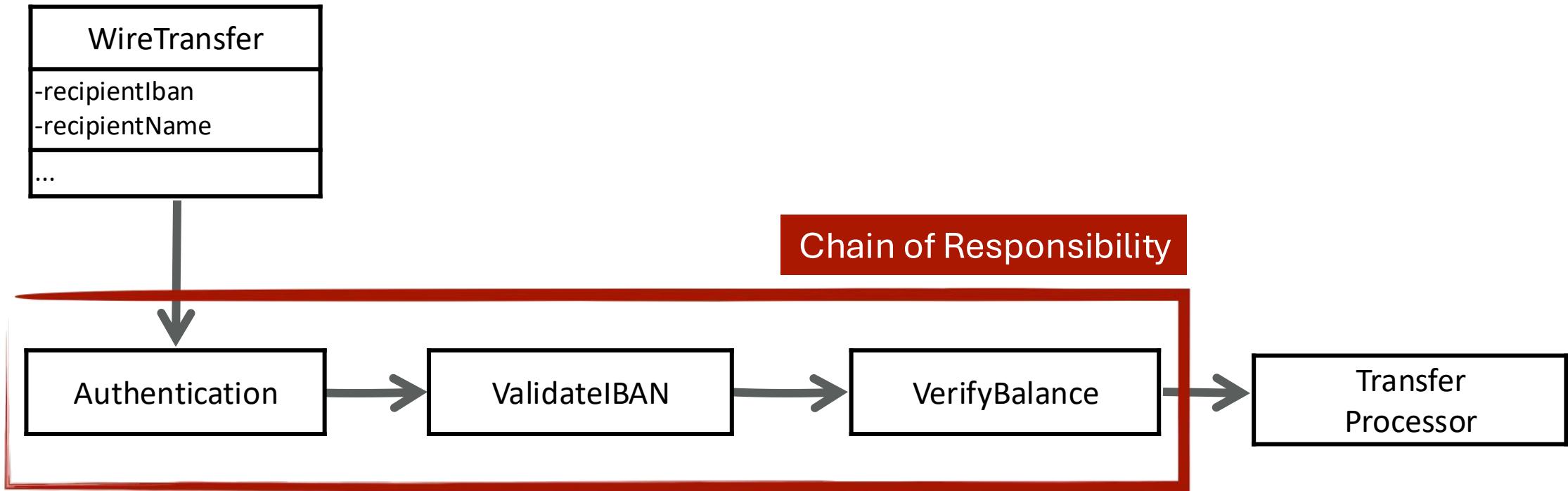
- We want to make a wire transfer to another bank account. There are multiple checks that need to be performed:
  - Authentication + Authorization
  - Validate the IBAN of the recipient
  - Verify that you have sufficient balance
- ...



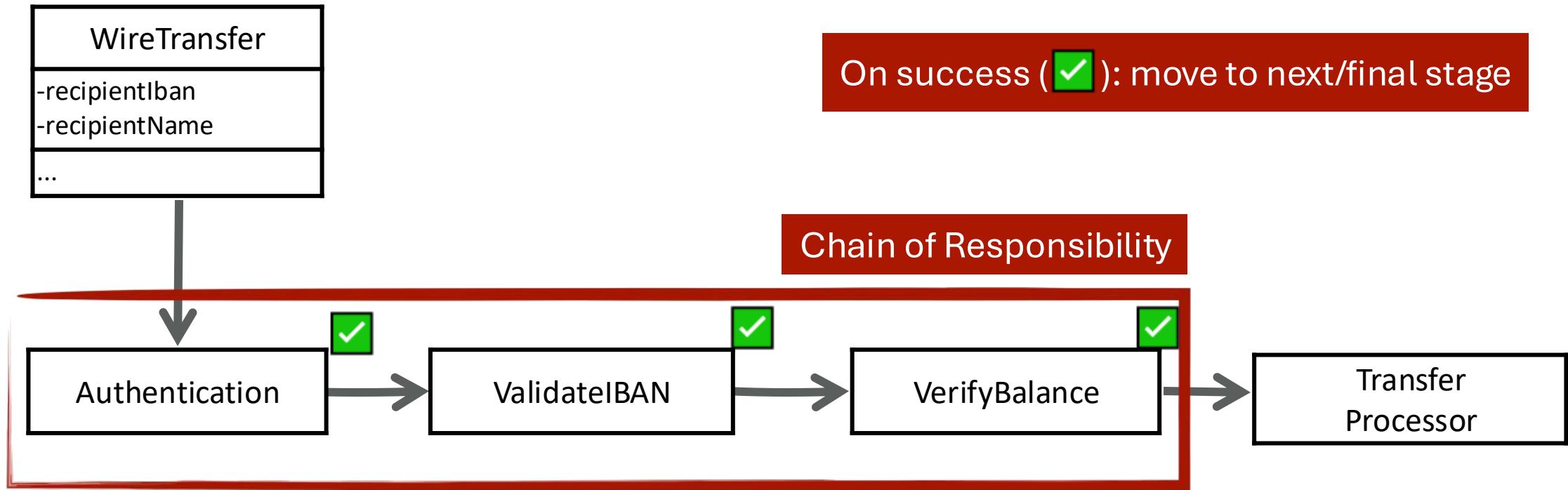
# CoR Pattern (A)



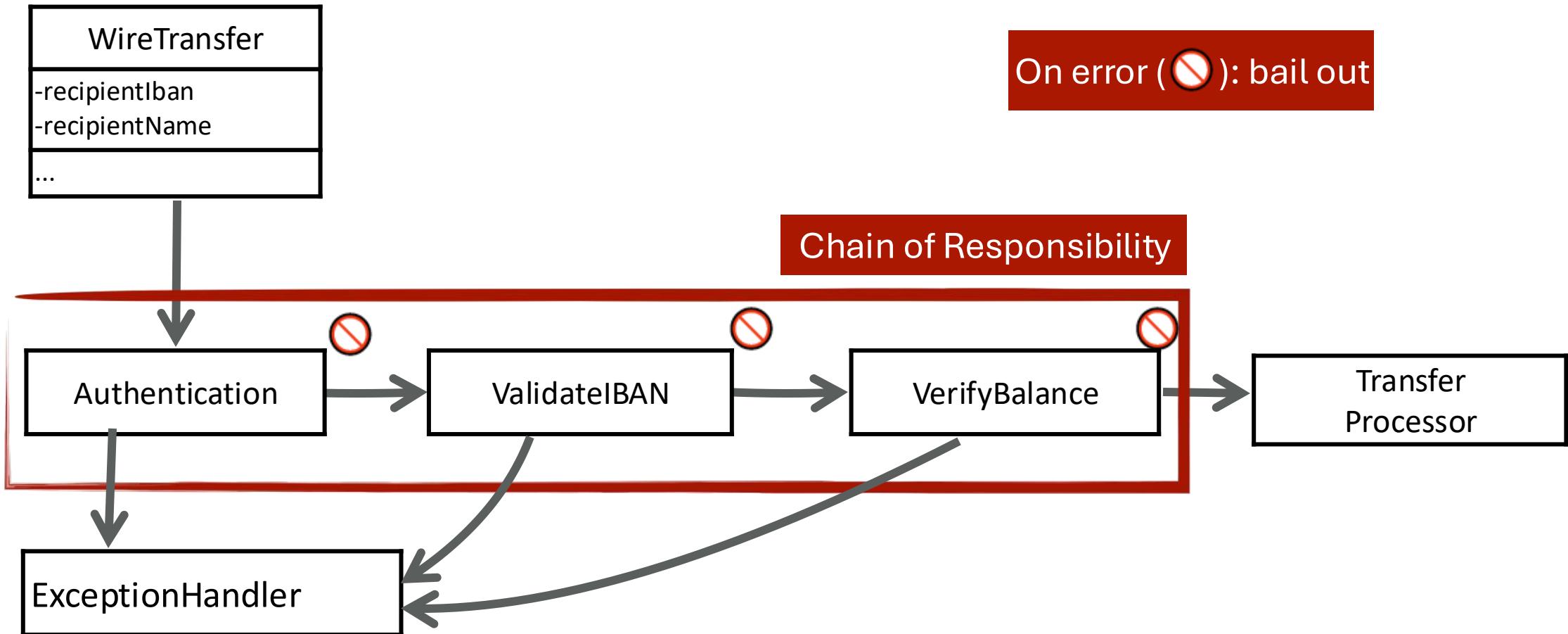
# CoR Pattern (A)



# CoR Pattern (A)

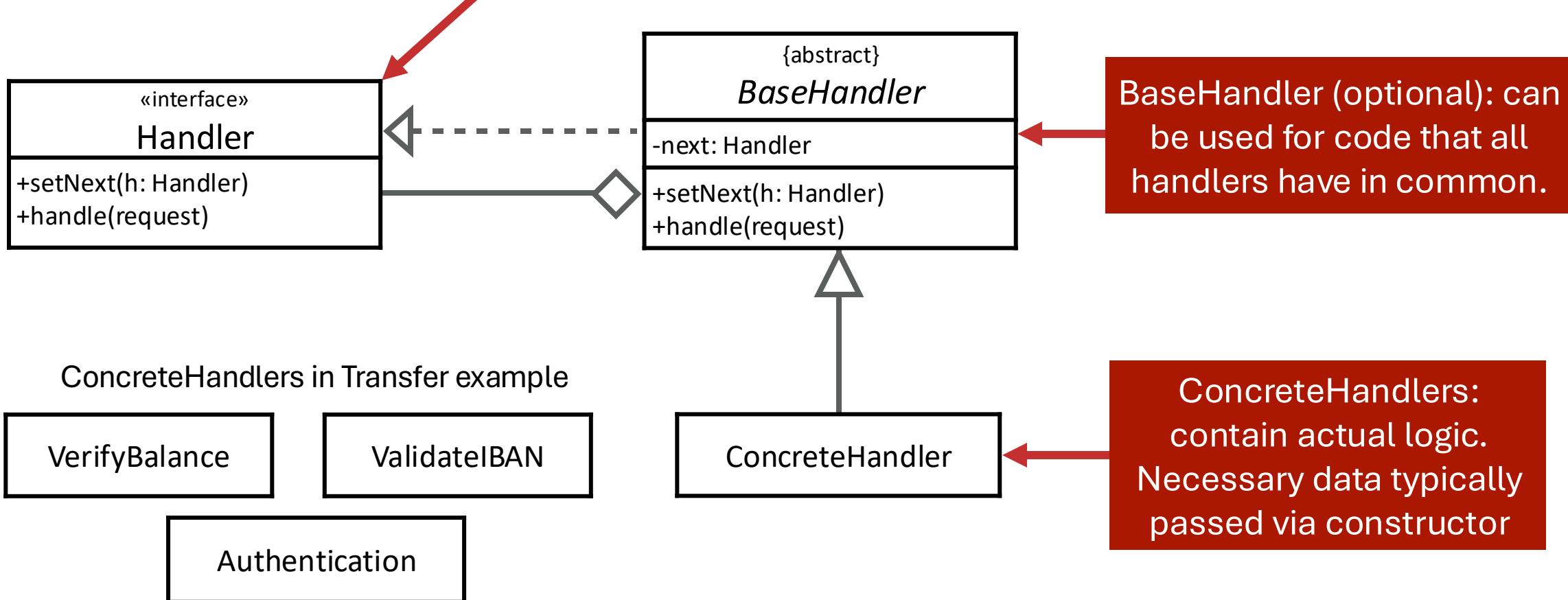


# CoR Pattern (A)



# CoR Pattern (A)

Handler defines the interface, typically has one method to proceed in chain. Can also have a method to change next handler.



# CoR Pattern (A) – Code

```
public interface Validator {  
  
    void setNext(Validator handler);  
    boolean handle(WireTransfer transfer) throws  
        InvalidTransferException;  
}
```

**Shared API for all Validators**

```
public abstract class BaseValidator implements Validator {  
    private Validator next;  
  
    public void setNext(Validator h){  
        this.next = h;  
    }  
  
    /**  
     * Runs check on the next object in chain or ends traversing  
     * if we are in last object in chain.  
     */  
    protected boolean checkNext(WireTransfer transfer) throws  
        InvalidTransferException {  
        if (next == null) {  
            return true;  
        }  
        return next.handle(transfer);  
    }  
}
```

**Handling the next element in the chain**

Notice that this is a simplified version of the classes used in the UML diagrams

# CoR Pattern (A) – Code

```
public class BalanceValidator extends BaseValidator {
    @Override
    public boolean handle(WireTransfer transfer) throws InvalidTransferException {
        // core logic to check the balance
        // ...
        if (transfer.getAmount() <= 0)
            throw new InvalidTransferException("Amount must be > 0.00");

        if (transfer.getAmount() > 200)
            throw new InvalidTransferException("Amount over the daily limit");

        return super.checkNext(transfer);
    }
}
```

**(ConcreteHandler)**

```
public class IBANValidator extends BaseValidator {
    @Override
    public boolean handle(WireTransfer transfer) throws InvalidTransferException {
        boolean valid = Modulo97.verifyCheckDigits(transfer.getIBAN());
        if (!valid)
            throw new InvalidTransferException("Invalid IBAN");

        boolean isSepa = IBAN.parse(transfer.getIBAN()).isSEPA();
        if (!isSepa)
            throw new InvalidTransferException("IBAN not from a SEPA-participating country");

        boolean isRegistered = IBAN.parse(transfer.getIBAN()).isInSwiftRegistry();
        if (!isRegistered)
            throw new InvalidTransferException("IBAN is not in the SWIFT Registry");

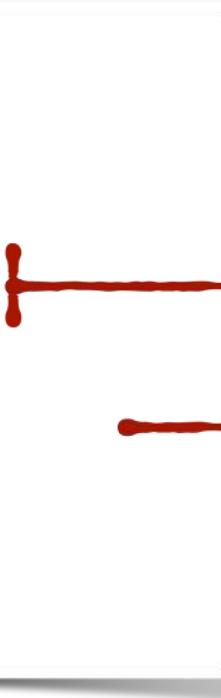
        return super.checkNext(transfer);
    }
}
```

**(ConcreteHandler)**

# CoR Pattern (A) – Code

*(Main Application)*

```
public class Application {  
    public static void main(String[] args) {  
        WireTransfer transfer = new WireTransfer(  
            "GB33BUKB20201555555555",  
            "Jane Smith",  
            150.00);  
  
        Validator handler = new IBANValidator();  
        handler.setNext(new BalanceValidator());  
  
        try {  
            boolean isValid = handler.handle(transfer);  
            System.out.print("Transfer status = " + isValid);  
        } catch (InvalidTransferException e) {  
            e.printStackTrace();  
        }  
    }  
}
```



*Creating the Chain*



*One method call to rule the chain*

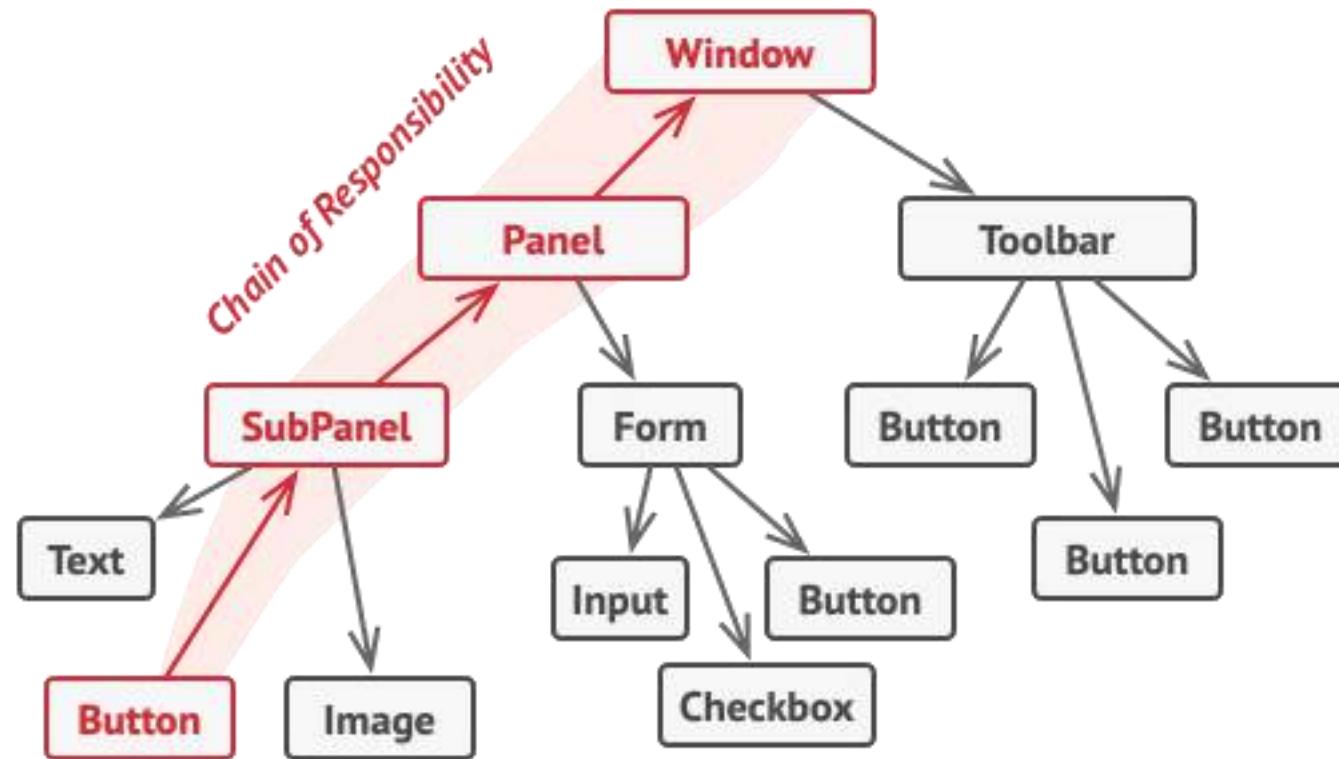


# CoR Pattern (B)

- Intent (B)
  - Pass requests along a chain of handlers
  - Each handler decides either to process the request or pass it to the next handler in the chain  
(continue down chain until handled)
- When
  - More than one object needs to handle a request, handler is not known a priori. Should be determined automatically/dynamically.
  - You want to issue a request to one of several objects, without specifying the receiver explicitly.

# CoR Pattern (B) – Problem

- JavaScript Event Handlers propagating up the DOM



# Chain of Responsibility Pattern

- Pros
  - Reduced coupling. No need for explicitly knowing who/how a request will be handled. Objects only know that requests are handled “appropriately”.
  - Receiver/sender don’t have to know each other.
  - Run-time changing of chain possible.
  
- Cons
  - No explicit receiver, request can be left unhandled when the chain is not configured properly.



# **Behavioral Patterns**

## **Observer**

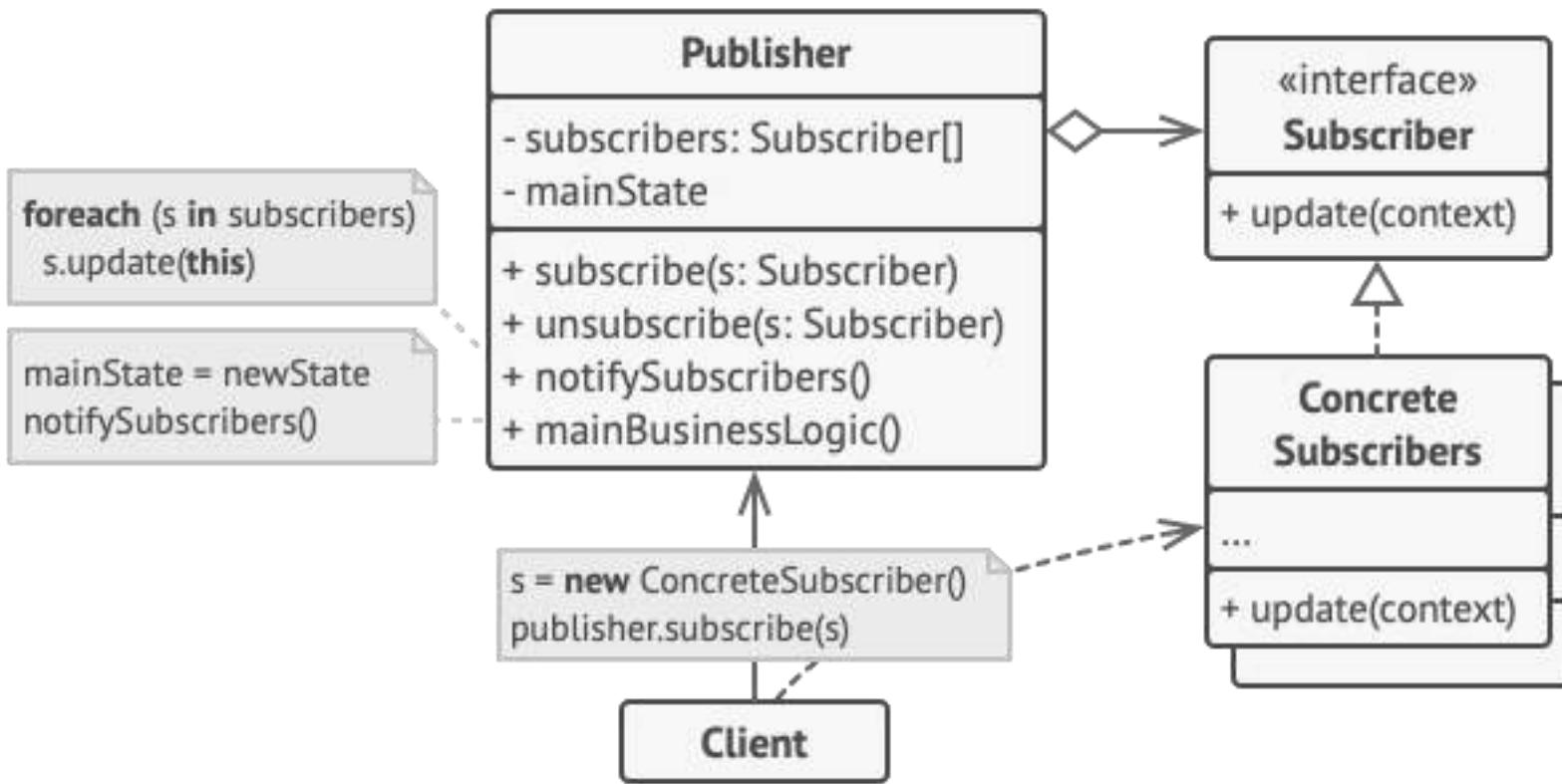
# Observer Pattern

- Intent
  - Define a subscription mechanism to notify multiple objects
- When
  - You want to inform others about updates or events
  - Order in which the subscribers are informed is not relevant

# Observer Pattern – Problem

- You run a store which sells phones
- A new iPhone has come out, it will soon be available in your store
- Customers come and check multiple times per day!
- You could send out an email to your customers when it is available, but this would annoy all Android users

# Observer Pattern



# Observer Pattern

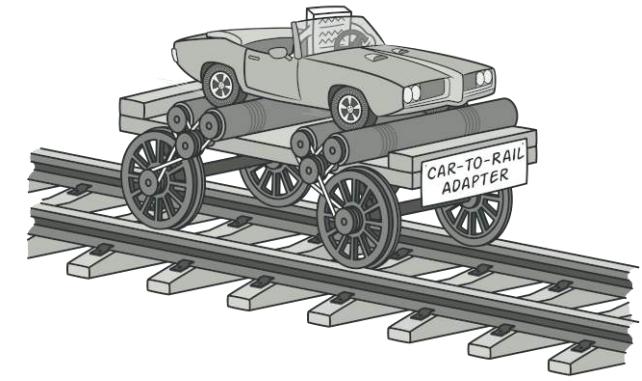
- Pros
  - OCP: You can introduce new subscriber classes without having to change the publisher code (and vice versa)
  - You can establish relations between objects at runtime
- Cons
  - Subscribers are notified in random order.



# **Structural Patterns**

## **Adapter Pattern**

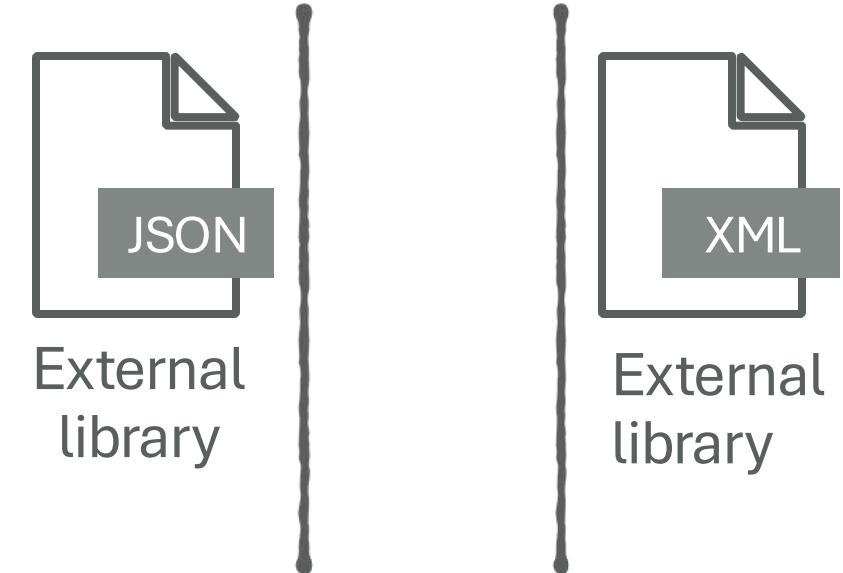
# Adapter Pattern



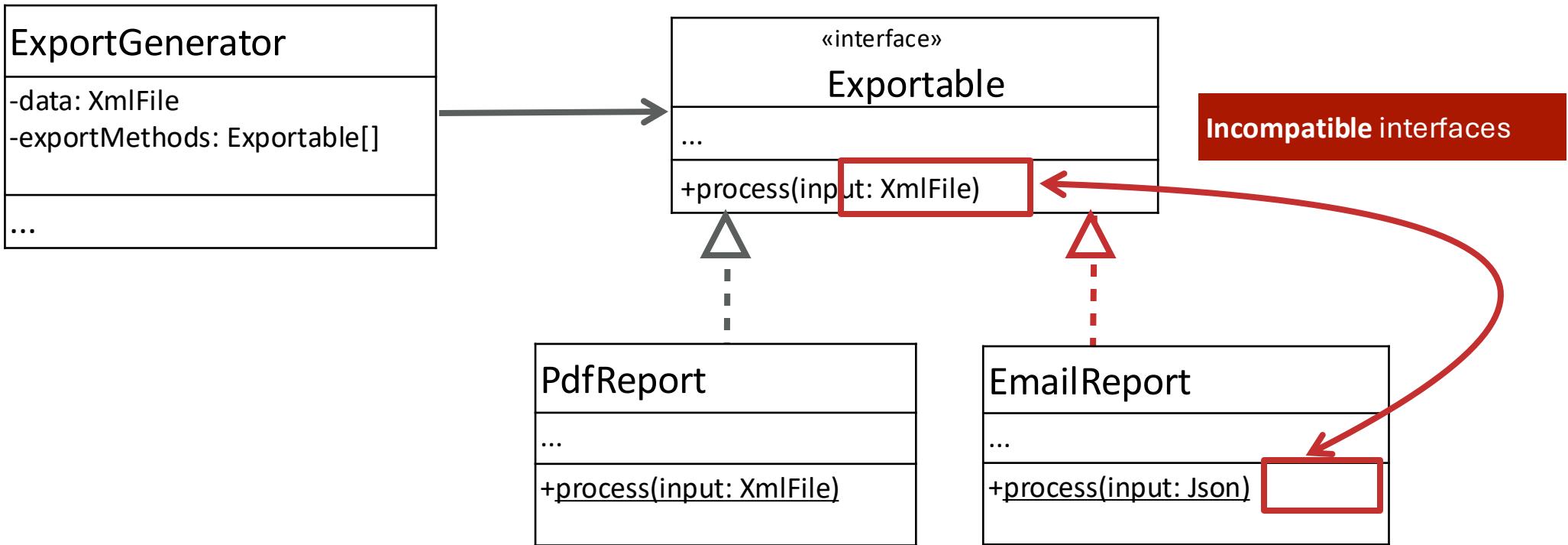
- Intent
  - Allow objects with incompatible interfaces to collaborate
- When
  - You want to use an existing class which has an incompatible interface.
  - You want to create a reusable class that cooperates with unrelated or unforeseen classes (classes that don't necessarily have compatible interfaces).

# Adapter Pattern – Problem

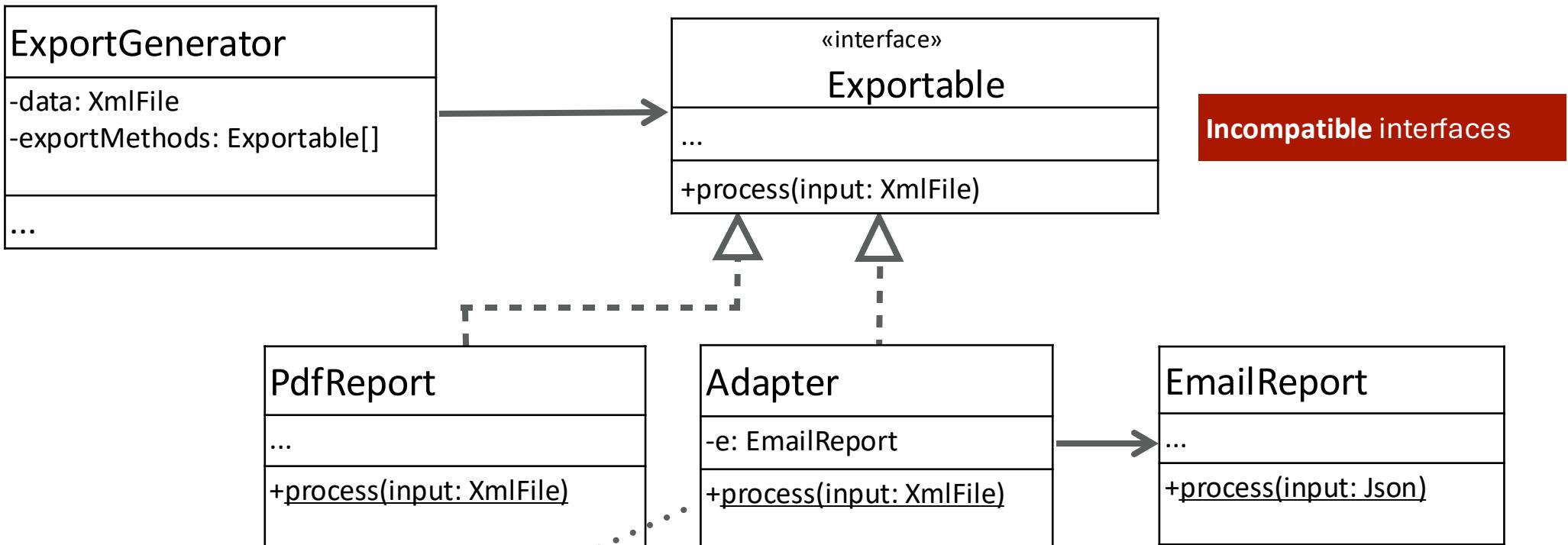
- Imagine a stock tracking app. You get input in the form of XML.
- Internally everything is XML and follows your self-defined interfaces.
- You want to make use of a new external service for reporting, but this service only accepts JSON format.



# Adapter Pattern



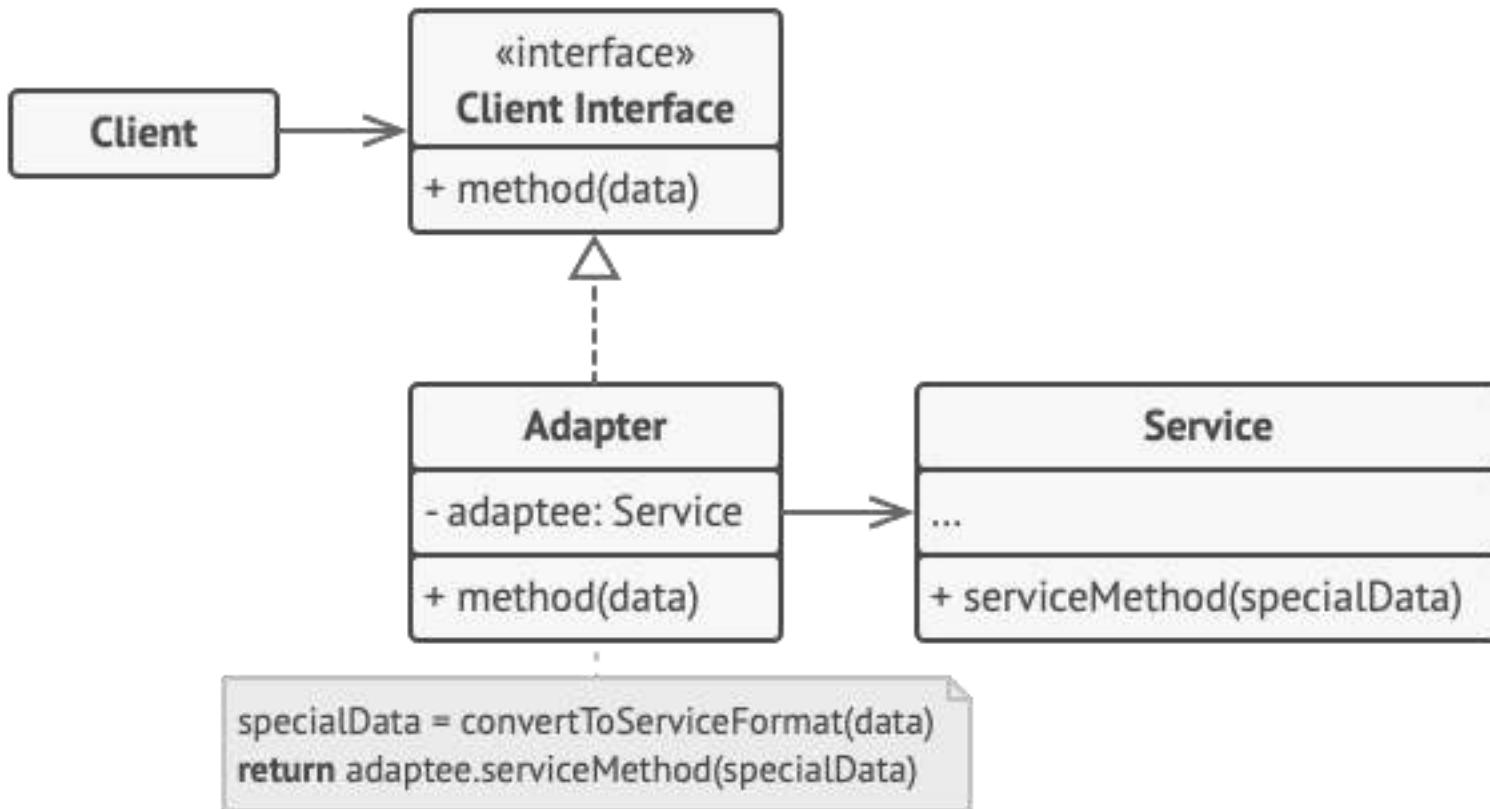
# Adapter Pattern



```
public void process(XmlFile input) {  
    Json json = transformXmlToJson(input);  
    this.e.process(json);  
}
```

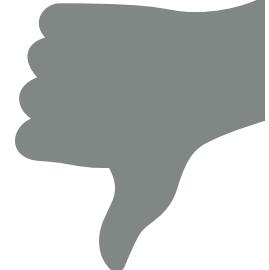
Adapt input and forward call

# Adapter Pattern



# Adapter Pattern

- Pros
  - Client doesn't know implementation details of the interface and/or how the data/code is adapted.
  - Helps reducing code duplication
- Cons
  - Adapter won't work when we want to adapt a class and all its subclasses.
  - One additional class.



# **Structural Patterns**

## **Facade Pattern**

# Facade Pattern

- Intent
  - Provide a simplified interface to a library, framework or other complex set of classes
- When
  - You want to provide a simple interface to a complex subsystem.
  - There are many dependencies between clients and the implementation classes of an abstraction. Introduce a facade to decouple the subsystem from clients and other subsystems.
  - You want to layer your subsystems. Facades can be entry points to each subsystem level.

# Facade Pattern – Problem

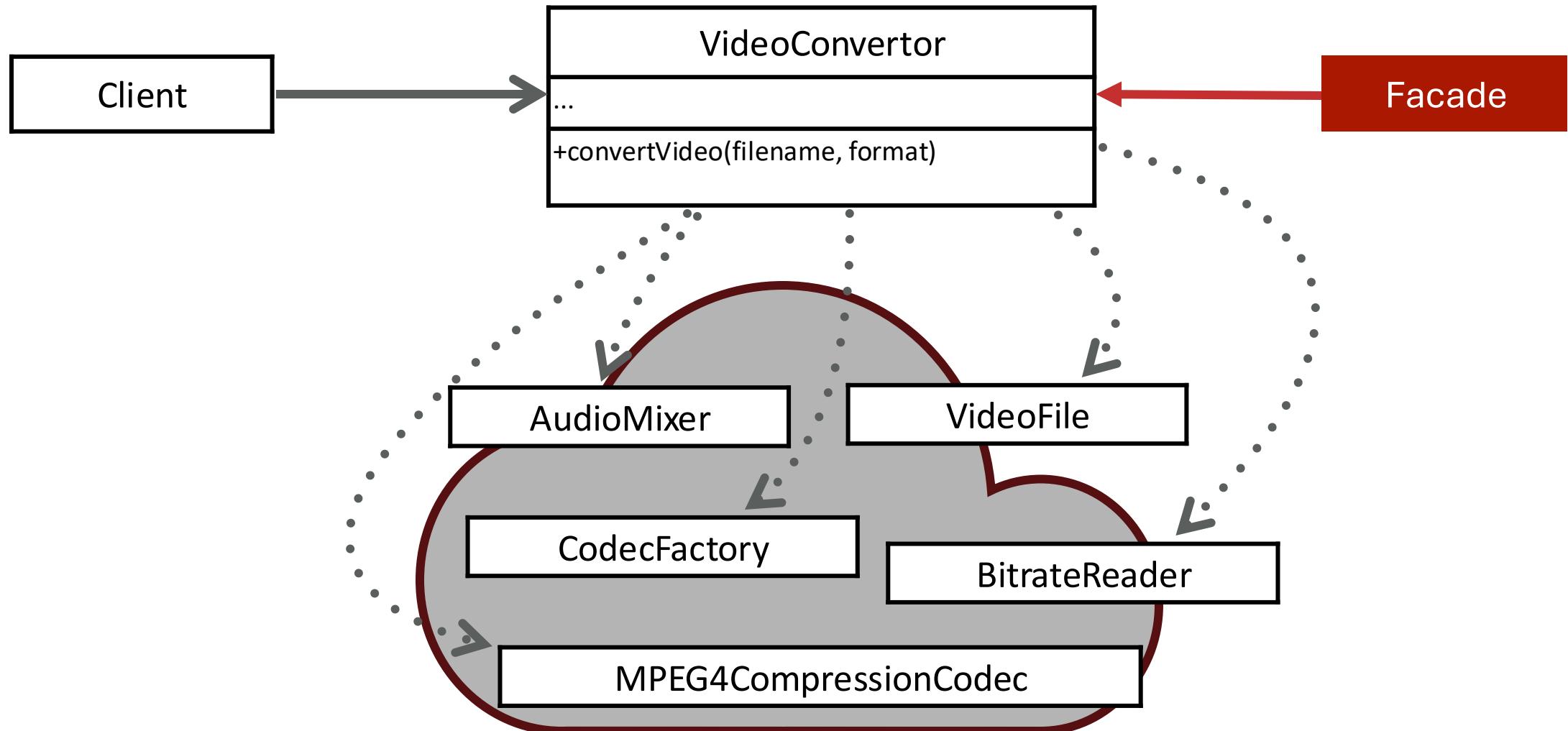
- You have a quite complex software system that can perform many different video conversion operations.
- As a developer you do not care about the internals, you just want a simple interface to talk to.

**What it is behind!**

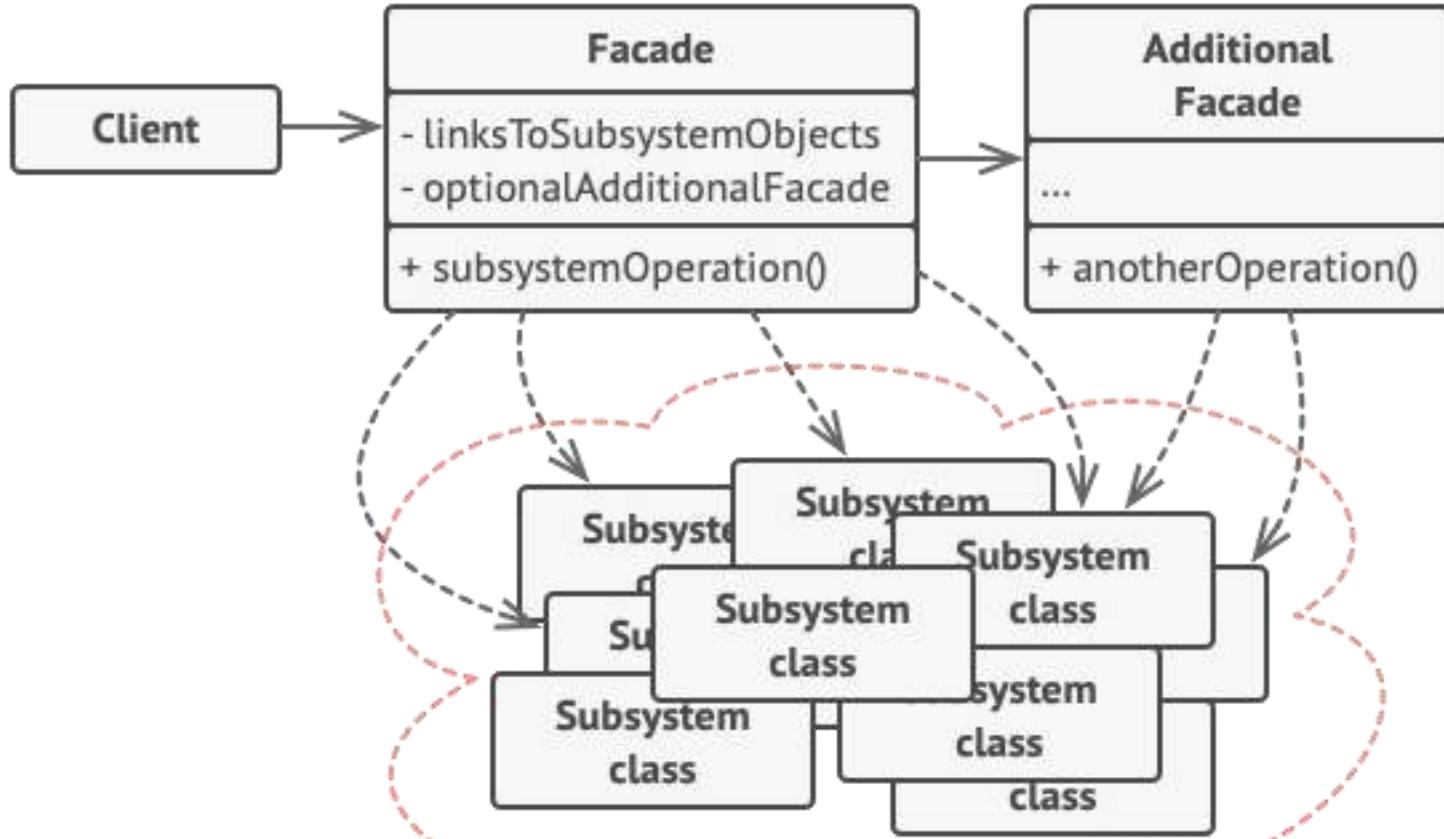


**What you see!**

# Facade Pattern

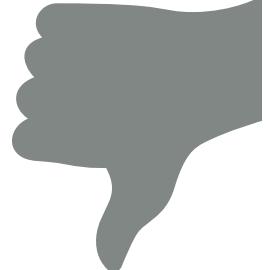


# Facade Pattern



# Facade Pattern

- Pros
  - Helps isolating clients from subsystems
  - Supports the open-closed principle
- Cons
  - Facades can become god-classes where they are coupled to all classes of the application.



# **Structural Patterns**

## **Proxy Pattern**

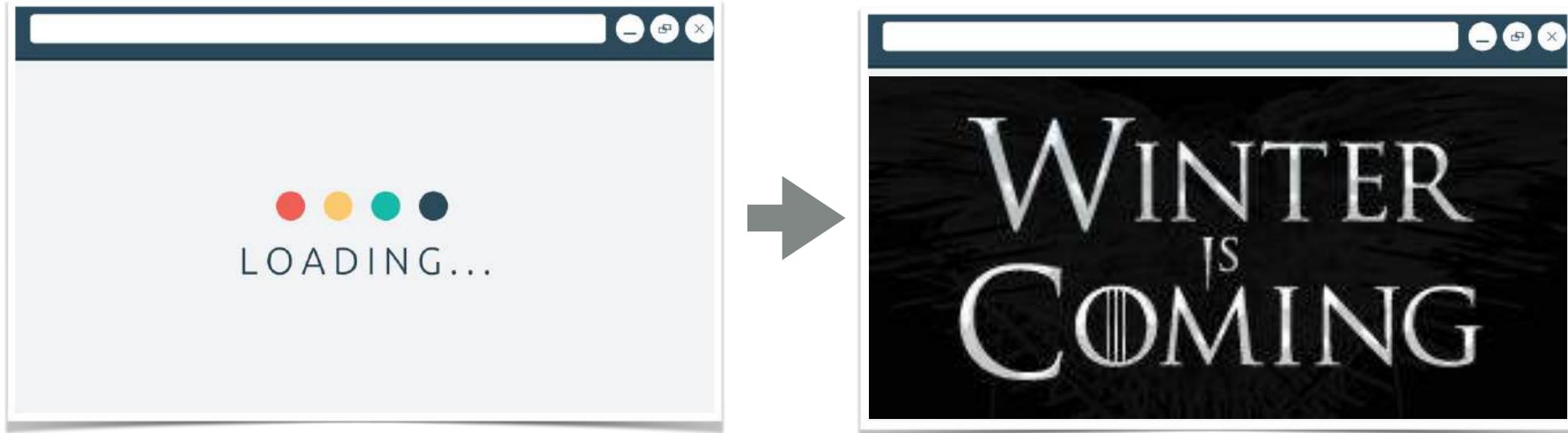
# Proxy Pattern

- Intent
  - Provide a substitute or placeholder for another object
  - Control access to the original object, allowing you to perform something before or after the request gets through to the original
- When
  - **(Remote Proxy)** Local object represents a remote object
  - **(Virtual Proxy)** Lazily load the real object on demand, only provide skeleton representation by default
  - **(Protection Proxy)** Control access to a resource based on access rights
  - **(Caching Proxy)** Cache results of client requests that are large and expensive to generate. The proxy also manages the lifecycle.

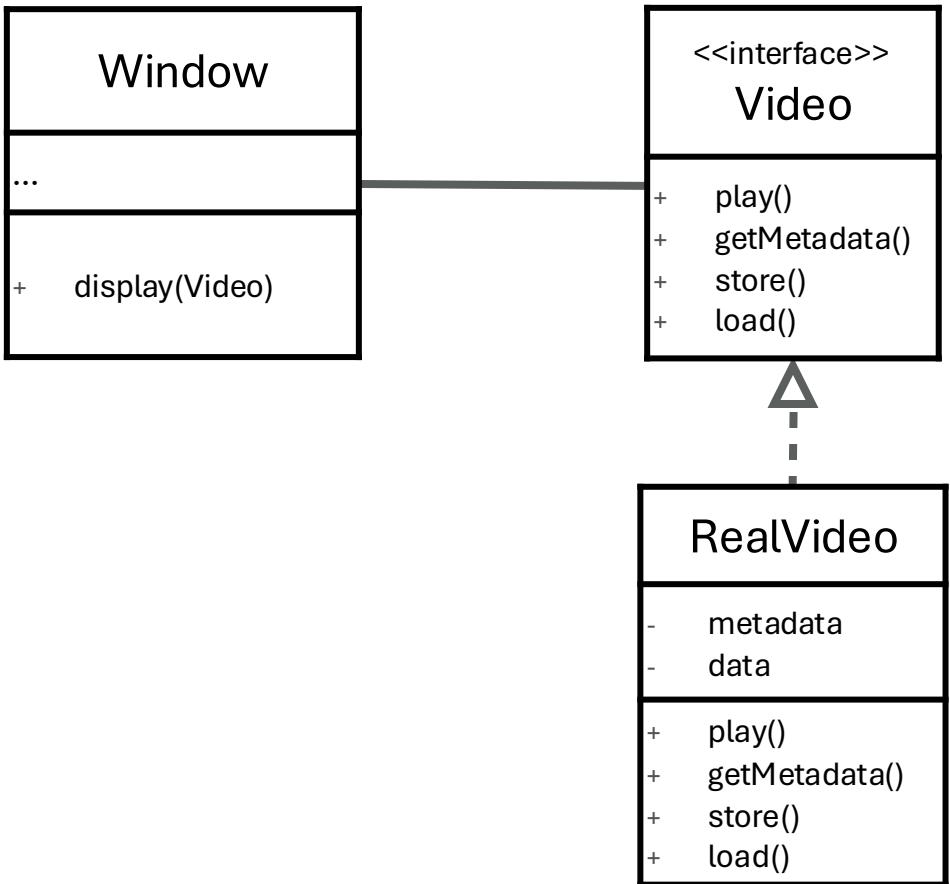
# Proxy Pattern – Problem

Our software system uses many **high-definition videos** that are stored on the server hard disk. We want to implement a class that **loads** and **displays** the videos when needed.

However, keeping the videos in-memory is not efficient as we won't need to display them in every GUI Window. Besides, we want to avoid multiple loading of the videos, accessing them in a memory-saving manner.



# Proxy Pattern – A Bad Solution

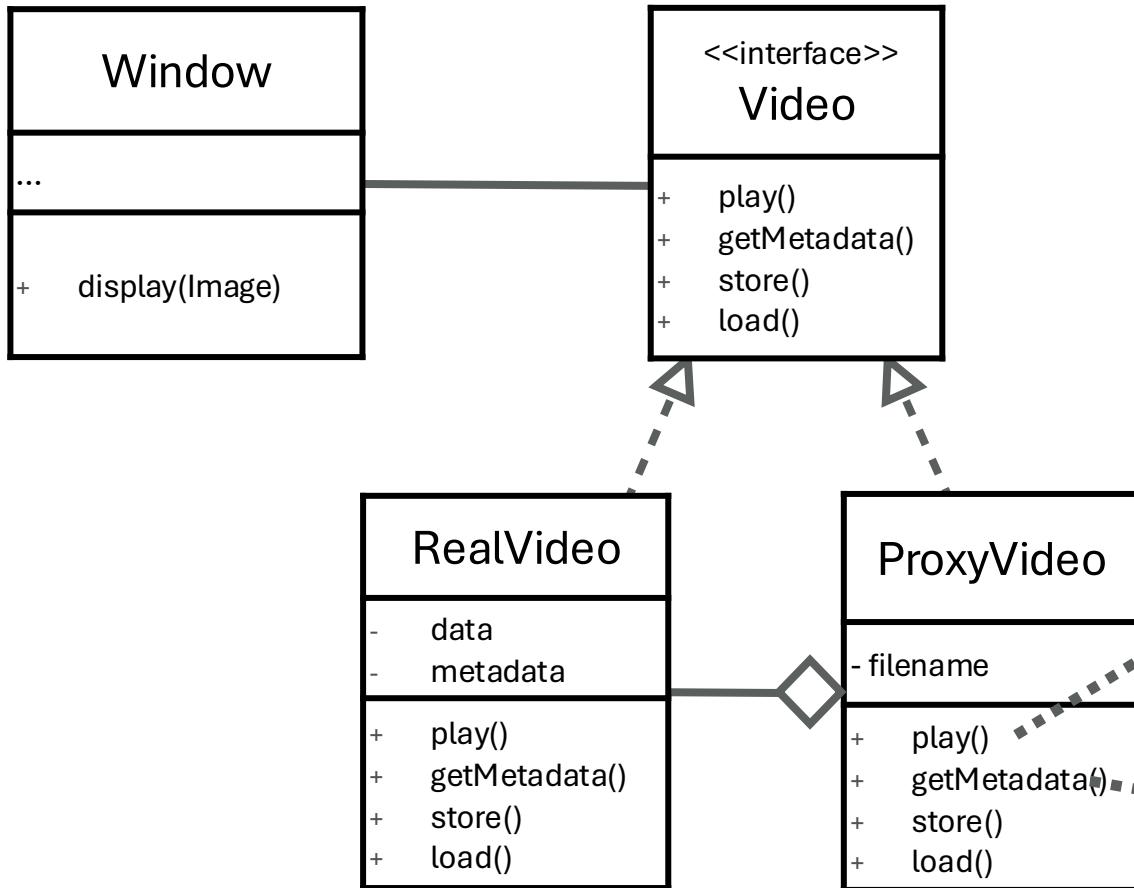


In this implementation, a video can be loaded in memory more than once.

Once loaded, the video remains in memory even if we may not need it for a while.

How can we keep a reference to the class **RealVideo** but display the actual video only on demand?

# Proxy Pattern – A Good Solution



*Window creates objects of the class **ProxyVideo**. The proxy loads the video on demand (lazy initialization)*

```
if (realVideo==null){  
    realVideo = loadVideo(filename);  
}  
realVideo.display();
```

```
if (realVideo==null){  
    return metadata;  
}  
return realVideo.getMetadata();
```

# Proxy vs. Other Patterns

## Proxy vs. Adapter

- The proxy provides the same interface of the wrapped object.
- The adapter provides a different interface to the wrapped object.

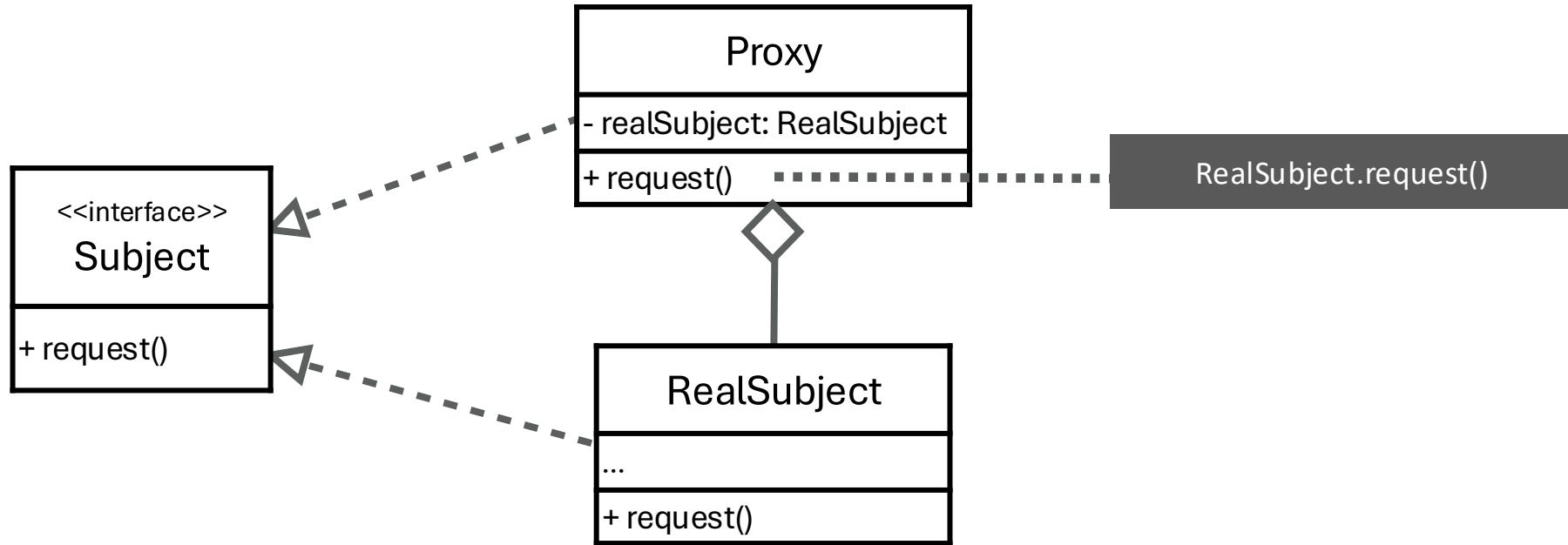
## Proxy vs. Decorator

- The proxy provides the same interface of the wrapped object.
- The decorator provides an enhanced interface.

## Proxy vs. Facade

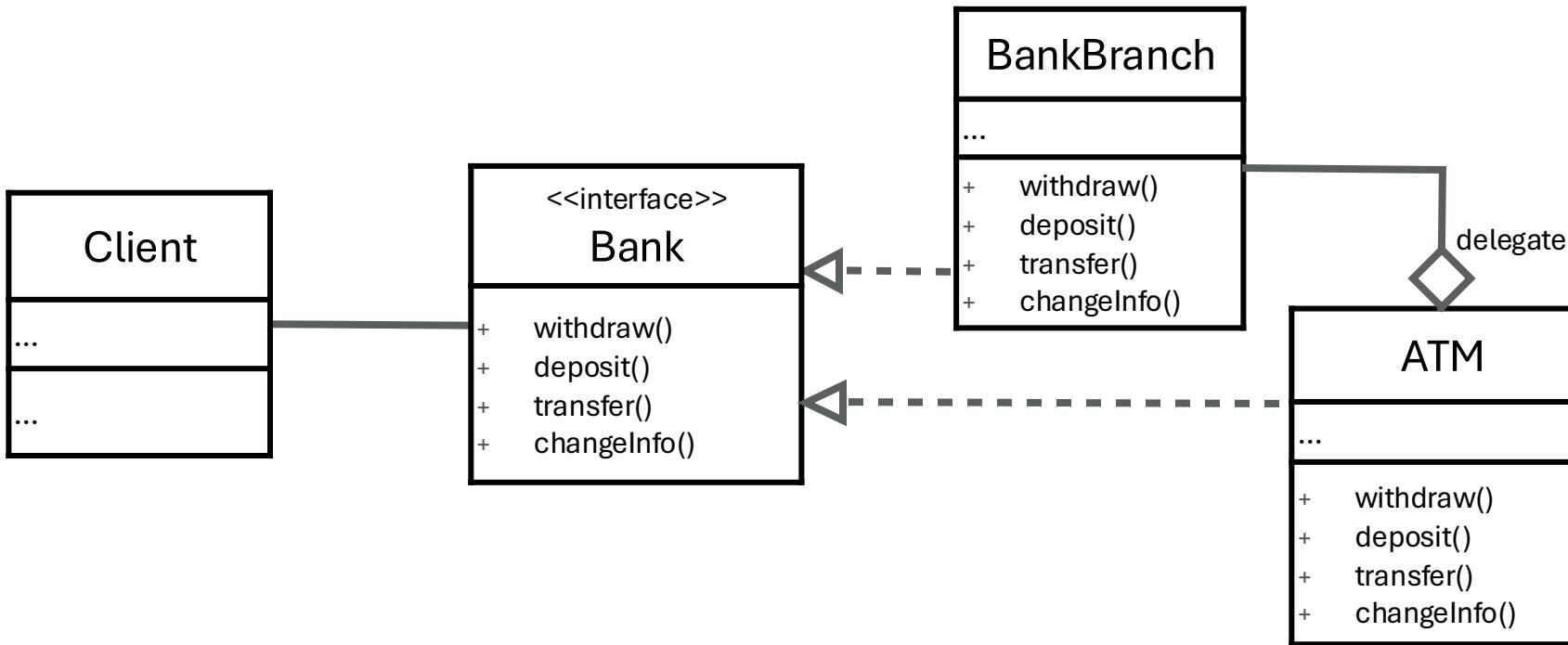
- The proxy provides the same interface of the wrapped object.
- The facade provides a simplified interface to the other objects.

# Proxy Pattern



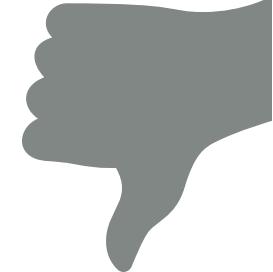
# Remote Proxy Example

To withdraw some cash we go to the nearest ATM and get the cash. The ATM forward the request (withdraw) to the actual bank. Therefore, the ATM is a **remote proxy** for the actual bank: the client makes a request, which is forwarded to a remote system (the bank).



# Proxy Pattern

- Pros
  - Control proxied object without clients knowing about it
  - You can manage the lifecycle of the proxied object when clients don't care about it
  - The proxy works even if the proxied object is not ready or unavailable
  - OCP: You can introduce new proxies without changing the proxied object or clients
- Cons
  - Code may be more complex since you need new classes
  - Additional latency on response due to wrapping

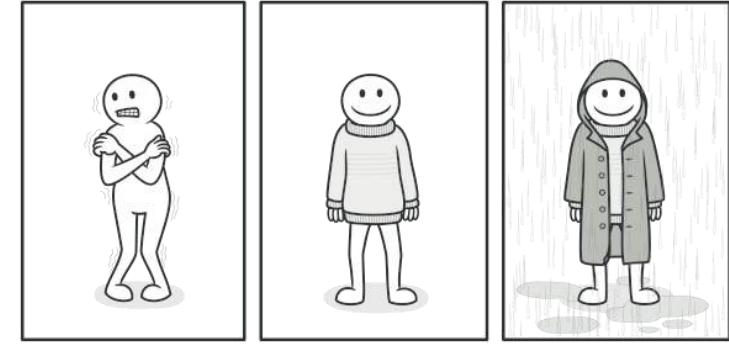


# **Structural Patterns**

## **Decorator Pattern**

# Decorator Pattern (Wrapper)

- Intent
  - Attach new behaviors by objects by wrapping them in special wrapper objects
- When
  - You want to add responsibilities to individual objects dynamically and transparently, that is, without affecting other objects.
  - You have responsibilities that can be withdrawn.
  - Extension by subclassing is impractical.

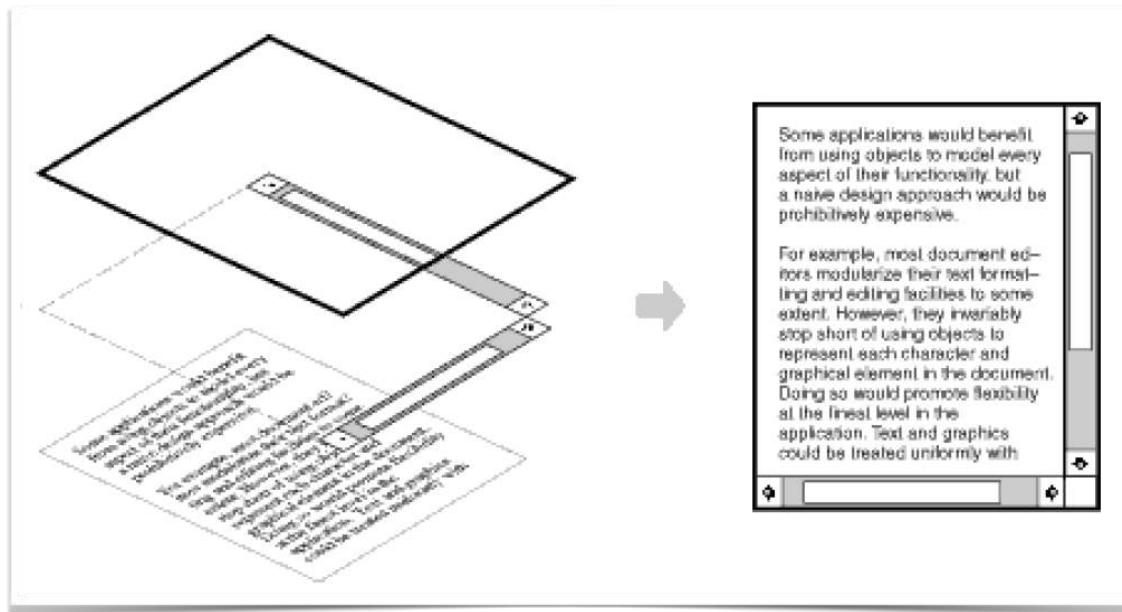


Refactoring Guru, “Decorator”

# Decorator Pattern – Problem

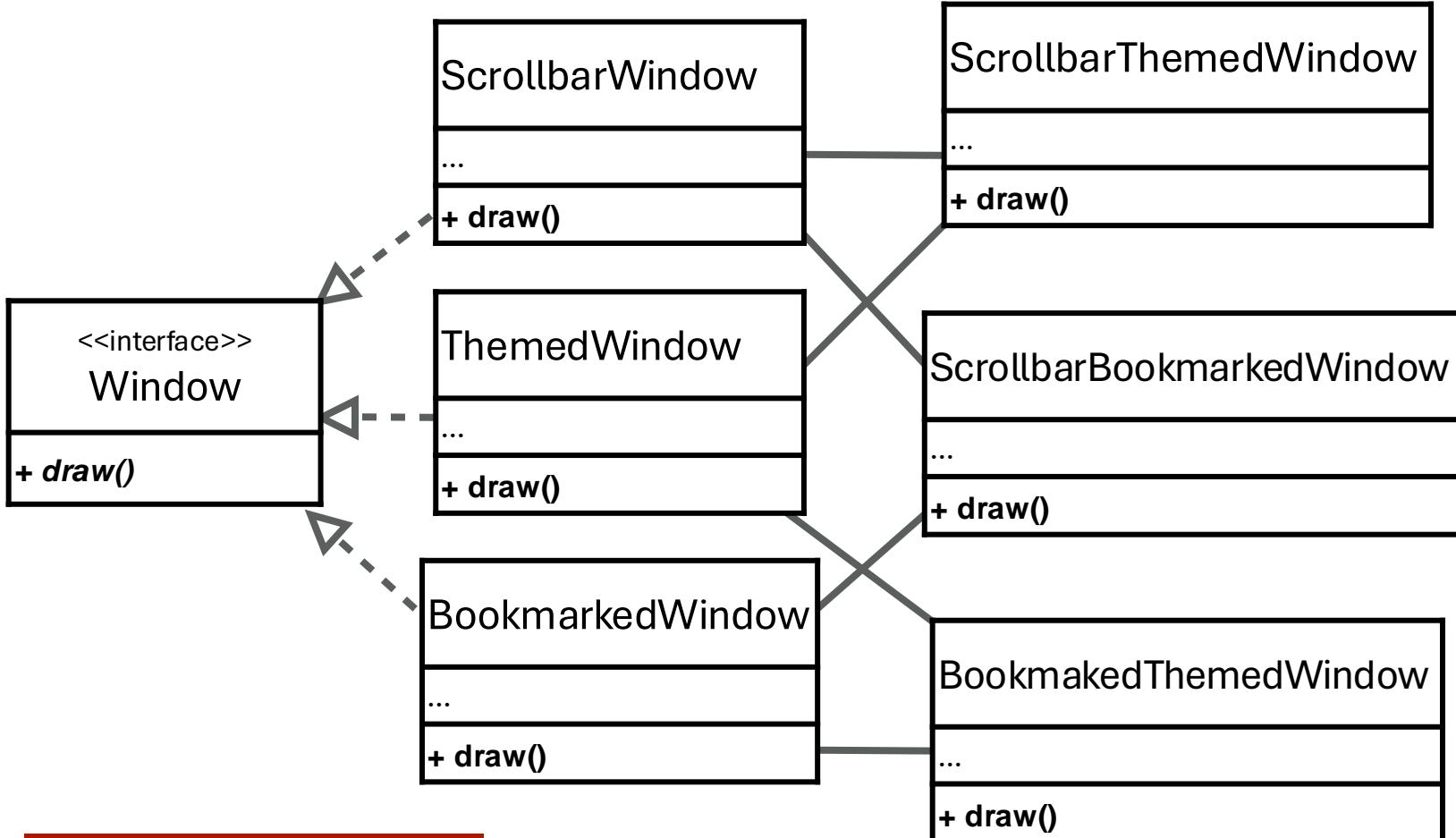
We want to develop an **Editor** that displays textual documents using GUI. The **look** of the Editor should **change** depending on the context, such as the length of the text, the theme (background color and text color) of the editor, if the text contains chapters, as so on.

By default, the editor has no scroll bars because we might not always need them (scroll-bars not always apply). We also want to be able to change the theme, add bookmarks, and so on.



Example inspired by book by Gamma et al.

# A bad Solution



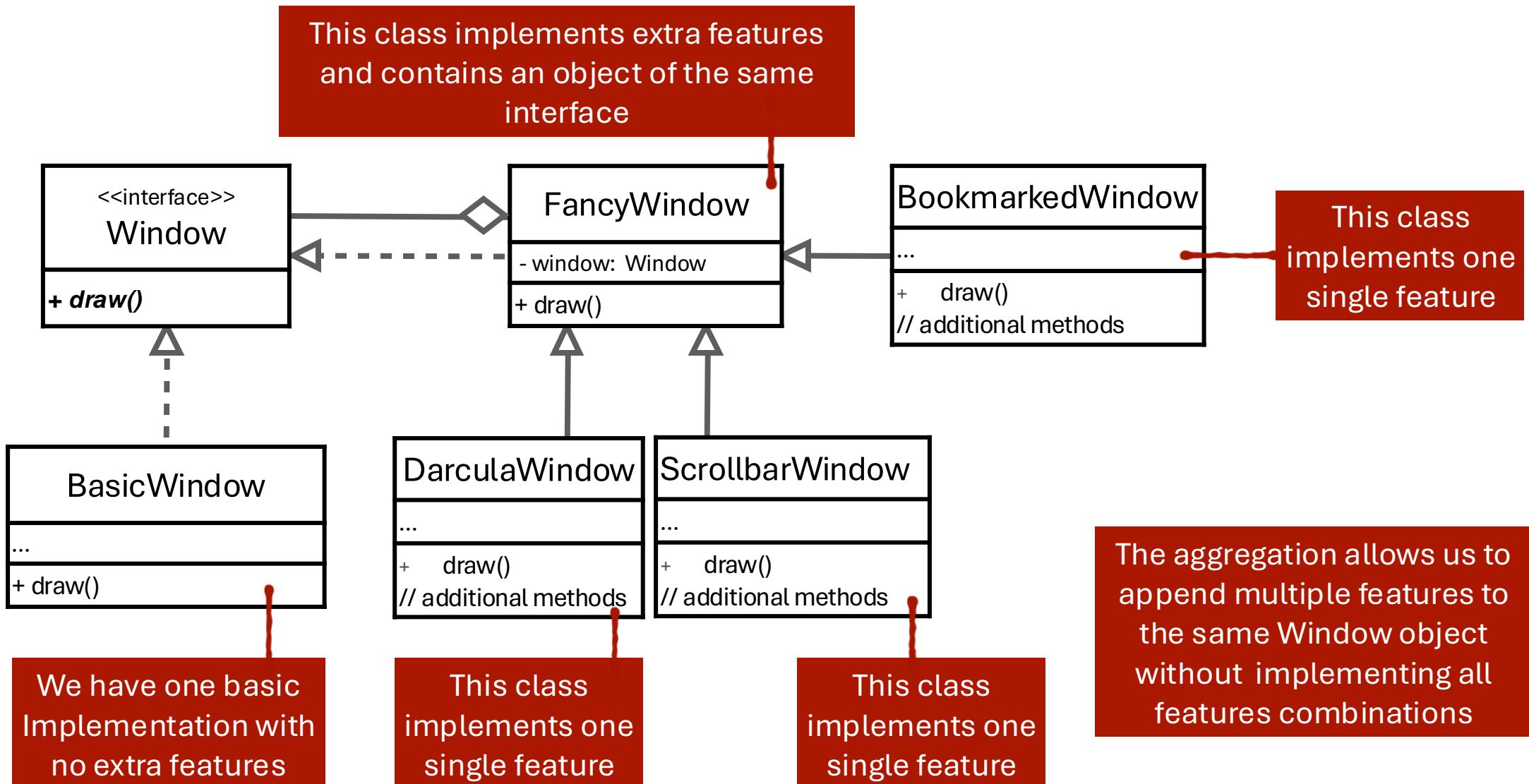
How do we change the code if we want to have a multiple features?

Very Bad solutions:

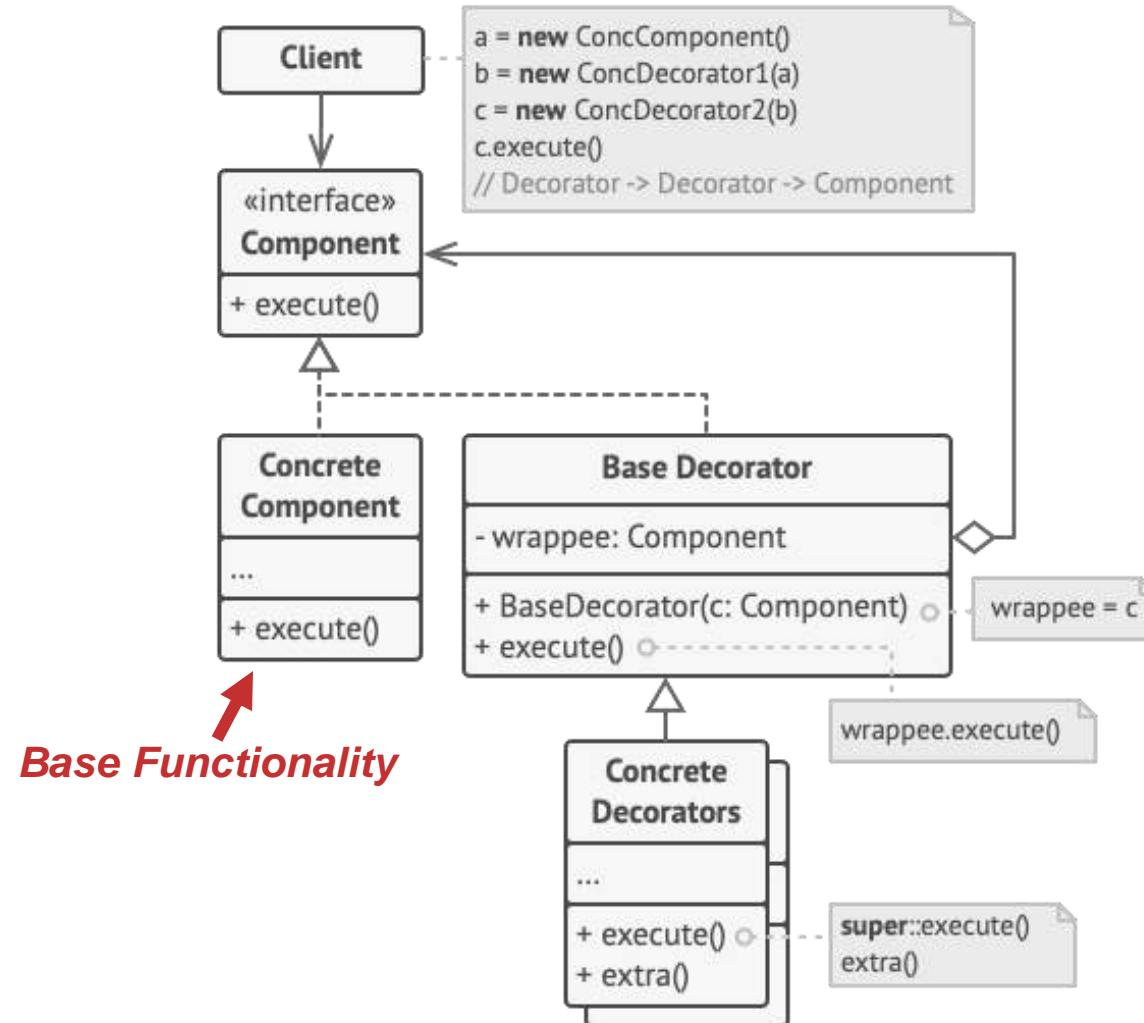
- Code clones
- More features means more combinations (classes) to implement
- If one feature is faulty, all related combinations are faulty

This implementation clearly violates the open-close principle

# Decorator Pattern – Good Solution

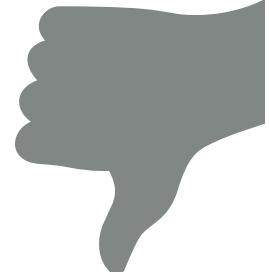


# Decorator Pattern



# Decorator Pattern

- Pros
  - Allows behaviour modification at runtime
  - Decorators provide a better solution to permutation issues (e.g., all combinations of armours and weapons)
  - It supports the open-close principle
- Cons
  - Decorators can result in too many small objects if overused.



# **Structural Patterns**

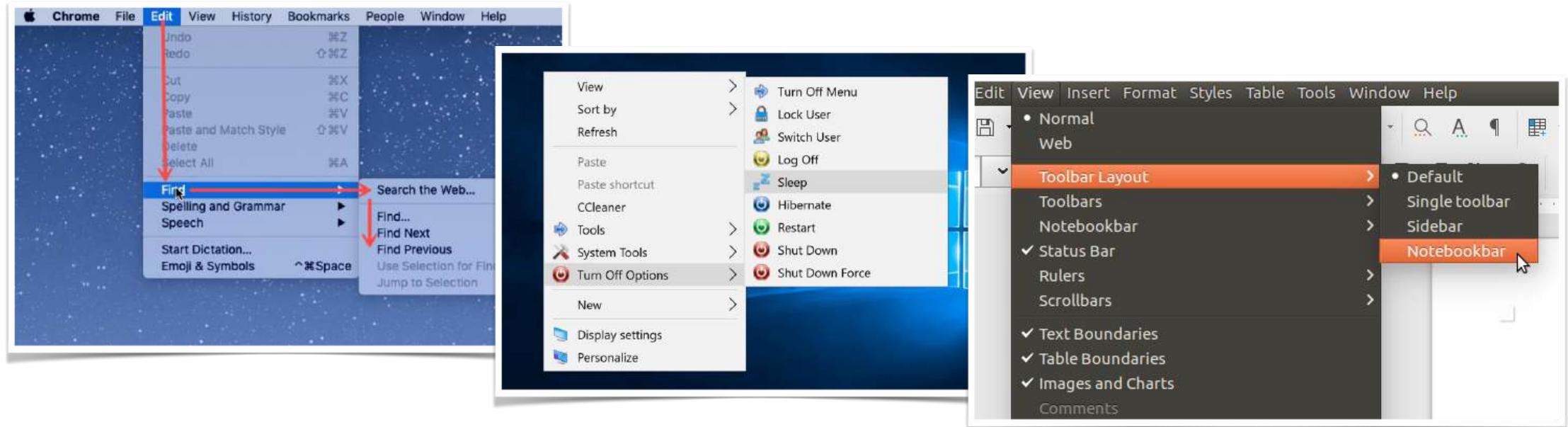
## **Composite Pattern**

# Composite Pattern

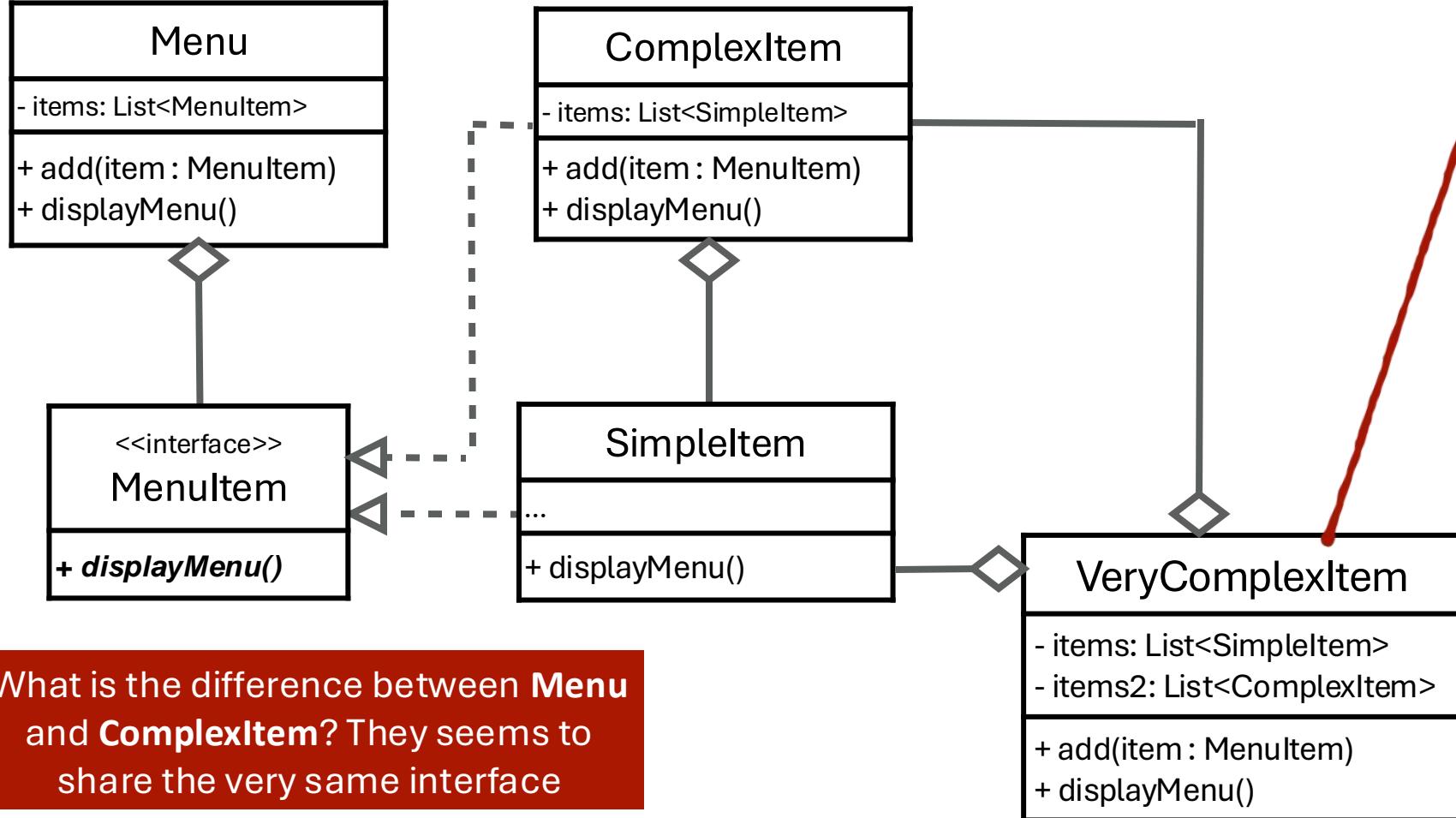
- Intent
  - Composes objects into tree structures
  - Work with these structures as if they were individual objects
- When
  - You want to represent part-whole hierarchies of objects.
  - You want clients to be able to ignore the difference between compositions of objects and individual objects. Clients will treat all objects in the composite structure uniformly.

# Composite Pattern – Example

We want to implement a GUI with a menu and sub-menus. Each menu item can be either a menu itself or a simple item with no sub-menu (recursive menu). Let's implement the menu in such a way we don't need to remember which menu item is “simple” or “a sub menu”.



# Composite Pattern – Bad Solution



What is the difference between **Menu** and **ComplexItem**? They seems to share the very same interface

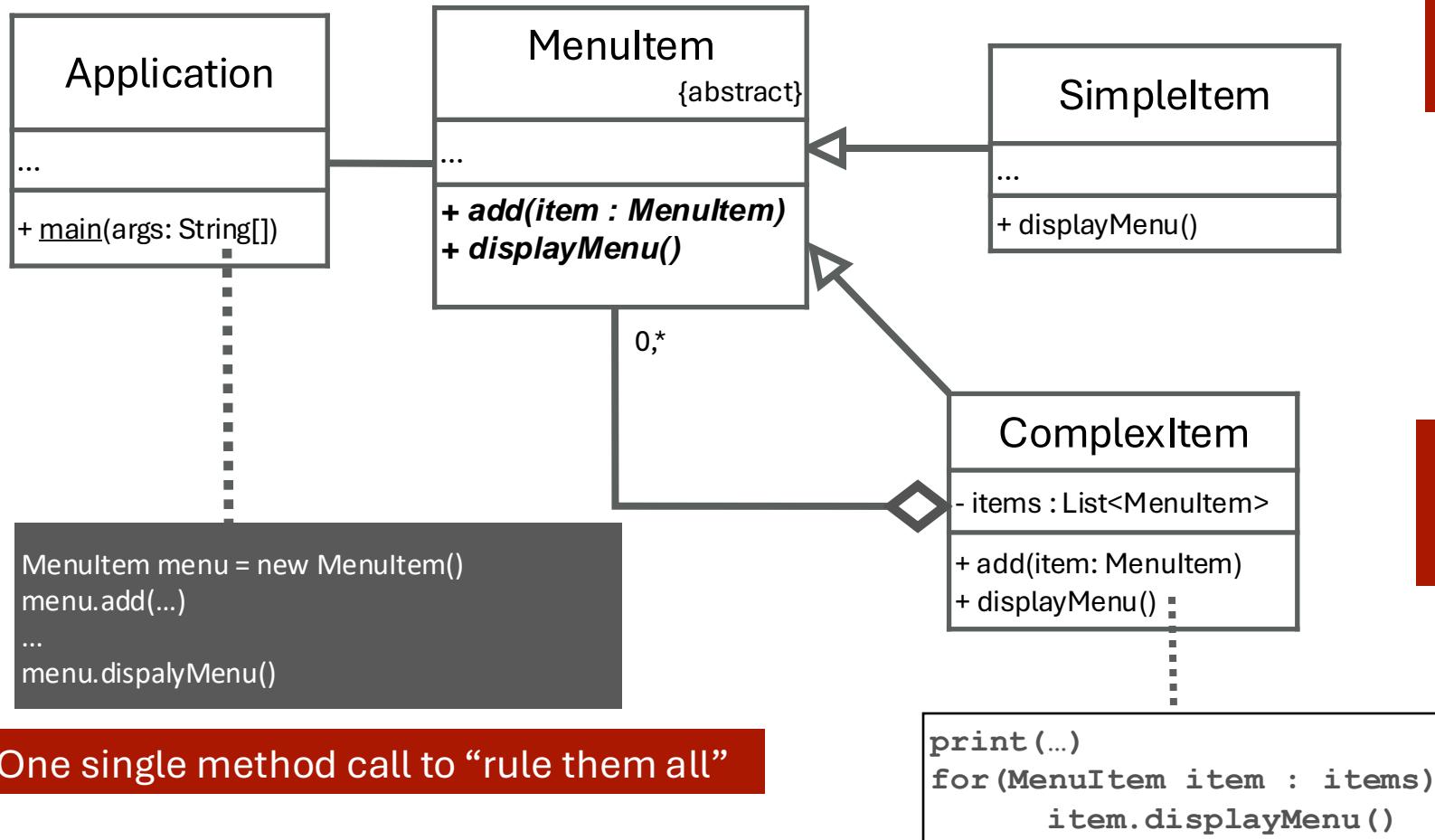
This implementation allows only one nested level of sub-menu

If we want to add multiple nested sub-menu, we need to define more classes

The main problem here is that all items (and menu) don't share the same interface

To display the full menu, we have to manually call the **displayMenu()** methods from all classes

# Composite Pattern – Good Solution



**SimpleItem** does not implement the other method (`add`)

(You can see this class as the termination of the recursive composition)

A **complexItem** contains a list of objects with the very same interface

(You can see this as recursive composition)

# Composite Pattern – Code

*All (sub)components in the hierarchy have the same interface (of this abstract class)*

```
public abstract class MenuItem {  
  
    private String name;  
    private String description;  
  
    public MenuItem(String pName, String text){  
        this.name = pName;  
        this.description = text;  
    }  
  
    public abstract void add(MenuItem component);  
  
    public abstract void displayMenu();  
  
    public String getName() {  
        return name;  
    }  
  
    public String getDescription() {  
        return description;  
    }  
}
```

*This method will be implemented and actually used only by complex menu items*

*This method will be implemented by all subclasses (either simple or complex item)*

# Composite Pattern – Code

```
public class SimpleItem extends MenuItem {
    public SimpleItem(String name, String url) {
        super(name, url);
    }

    @Override
    public void add(MenuItem component) {
        throw new UnsupportedOperationException();
    }

    @Override
    public void displayMenu() {
        System.out.println(this.getName() +
            " is the " +
            this.getDescription());
    }
}
```

*Printing code for simple items*

*“Adding” sub-item is not allowed  
for simple menu items*

```
import java.util.ArrayList;
import java.util.List;

public class ComplexItem extends MenuItem {
    List<MenuItem> subMenus = new ArrayList<>();

    public ComplexItem(String name, String url) {
        super(name, url);
    }

    @Override
    public void add(MenuItem component) {
        if (component != this)
            this.subMenus.add(component);
        else
            throw new IllegalArgumentException("Circular call");
    }

    @Override
    public void displayMenu() {
        System.out.println(this.getName() +
            " is the " +
            this.getDescription());

        this.subMenus.forEach(MenuItem::displayMenu);
    }
}
```

*Invoking the printing for all sub-items  
(they all have the same interface)*

 **Complex item includes multiple sub-items (aggregation)**

 **An item cannot include itself (no cycle allowed)**

 **Its own printing code**

# Composite Pattern – Code

```
public class Application {  
  
    public static void main(String args[]){  
        MenuItem menu = new ComplexItem("Edit Menu", "overall menu");  
  
        // let's create a complex menu item  
        MenuItem editMenu = new ComplexItem("Find", "first sub-menu");  
        menu.add(editMenu);  
  
        editMenu.add(new SimpleItem("Find", "..."));  
        editMenu.add(new SimpleItem("Replace", "..."));  
  
        // let's create another complex menu item  
        MenuItem shortCuts = new ComplexItem("Shortcuts", "third sub-menu");  
        menu.add(shortCuts);  
  
        MenuItem cut = new SimpleItem("Cut", "...");  
        shortCuts.add(cut);  
        MenuItem copy = new SimpleItem("Copy", "...");  
        shortCuts.add(copy);  
        MenuItem paste = new ComplexItem("Paste", "sub sub-menu");  
        shortCuts.add(paste);  
  
        // let's add a sub sub-menu  
        ...  
  
        menu.displayMenu();  
    }  
}
```



*Root of the menu (of the tree structure)*



*Sub-menu (internal node of the tree)*



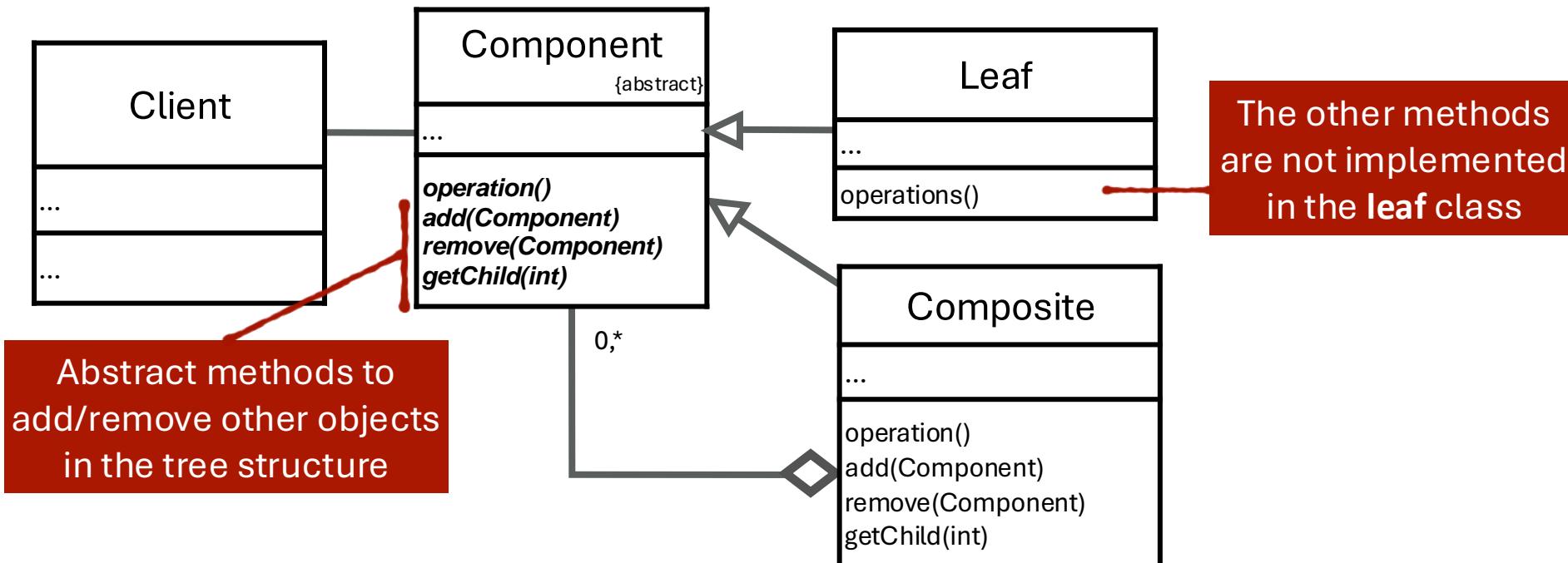
*Final items within the sub-menu  
(leaf node of the tree)*



*With one single call we can print the  
entire tree structure*

# Composite Pattern

The **Composite Pattern** composes objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly. The composite implements a “recursive” composition



# Composite Pattern

- Pros
  - It defines class hierarchies consisting of primitive objects and composite objects. Primitive objects can be composed into more complex objects, which in turn can be composed, and so on recursively.
  - It makes the client simple.
  - It makes it easier to add new kinds of components.
  - OCP: You can add new element types without breaking existing code, which now works with object trees.
- Cons
  - Difficult to provide a common interface for classes whose functionality differs too much, can cause complexity



# **Creational Patterns**

## **Factory Method Pattern**

# Factory Method Pattern

- Intent
  - Provide an interface for creating objects in a superclass
  - Allows subclasses to alter the type of objects that will be created
- When
  - A class can't anticipate the class of objects it must create.
  - A class wants its subclasses to specify the objects it creates.
  - Classes delegate responsibility to one of several helper subclasses, and you want to localize the knowledge of which helper subclass is the delegate.

# Problem

We want to develop a system that needs to **converts** (serialize) a Song into its a string representation using a specific string format (XML and JSON). We also want to de-serialize JSON and XML strings back to instances of the Song class.

We want to allow **multiple formats** (XML and JSON) but we also want to design the code such that we **can add more formats in the future**.

```
{  
  "id": "ID01",  
  "title": "Sweet Child O\u0027 Mine",  
  "artist": "Guns N\u0027 Roses",  
  "record-label": "Geffen Records",  
  "year": 1990  
}
```

```
<Song>  
  <id>ID01</id>  
  <title>Sweet Child O' Mine</title>  
  <artist>Guns N' Roses</artist>  
  <recordLabel>Geffen Records</recordLabel>  
  <year>1990</year>  
</Song>
```

# A Bad Solution

```
public class Song {  
    private String id;  
    private String title;  
    private String artist;  
    private String recordLabel;  
    private int year;  
  
    public Song(String pId, String pTitle,  
               String pSinger, String pRecord, int pYear){  
        this.id = pId;  
        this.title = pTitle;  
        this.artist = pSinger;  
        this.recordLabel = pRecord;  
        this.year = pYear;  
    }  
  
    // getters  
    ...  
}
```

```
public enum Format {  
    XML, JSON  
}
```

```
public class SongSerializer {  
  
    public String serialize (Song song, Format format) {  
        String result = "";  
        if (format == Format.JSON) {  
            // ...  
        } else if (format == Format.XML) {  
            // ...  
        }  
        return result;  
    }  
  
    public Song deserialize(String string, Format format) {  
        if (format == Format.JSON) {  
            // ...  
        } else if (format == Format.XML) {  
            // ...  
        }  
        return null;  
    }  
}
```

The entire logic is implemented in the SongSerializer class

# A Bad Solution

```
public class SongSerializer {  
  
    public String serialize (Song song, Format format) {  
        String result = "";  
        if (format == Format.JSON){  
            Gson gson = new Gson();  
  
            JsonObject json = new JsonObject();  
            json.add("id", new JsonPrimitive(song.getId()));  
            json.add("title", new JsonPrimitive(song.getTitle()));  
            json.add("artist", new JsonPrimitive(song.getArtist()));  
            json.add("record-label", new JsonPrimitive(song.getRecordLabel()));  
            json.add("year", new JsonPrimitive(song.getYear()));  
  
            result = gson.toJson(json);  
        } else if (format == Format.XML) {  
            XmlMapper xmlMapper = new XmlMapper();  
            try {  
                result = xmlMapper.writeValueAsString(song);  
            } catch (JsonProcessingException e) {  
                e.printStackTrace();  
            }  
        }  
        return result;  
    }  
  
    public Song deserialize(String string, Format format){  
        if (format == Format.JSON) {  
            // ...  
        } else if (format == Format.XML) {  
            // ...  
        }  
        return null;  
    }  
}
```

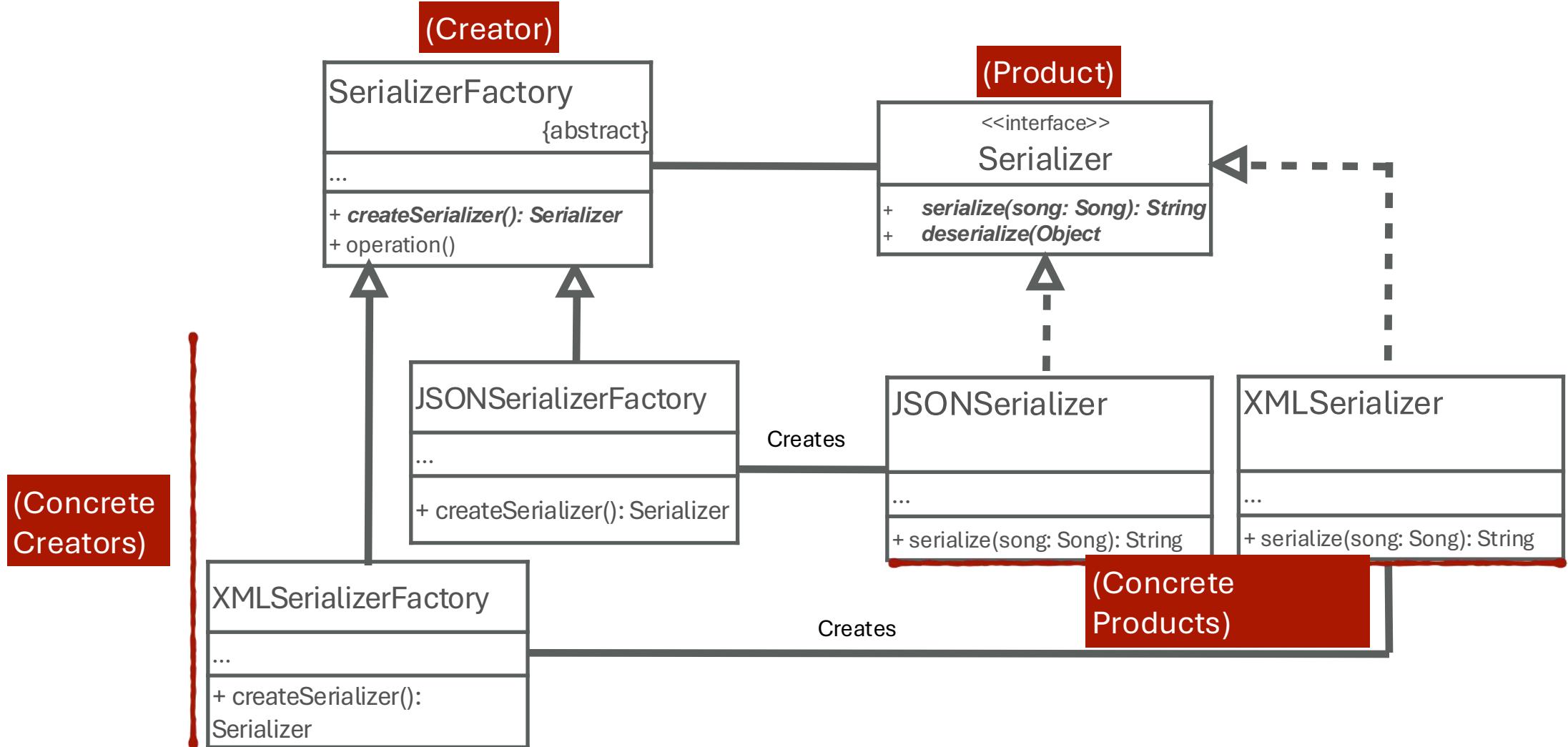
Violation of the single-responsibility principle (this class implements multiple functionalities)

Violation of the open-close principle: to add more formats, we have to change this code

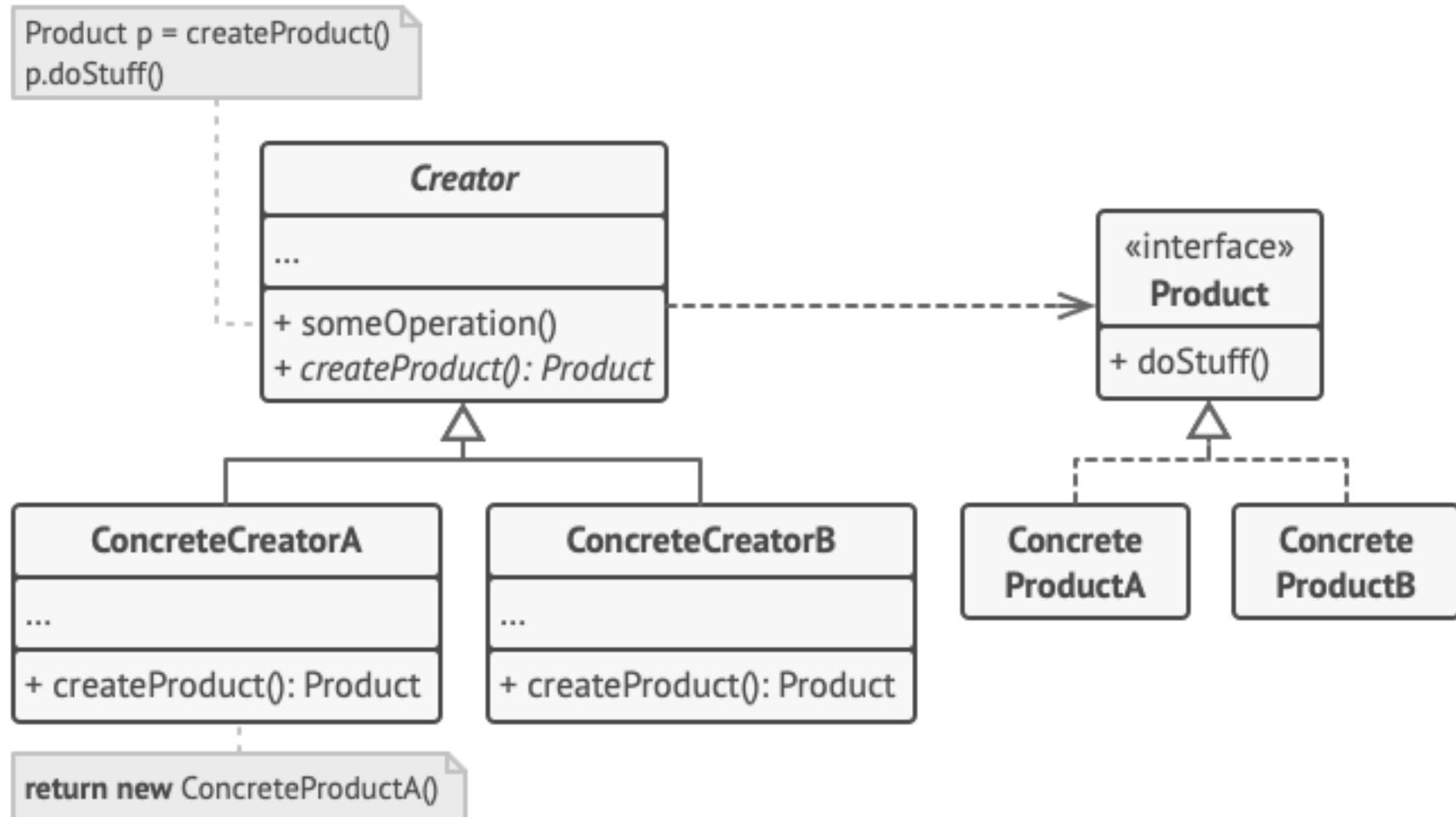
If we add more formats, this class will bloat up continuously increasing in size and complexity

Is this maintainable in a longer term?

# Factory Method – Good Solution



# Factory Method Pattern



# Factory Method Pattern

- Pros
  - Avoid tight coupling between creator and concrete products
  - SRP: You can move product creation code into one place in the program
  - OCP: You can add new types of products into the program without breaking existing client code
- Cons
  - More complex code due to subclasses necessary to implement the pattern



# **Creational Patterns**

## **Abstract Factory Pattern**

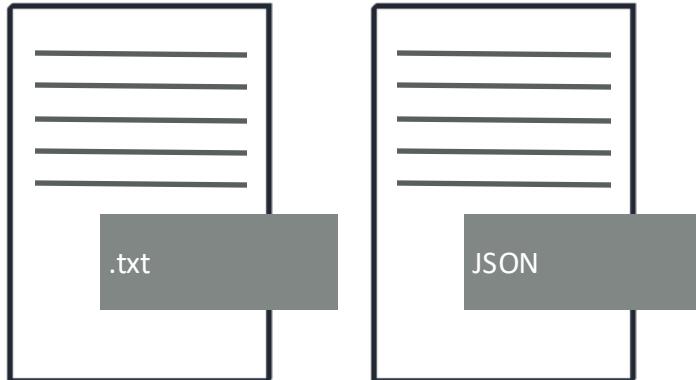
# Abstract Factory Pattern

- Intent
  - Allows you to produce families of related objects without specifying their concrete classes.
- When
  - A system should be independent of how its products are created, composed, and represented.
  - A system should be configured with one of multiple families of products.
  - A family of related product objects is designed to be used together, and you need to enforce this constraint.
  - You want to provide a class library of products, and you want to reveal just their interfaces, not their implementations.

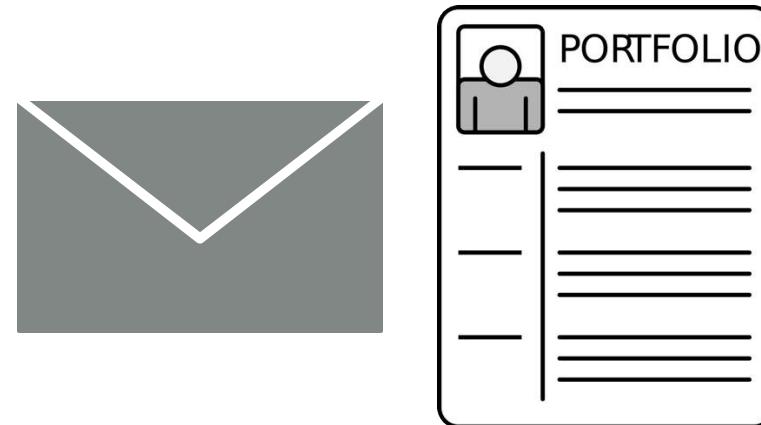
# Problem

We are developing an application for **creating** different **types of documents** and in different formats. We want to be able to generate different **types of documents** (e.g., letter, resume, essay, etc.) and, at the same time, allowing to use **different formats** for these documents (text file, JSON, etc.). We want to make the system extensible for more formats and type of documents.

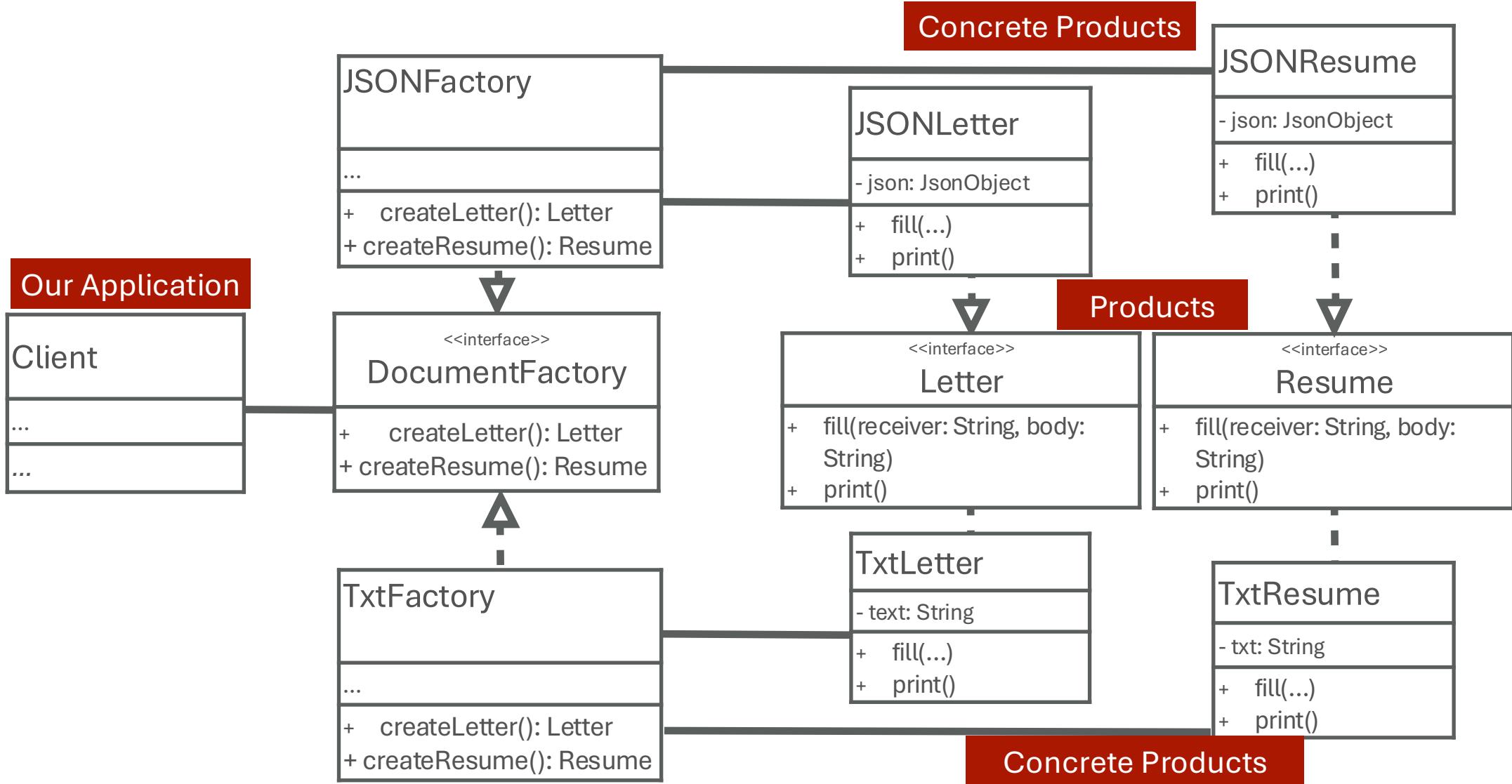
Formats



Document Types



# Abstract Factory – Good Solution



# Abstract Factory – Code

## *Abstract Product 1*

```
public interface Letter {  
    void fill(String receiver, String body);  
    void print();  
}
```

## *Concrete Product 1*

```
public class JSONLetter implements Letter {  
    private JsonObject json;  
  
    @Override  
    public void fill(String receiver, String body) {  
        this.json = new JsonObject();  
        json.add("receiver", new JsonPrimitive(receiver));  
        json.add("body", new JsonPrimitive(body));  
        json.add("ending", new JsonPrimitive("Best, Sansa Stark"));  
    }  
  
    @Override  
    public void print() {  
        Gson gson = new Gson();  
        System.out.println(gson.toJson(json));  
    }  
}
```

## *Concrete Product 1*

```
public class TxtLetter implements Letter {  
    private String letter;  
  
    @Override  
    public void fill(String receiver, String body) {  
        letter = "Dear " + receiver + ",\n" +  
            body + "\n" +  
            "Regards,\n" +  
            "John Smith";  
    }  
  
    @Override  
    public void print() {  
        System.out.println(letter);  
    }  
}
```

# Abstract Factory – Code

## *Abstract Product 2*

```
public interface Resume {  
    void fill(String info, String education, String skills);  
    void print();  
}
```

## *Concrete Product 2*

```
public class JSONResume implements Resume {  
    private JSONObject json;  
  
    @Override  
    public void fill(String info, String education, String skills) {  
        this.json = new JSONObject();  
        json.add("info", new JsonPrimitive(info));  
        json.add("education", new JsonPrimitive(education));  
        json.add("skills", new JsonPrimitive(education));  
    }  
  
    @Override  
    public void print() {  
        Gson gson = new Gson();  
        System.out.println(gson.toJson(json));  
    }  
}
```

## *Concrete Product 2*

```
public class TxtResume implements Resume {  
    private String letter;  
  
    @Override  
    public void fill(String info, String education, String skills) {  
        letter = "Name and Surname " + info + ",\n" +  
            "My education = " + education + "\n" +  
            "My Skills = skills \n";  
    }  
  
    @Override  
    public void print() {  
        System.out.println(letter);  
    }  
}
```

# Abstract Factory – Code

## *Abstract Factory*

```
public interface DocumentFactory {  
    Letter createLetter();  
    Resume createResume();  
}
```

## *Concrete Product 2*

```
public class JSONFactory implements DocumentFactory {  
  
    @Override  
    public Letter createLetter() {  
        return new JSONLetter();  
    }  
  
    @Override  
    public Resume createResume() {  
        return new JSONResume();  
    }  
}
```

## *Concrete Product 2*

```
public class TextFactory implements DocumentFactory {  
  
    @Override  
    public Letter createLetter() {  
        return new TxtLetter();  
    }  
  
    @Override  
    public Resume createResume() {  
        return new TxtResume();  
    }  
}
```

# Abstract Factory – Code

## Main Application

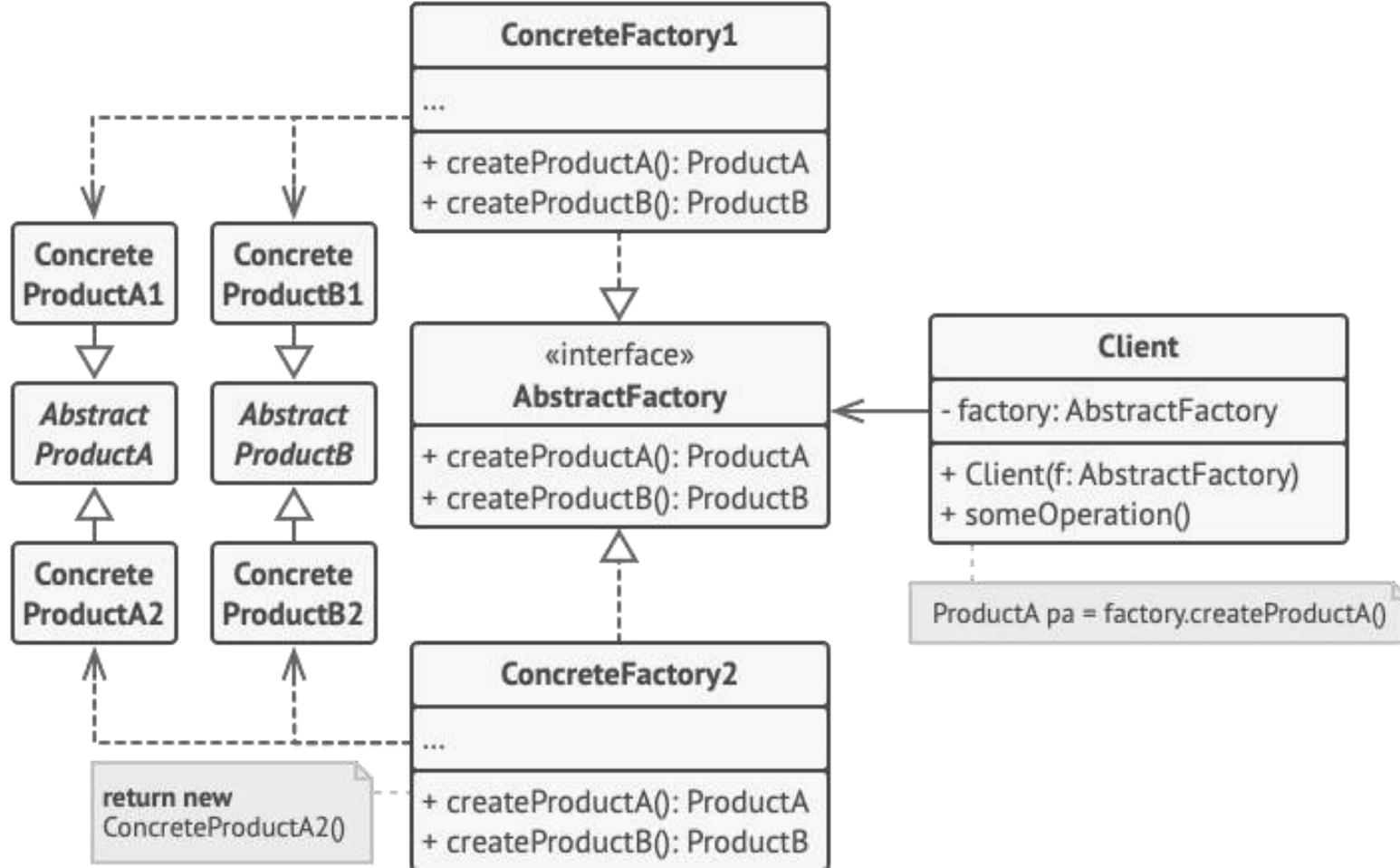
```
public class Application {  
    enum Type {JSON, TXT}  
  
    public static void main(String[] args) {  
        DocumentFactory factory;  
        Letter letter;  
  
        Type type = Type.TXT;  
        if (type == Type.JSON) {  
            factory = new JSONFactory();  
        } else {  
            factory = new TextFactory();  
        }  
  
        letter = factory.createLetter();  
        letter.fill("John Snow", "You know nothing");  
        letter.print();  
    }  
}
```

All factories share a common interface  
(Document Factory)

Let's choose the concrete factory we  
need

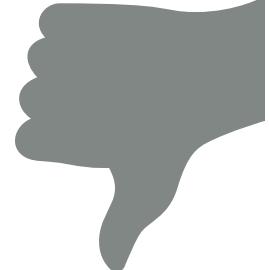
The factory will create the product  
(JSON letter) we need

# Abstract Factory Pattern



# Abstract Factory Pattern

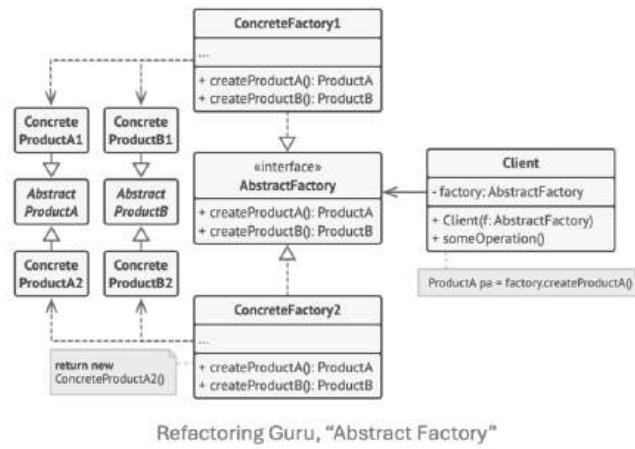
- Pros
  - Factory produces compatible products
  - Avoids tight coupling between concrete products and client code
  - SRP: You can extract product creation code to one place
  - OCP: You can introduce new variants of products without breaking existing code
- Cons
  - Code complexity, since a lot of interfaces and classes are needed.



# Abstract factory vs. Factory Method



## Abstract Factory Pattern



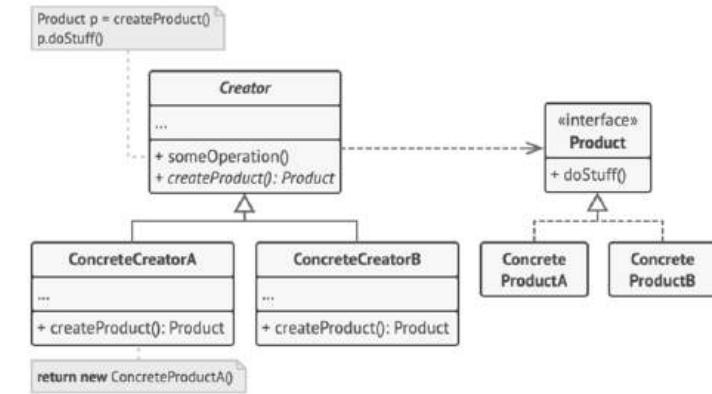
Refactoring Guru, "Abstract Factory"

161

We abstract factories



## Factory Method Pattern



Refactoring Guru, "Factory Method"

151

We abstract one single method within a single factory

# **Creational Patterns**

## **Builder Pattern**

# Builder Pattern

- Intent
  - Construct complex objects step by step
  - Produce different types and representations of objects using the same construction code
- When
  - The algorithm for creating a complex object should be independent of the parts that make up the object and how they're assembled.
  - The construction process must allow different representations for the object that is constructed.

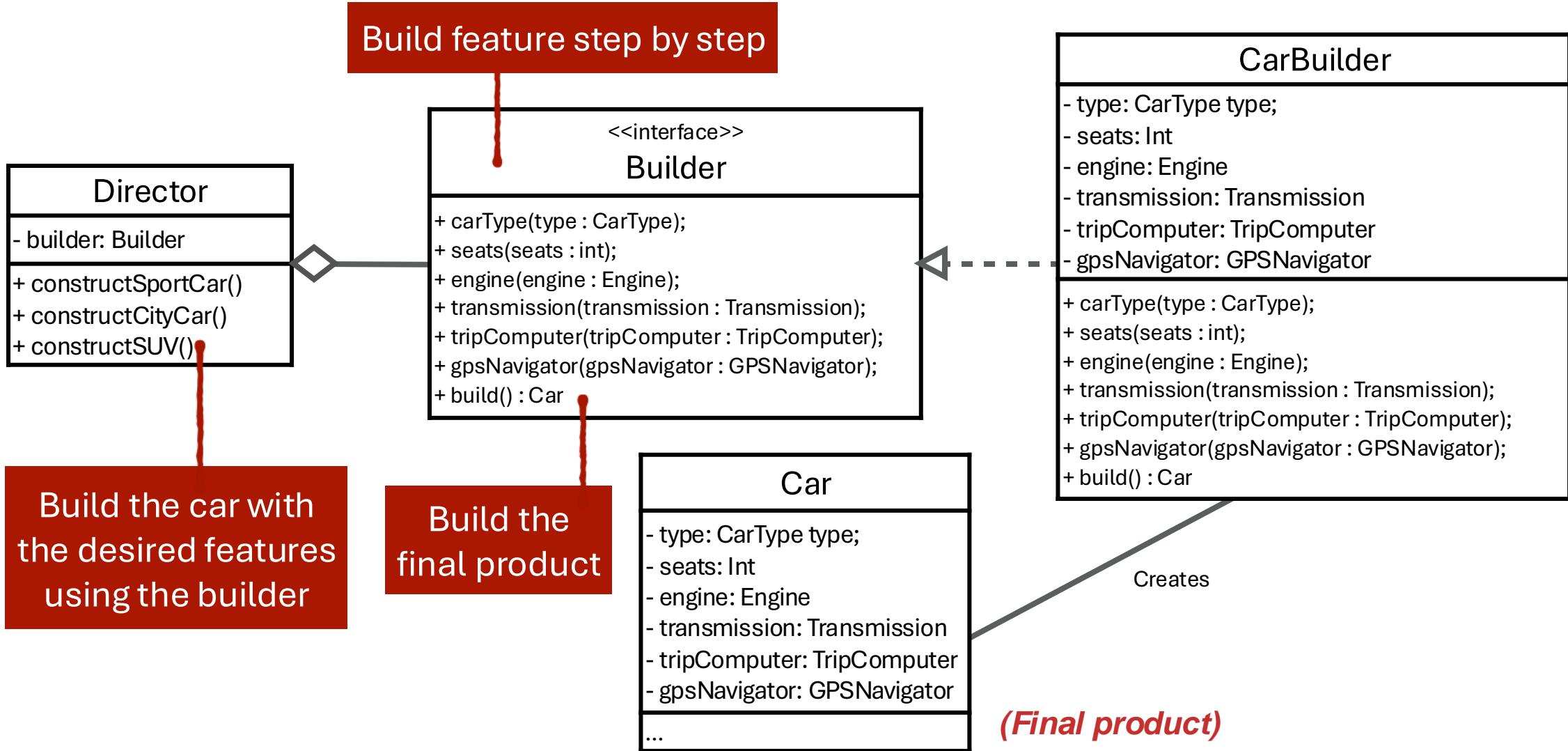
# Builder Pattern – Problem

We want to design a system that allows us to create different car models. We have different car models, such as sport car, city car, and SUV. Each car model has its own basic feature, such as the number of seats, type of engine, and type of transmission.

However, more features can be added or upgrading existing features based on the contract agreement (and the price) with the customer.



# Builder Pattern – Good Solution



# Builder Pattern – Code

*The main product*

```
public class Car {  
    private final CarType carType;  
    private final int seats;  
    private final Engine engine;  
    private final Transmission transmission;  
    private final TripComputer tripComputer;  
    private final GPSNavigator gpsNavigator;  
    private double fuel = 0;  
  
    public Car(CarType carType, int seats, Engine engine, Transmission transmission,  
              TripComputer tripComputer, GPSNavigator gpsNavigator) {  
        this.carType = carType;  
        this.seats = seats;  
        this.engine = engine;  
        this.transmission = transmission;  
        this.tripComputer = tripComputer;  
        if (this.tripComputer != null) {  
            this.tripComputer.setCar(this);  
        }  
        this.gpsNavigator = gpsNavigator;  
    }  
  
    // getters and setters  
}
```



*All possible features of a car*

*Getters and setters omitted for simplicity*

# Builder Pattern – Code

```
public class TripComputer {
    private Car car;

    public void setCar(Car car) {
        this.car = car;
    }

    public void showFuelLevel() {
        System.out.println("Fuel level: " + car.getFuel());
    }

    public void showStatus() {
        //...
    }
}
```

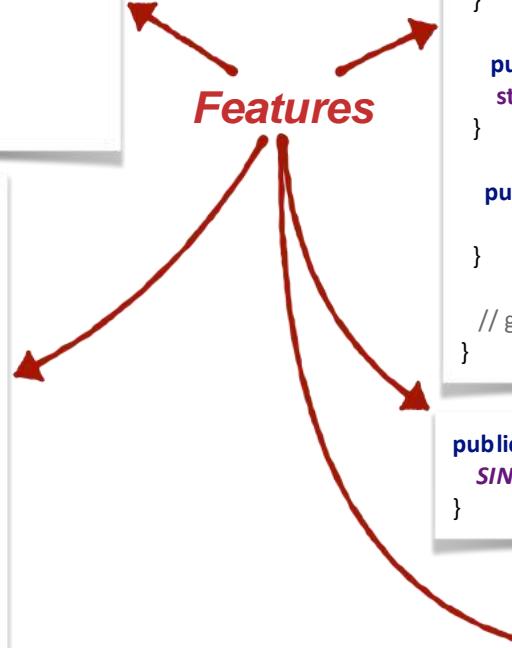
```
public class GPSNavigator {
    private String route;

    public GPSNavigator() {
        this.route = "...";
    }

    public GPSNavigator(String manualRoute) {
        this.route = manualRoute;
    }

    public String getRoute() {
        return route;
    }
}
```

**Features**



```
public class Engine {
    private final double volume;
    private double mileage;
    private boolean started;

    public Engine(double volume, double mileage) {
        this.volume = volume;
        this.mileage = mileage;
    }

    public void start() {
        started = true;
    }

    public void stop() {
        started = false;
    }

    public boolean isStarted() {
        return started;
    }

    // getter methods
}

public enum Transmission {
    SINGLE_SPEED, MANUAL, AUTOMATIC, SEMI_AUTOMATIC
}
```

```
public enum CarType {
    CITY_CAR, SPORTS_CAR, SUV
}
```

# Builder Pattern – Code

Build feature step by step

```
public interface Builder {  
    void setCarType(CarType type);  
    void setSeats(int seats);  
    void setEngine(Engine engine);  
    void setTransmission(Transmission transmission);  
    void setTripComputer(TripComputer tripComputer);  
    void setGPSNavigator(GPSNavigator gpsNavigator);  
    Car build();  
}
```

Build the final product

```
public class CarBuilder implements Builder {  
    private CarType type;  
    private int seats;  
    private Engine engine;  
    private Transmission transmission;  
    private TripComputer tripComputer;  
    private GPSNavigator gpsNavigator;  
  
    public void setCarType(CarType type) {  
        this.type = type;  
    }  
  
    public void setSeats(int seats) {  
        this.seats = seats;  
    }  
  
    public void setEngine(Engine engine) {  
        this.engine = engine;  
    }  
  
    public void setTransmission(Transmission transmission) {  
        this.transmission = transmission;  
    }  
  
    public void setTripComputer(TripComputer tripComputer) {  
        this.tripComputer = tripComputer;  
    }  
  
    public void setGPSNavigator(GPSNavigator gpsNavigator) {  
        this.gpsNavigator = gpsNavigator;  
    }  
  
    public Car build() {  
        return new Car(type, seats, engine, transmission, tripComputer, gpsNavigator);  
    }  
}
```

**Concrete Builder**

# Builder Pattern – Code

```
public class Director {  
  
    public void constructSportsCar(Builder builder) {  
        builder.setCarType(CarType.SPORTS_CAR);  
        builder.setSeats(2);  
        builder.setEngine(new Engine(3.0, 0));  
        builder.setTransmission(Transmission.SEMI_AUTOMATIC);  
        builder.setTripComputer(new TripComputer());  
        builder.setGPSNavigator(new GPSNavigator());  
    }  
  
    public void constructCityCar(Builder builder) {  
        builder.setCarType(CarType.CITY_CAR);  
        builder.setSeats(2);  
        builder.setEngine(new Engine(1.2, 0));  
        builder.setTransmission(Transmission.AUTOMATIC);  
        builder.setTripComputer(new TripComputer());  
        builder.setGPSNavigator(new GPSNavigator());  
    }  
  
    public void constructSUV(Builder builder) {  
        builder.setCarType(CarType.SUV);  
        builder.setSeats(4);  
        builder.setEngine(new Engine(2.5, 0));  
        builder.setTransmission(Transmission.MANUAL);  
        builder.setGPSNavigator(new GPSNavigator());  
    }  
}
```

## Main Application

```
public class Application {  
    public static void main(String[] args) {  
        // make the builder  
        Builder builder = new CarBuilder();  
  
        // build the car features using the director  
        Director director = new Director(builder);  
        director.constructCityCar();  
  
        // let's retrieve the results  
        Car car = builder.build();  
        System.out.print(car);  
    }  
}
```

*Each car model (e.g., SUV) has its own basic set of features*

```
public class Director {
    public void constructSportsCar(Builder builder) {
        builder.carType(CarType.SPORTS_CAR)
            .seats(2)
            .engine(new Engine(3.0, 0))
            .transmission(Transmission.SEMI_AUTOMATIC)
            .tripComputer(new TripComputer())
            .gpsNavigator(new GPSNavigator());
    }

    public void constructCityCar(Builder builder) {
        builder.carType(CarType.CITY_CAR)
            .seats(2)
            .engine(new Engine(1.2, 0))
            .transmission(Transmission.AUTOMATIC)
            .tripComputer(new TripComputer())
            .gpsNavigator(new GPSNavigator());
    }

    public void constructSUV(Builder builder) {
        builder.carType(CarType.SUV)
            .seats(4)
            .engine(new Engine(2.5, 0))
            .transmission(Transmission.MANUAL)
            .gpsNavigator(new GPSNavigator());
    }
}
```

Return an object of the same type

### Main Application

```
public class Application {
    public static void main(String[] args) {
        // make the builder
        Builder builder = new CarBuilder();

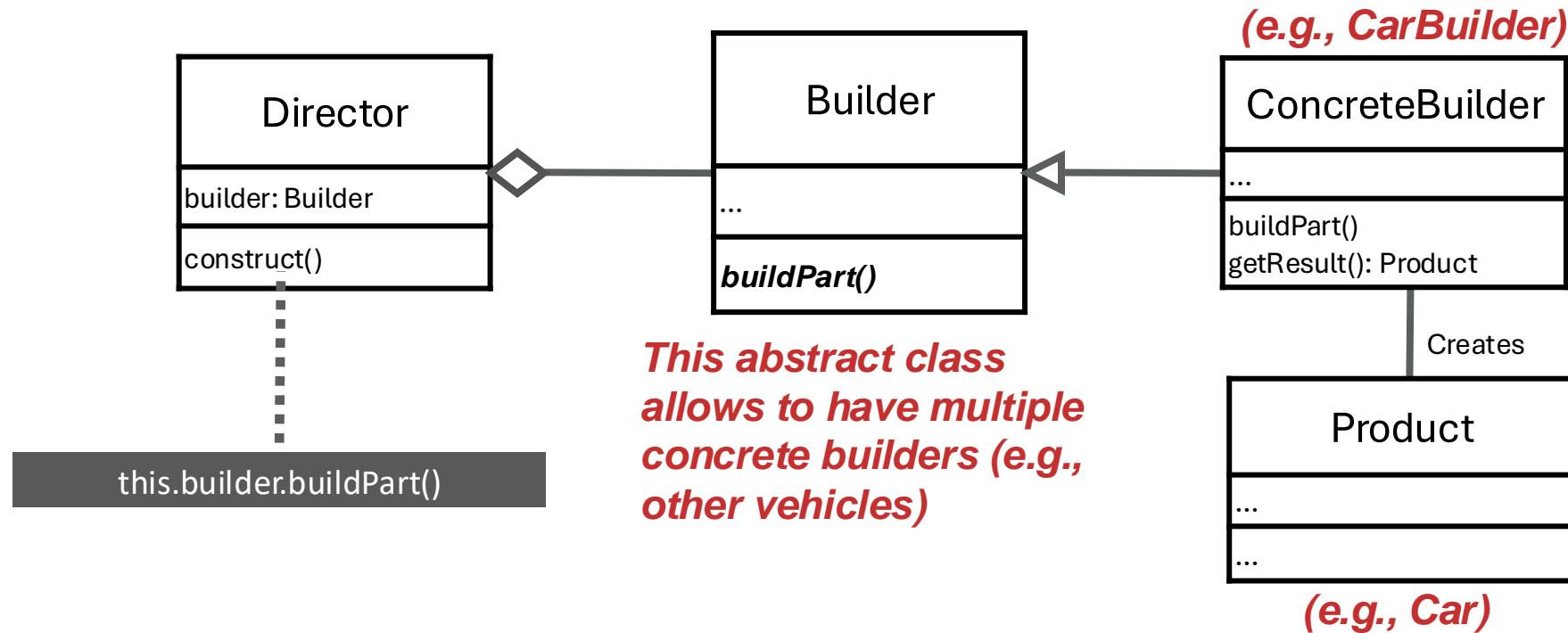
        // build the car features using the director
        Director director = new Director(builder);
        director.constructCityCar();

        // let's retrieve the results
        Car car = builder.build();
        System.out.print(car);
    }
}
```

```
public interface Builder {
    Builder carType(CarType type);
    Builder seats(int seats);
    Builder engine(Engine engine);
    Builder transmission(Transmission transmission);
    Builder tripComputer(TripComputer tripComputer);
    Builder gpsNavigator(GPSNavigator gpsNavigator);
    Car build();
}
```

# Builder Pattern

The **Builder Pattern** separates the construction of a complex object from its representation so that the same construction process can create different representations. A product is built step-by-step.



# Abstract factory vs. Builder

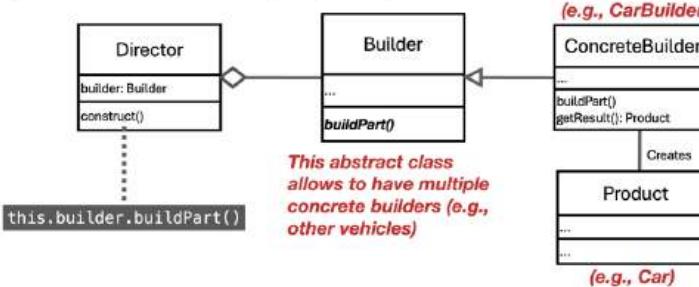
**Builder pattern** focuses on constructing a complex object step by step.

**Abstract Factory** emphasizes the families of product objects (either simple or complex) to create.



## Design Pattern: Builder

The **Builder Pattern** separates the construction of a complex object from its representation so that the same construction process can create different representations. A product is built step-by-step.

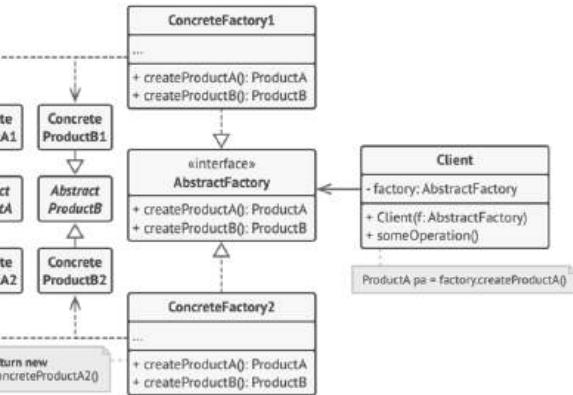


173



*The same product/object with different parts*

## Abstract Factory Pattern



Refactoring Guru, "Abstract Factory"

161

*Family of products*

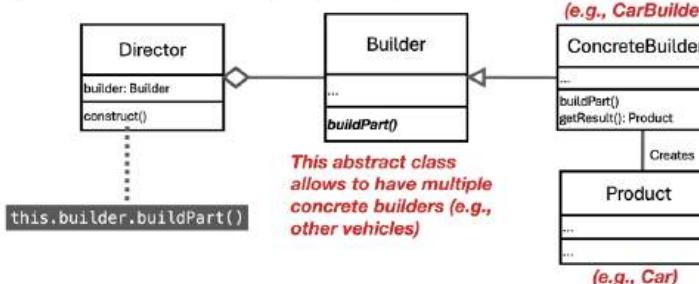
# Factory Method vs. Builder

**Builder pattern** focuses on constructing a complex object step by step. **Factory Method** instantiates different objects with the same APIs.



## Design Pattern: Builder

The **Builder Pattern** separates the construction of a complex object from its representation so that the same construction process can create different representations. A product is built step-by-step.

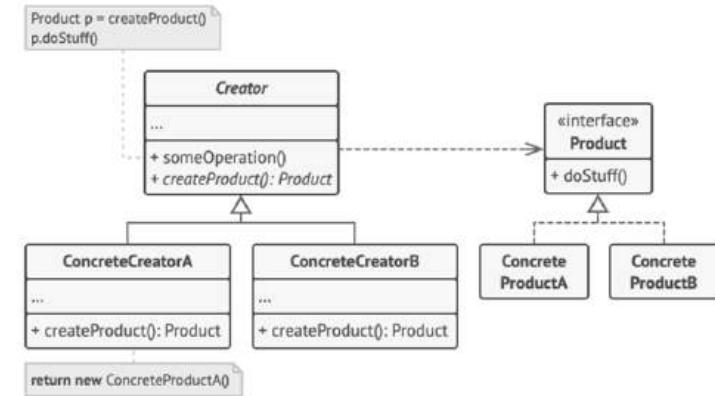


173

*The same product/object with different parts*



## Factory Method Pattern



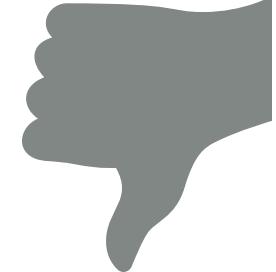
Refactoring Guru, "Factory Method"

151

*Factory for different products/objects*

# Builder Pattern

- Pros
  - Construct objects step-by-step, defer some steps or run steps recursively
  - Reuse the same construction code for various representations of products
  - SRP: isolate complex construction code from the business logic of the product
- Cons
  - Code complexity increases as builders require multiple new classes

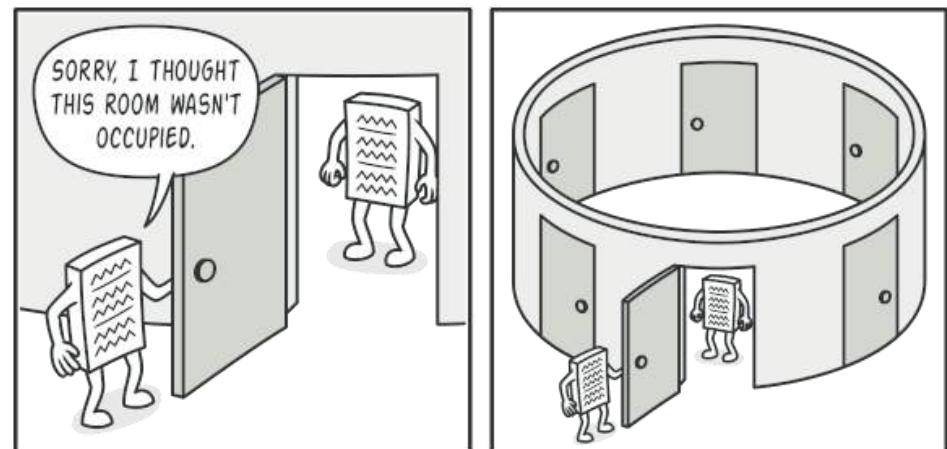


# **Creational Patterns**

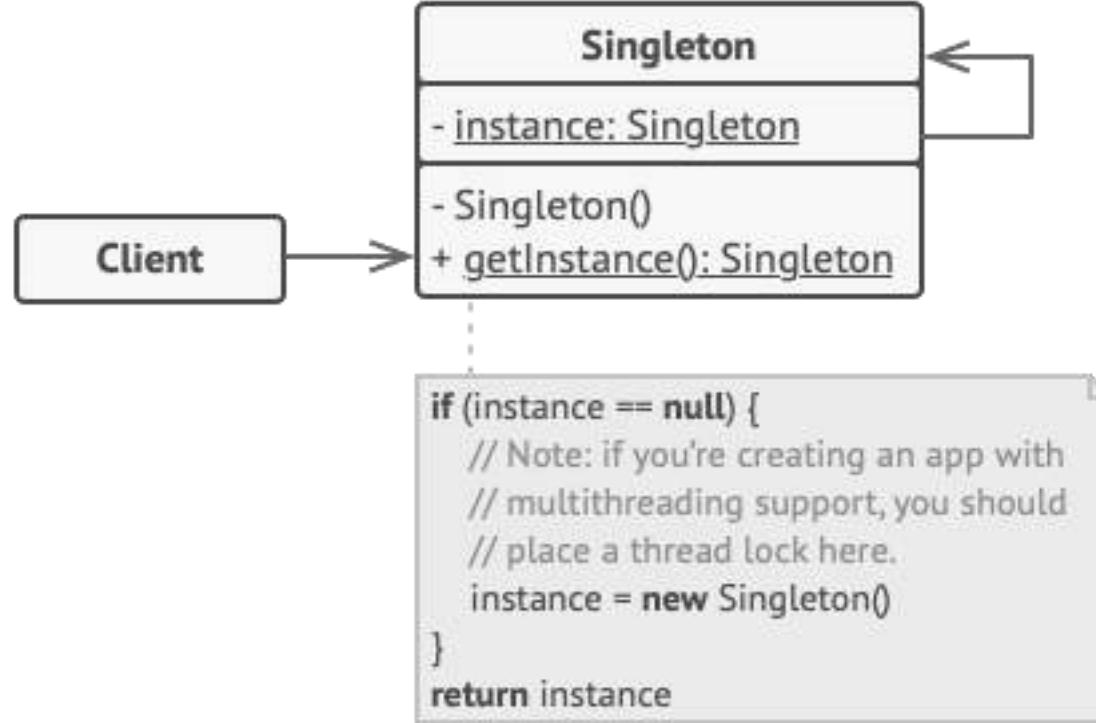
## **Singleton Pattern**

# Singleton Pattern

- Intent
  - Ensure that a class has only one instance
  - Provide global access to that class
- When
  - You can't use static (e.g. inheritance/interfaces)
  - There must be only one instance of a class



# Singleton Pattern



# Singleton Pattern

- Pros
  - Single instance
  - Globally accessible
  - Single instantiation
- Cons
  - Can mask bad design, for instance components that know too much about each other
  - Requires special treatment in a multithreaded environment to avoid multiton
  - Difficult to unit test clients of singletons
  - You cannot subclass from singletons



# Exercises

# Which Design Pattern to Apply?

## Scenario:

Design an online **investment** system for a large international bank allowing users to create investments. Users can create various **types of investments** (e.g., bonds) across **different markets** (e.g., US, EU), each with specific attributes like interest rate, risk level, and duration.

The system should accommodate multiple combinations of markets and investment types while minimizing code repetition. Additionally, the system should remain open for future expansion to include more markets and investment types



# Reverse Exercise

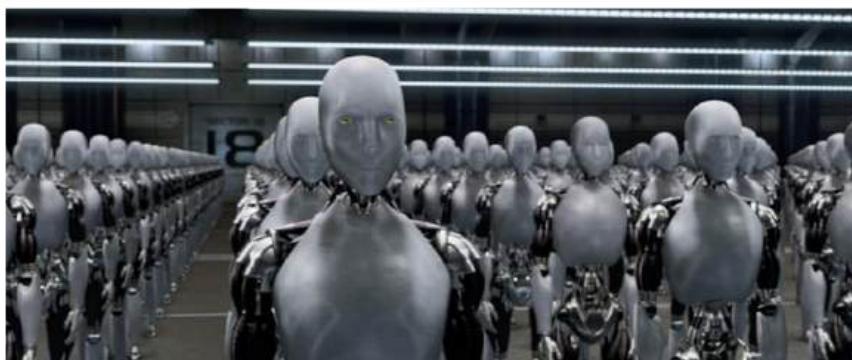
- Let's consider the **Strategy pattern**.
- Describe a real-world application or system where the **Strategy pattern** can be employed. Illustrate a scenario highlighting the benefits and advantages of using the Strategy pattern in that context.



# Which Design Pattern to Apply?

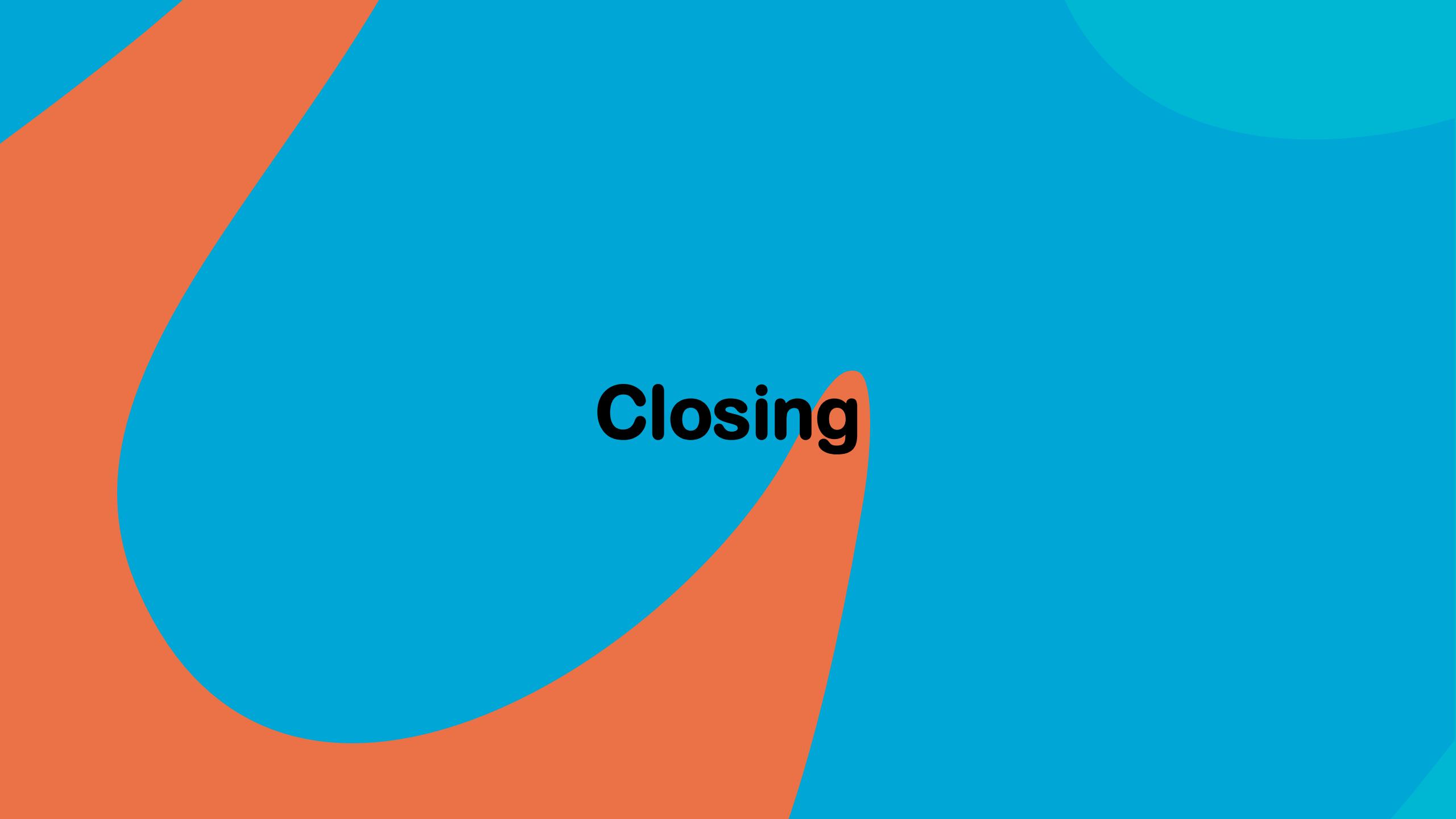
## Scenario:

Develop an online system for building **robots** with customizable hardware (**sensors**) and software (**AI engines**). Customers can incrementally select different types of sensors (e.g., cameras, tactile sensors, microphones) and their corresponding AI engines (e.g., deep-learning recognition, regression models, text-to-speech) to build customized robots.



“Io Robot” from Wikipedia

The system should allow step-by-step **assembly of robots** with **selectable sensor and AI engine combinations** while accommodating future expansions for additional sensor and AI options.

The background features a large, abstract shape composed of two main colors: orange and teal. The orange shape is located on the left side, starting from the top-left corner and curving down towards the bottom-right. The teal shape is on the right side, starting from the top-right corner and curving down towards the bottom-left. The two shapes overlap in the center.

# **Closing**

# Further Reading



- Refactoring Guru
- <https://refactoring.guru>

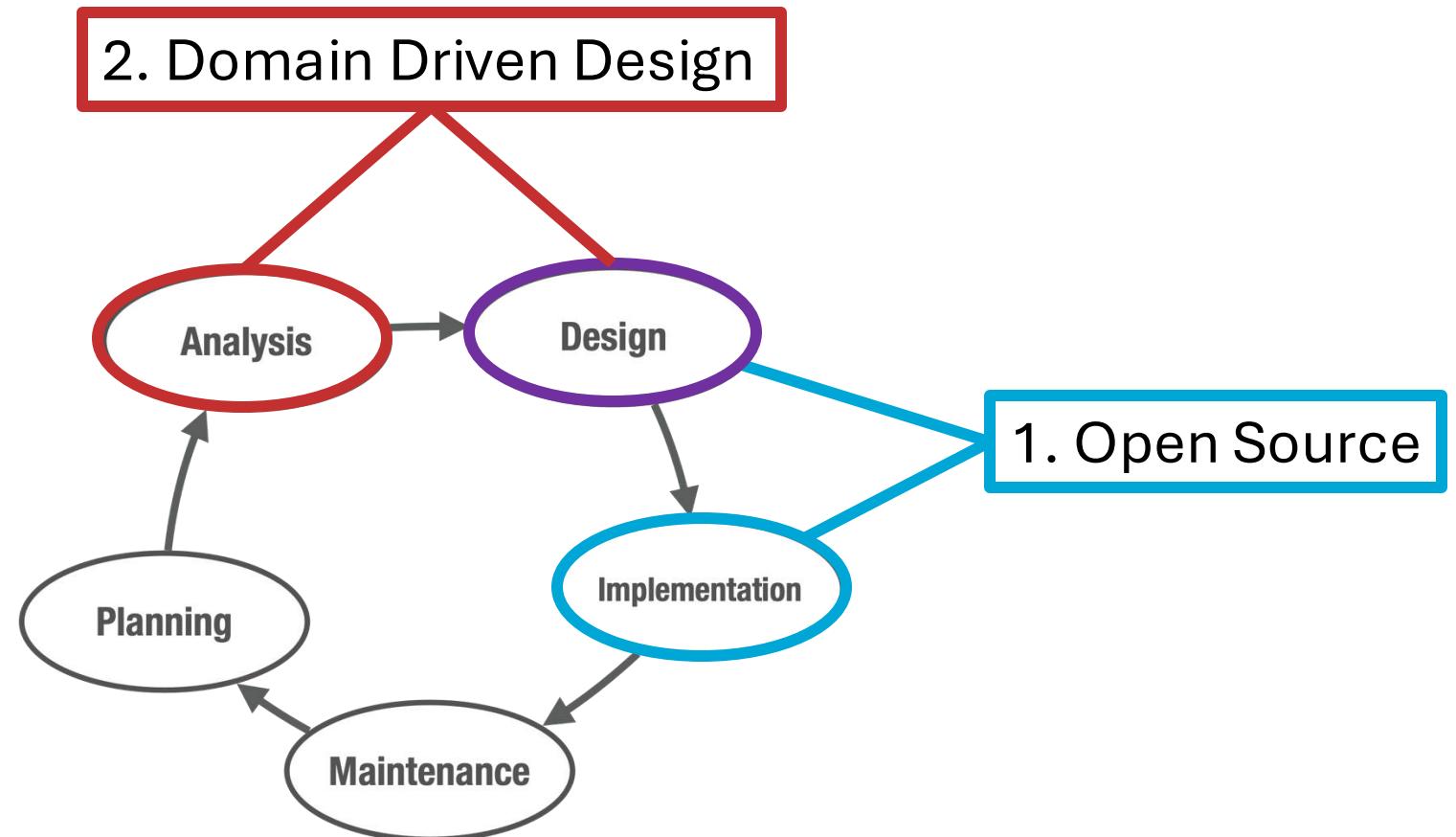
# Question from WebLab

# Other Questions

# Takeaways

- Design Patterns are reusable patterns for specific scenarios
- Design Patterns have clear tradeoffs
- Don't use too many patterns, or complexity will be high

# Next Week



# Working with open-source software: Selecting, using, adapting, and contributing back

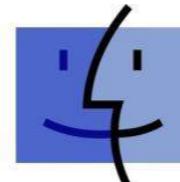
**Diomidis Spinellis**

Professor of Software Analytics  
Department of Software Technology  
Delft University of Technology

<http://www.spinellis.gr/>  
[D.Spinellis@tudelft.nl](mailto:D.Spinellis@tudelft.nl)



FreeBSD



Mac OS

# Rundeck

by PagerDuty



# Roadmap

Open-source software

What it is and why to use it

Selecting

Look at the product and the process

Using

Ways and responsibilities

Adapting

Tread with care

Contributing back

Why and how

# Roadmap



Open-source software

What it is and why to use it

Selecting

Look at the product and the process

Using

Ways and responsibilities

Adapting

Tread with care

Contributing back

Why and how

**django**



FreeBSD

**zlib**



musescore



**curl://**



 **SQLite**

 **R Studio**



 **php**



 **node**



 **APACHE**  
HTTP SERVER PROJECT

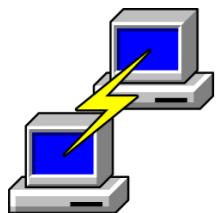
 **Vim**

express

**Drupal™**



 **Symfony**



 **fedora**

**Rundeck**  
by PagerDuty

**WIRESHARK**

 **eclipse**

 **python™**

 **blender®**

 **boost**  
C++ LIBRARIES

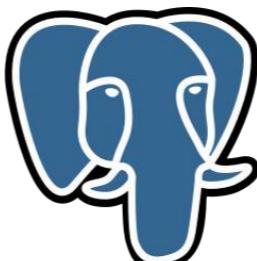
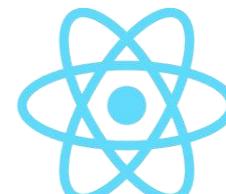
 **git**

 **OpenOffice.org**



 **FFmpeg**

 **TensorFlow**



 **puppet**



 **LibreOffice**  
The Document Foundation

 **Maven™**



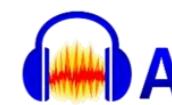
**debian**

 **GNOME™**

 **MySQL®**  
mariadb

 **MariaDB**

 **OpenSSL**  
Cryptography and SSL/TLS Toolkit

 **Audacity®**

# Examples of OSS

- OSs: Linux, FreeBSD, NetBSD, OpenBSD
- Databases: PostgreSQL, MariaDB, SQLite
- Editors: Visual Studio Code, Vim, GNU Emacs
- Compilers: GCC, LLVM
- Programming frameworks: OpenJDK, Mono, OpenCL
- Graphics: Inkscape, GIMP, Blender, Ghostscript
- IDEs: Eclipse, NetBeans, KDevelop
- Text processing: LaTeX, Pandoc, LibreOffice
- Web development: React, Angular, Django, Ruby on Rails, PHP
- App development: React native, Flutter, Xamarin



“curl is used daily by virtually every Internet-using human on the globe.”

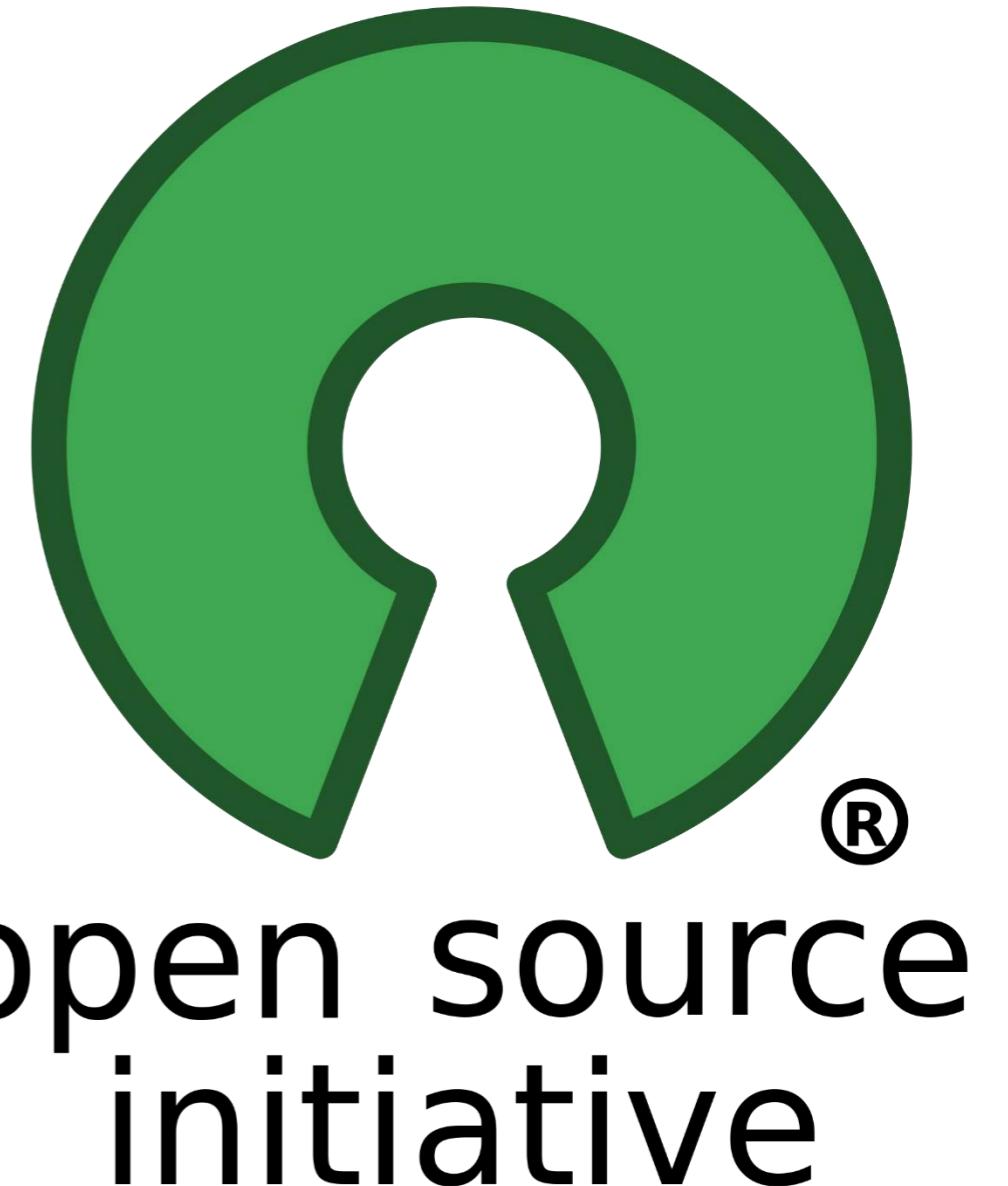
“curl is the Internet transfer engine for thousands of software applications in over **ten billion installations**.”



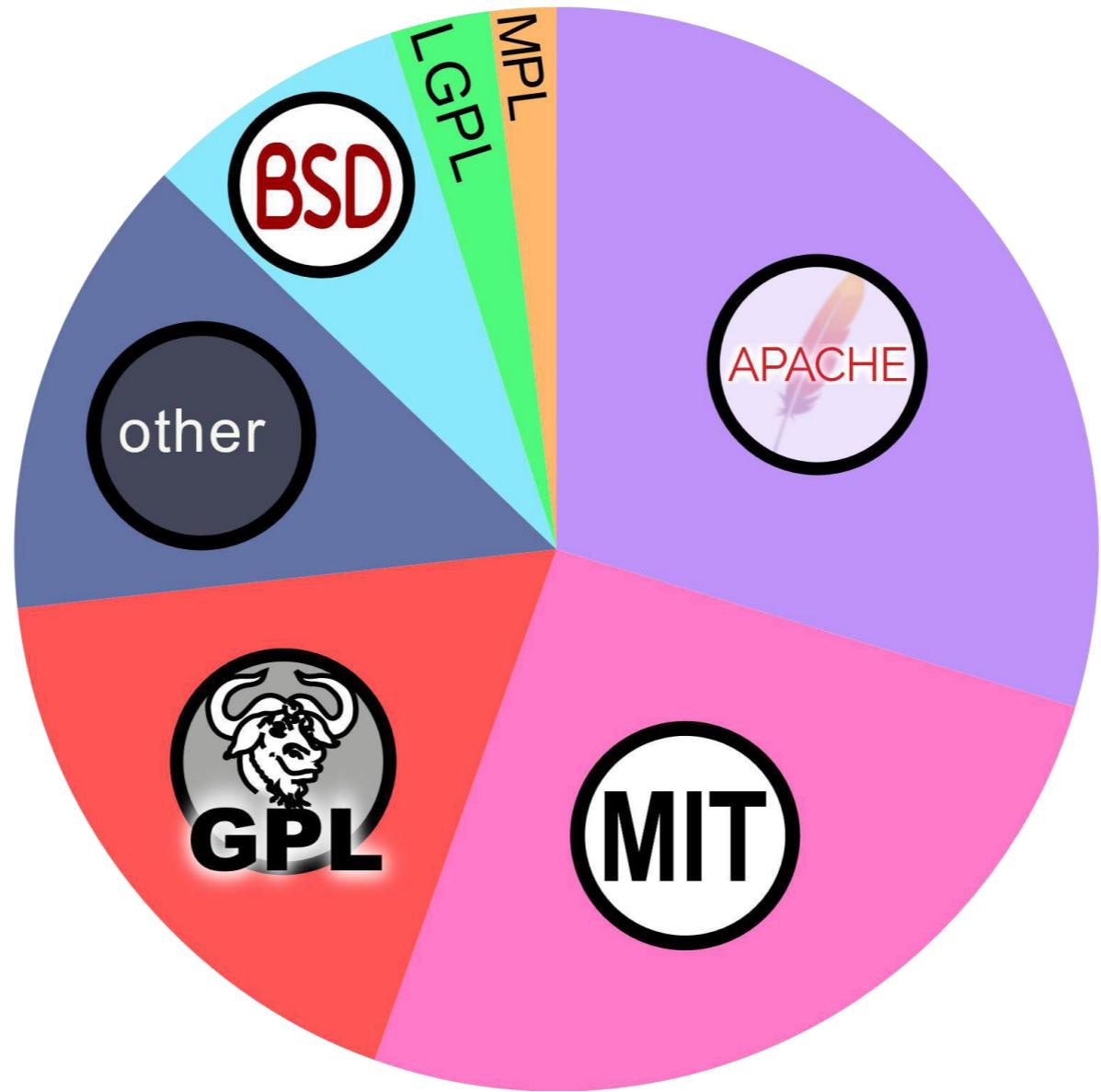
“Single most widely deployed and used software component. Over **one trillion** SQLite databases in active use.”

# Definition

- Free redistribution
- Source code availability
- Derived works
- Integrity of author's code
- No discrimination
- Licensing



# Licensing: permissive vs copyleft



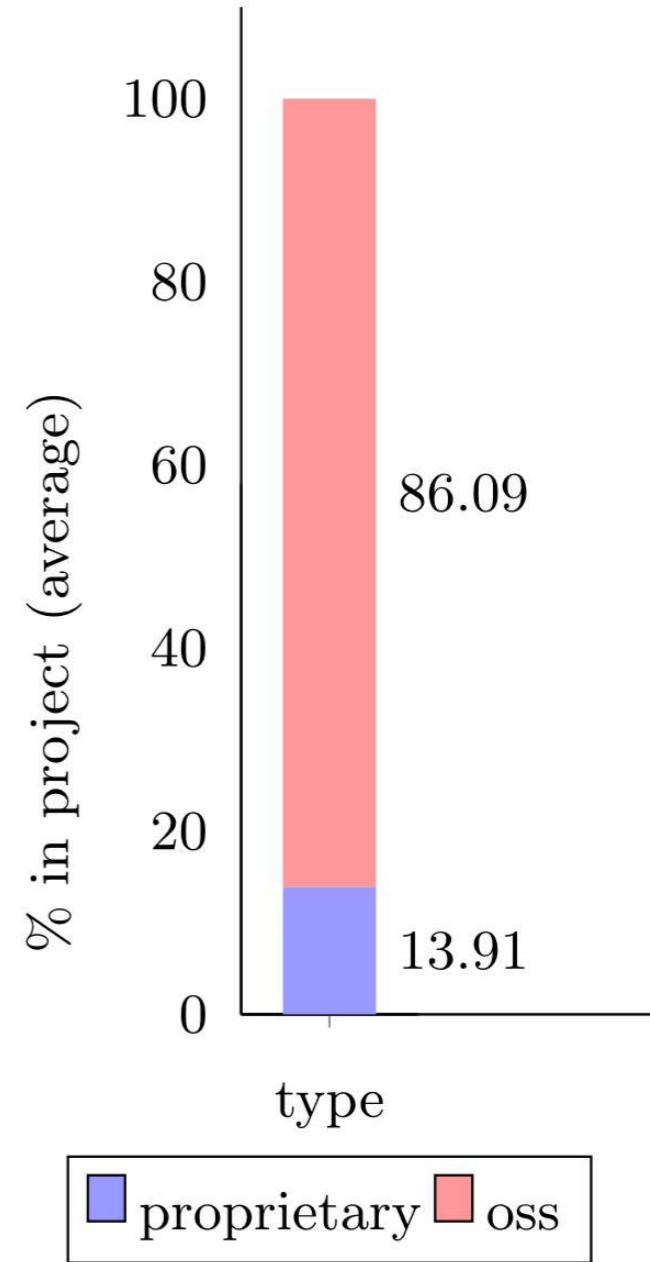
# The meaning of “free”

---



# The 80 / 20 rule

“If most of the code comprising your product or service isn’t open-source software, it’s highly likely that you’re wasting effort and cash reinventing the wheel”



7,024 Java projects developed at SAP

# Support the Guardian

Available for everyone, funded by readers

[Support us →](#)

# The Guardian

News

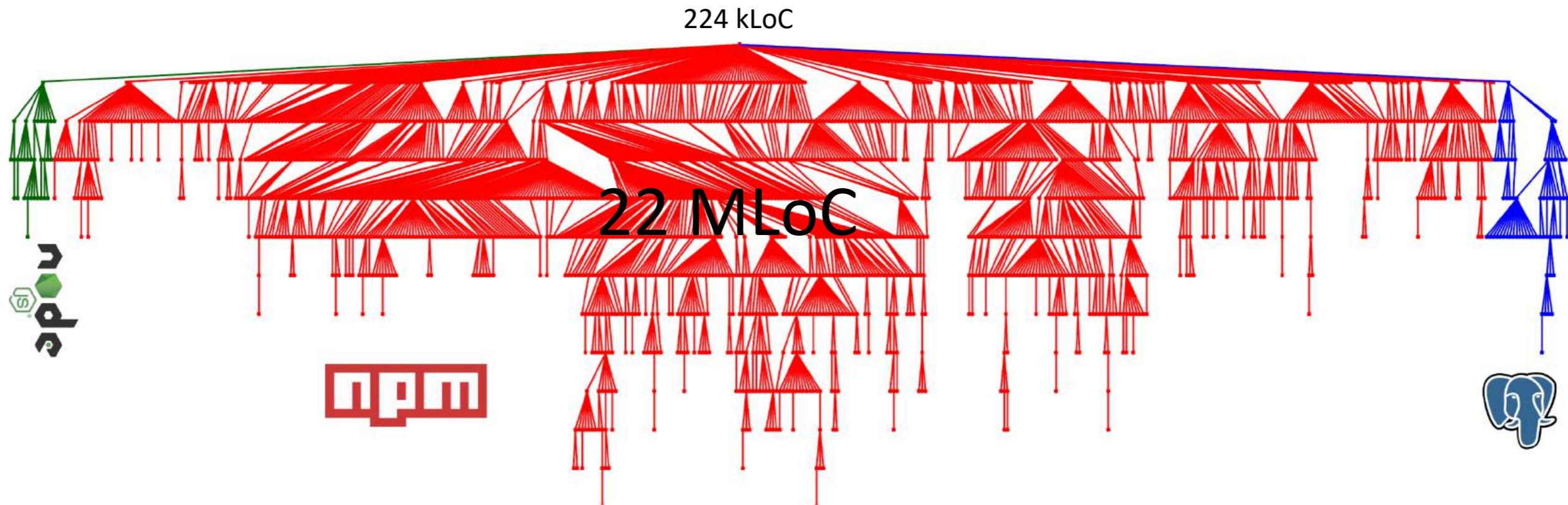
Opinion

Sport

Culture

Lifestyle

More ▾



11 MLoC

# Common myths and misconceptions



lower cost



security



no vendor 'lock in'



better quality

... but you need  
to budget  
(internal or  
external) support.



lower cost



security



no vendor 'lock in'



better quality

- Open to third-party audits
- Open to malicious analysis (less of an issue)
- Fewer resources dedicated to security
- Open to malicious users



lower cost



security



no vendor 'lock in'



better quality

Not all business  
models are  
benign



lower cost



security



no vendor 'lock in'



better quality

The jury is out...



lower cost



security



no vendor 'lock in'

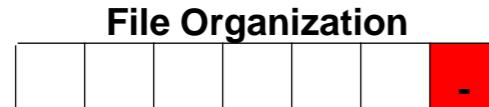


better quality

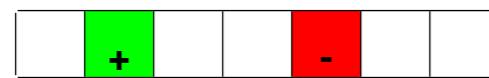
# A tale of four kernels...



FreeBSD



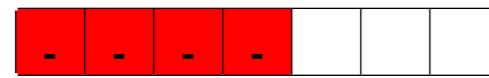
Linux



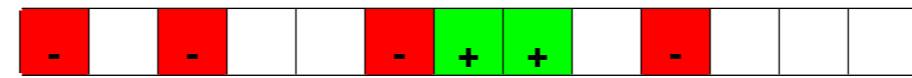
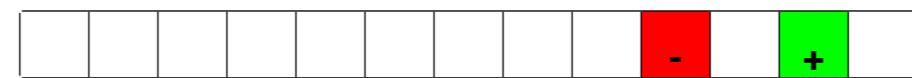
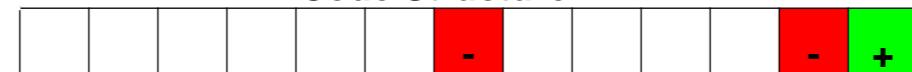
Solaris



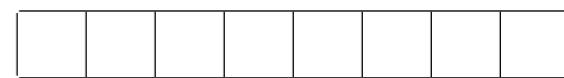
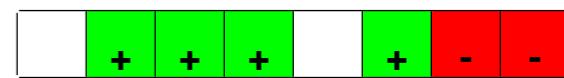
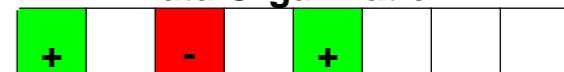
Windows



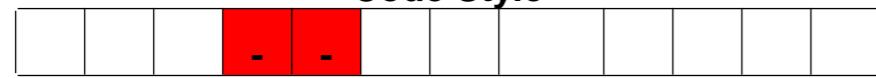
**Code Structure**



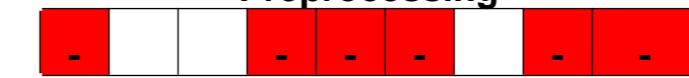
**Data Organization**



**Code Style**



**Preprocessing**



FreeBSD



Linux



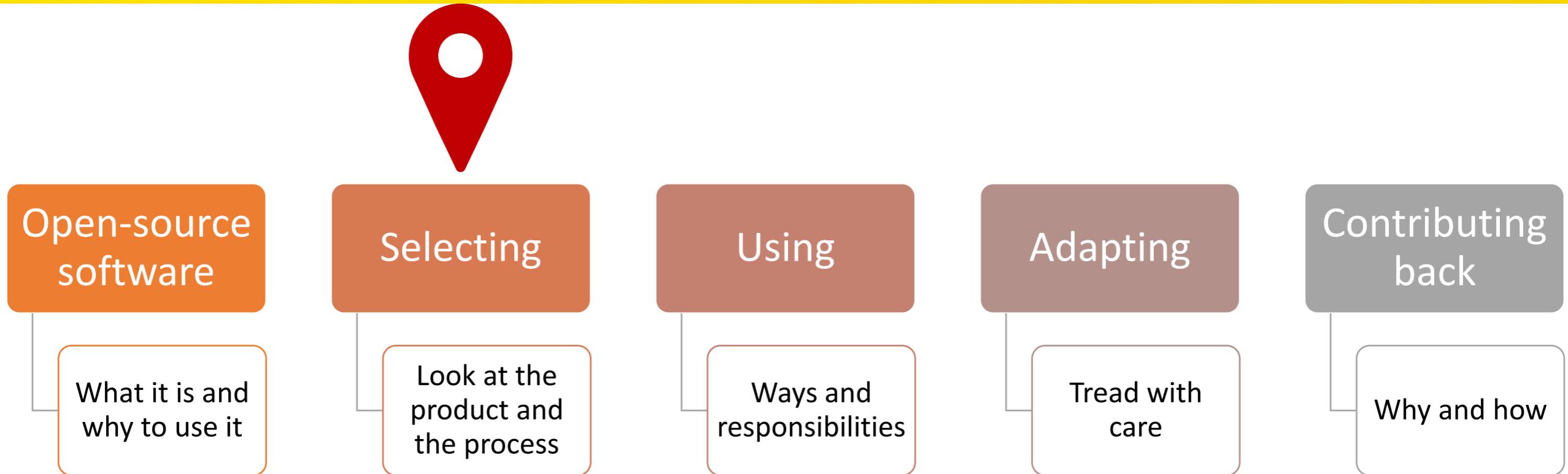
Solaris



Windows



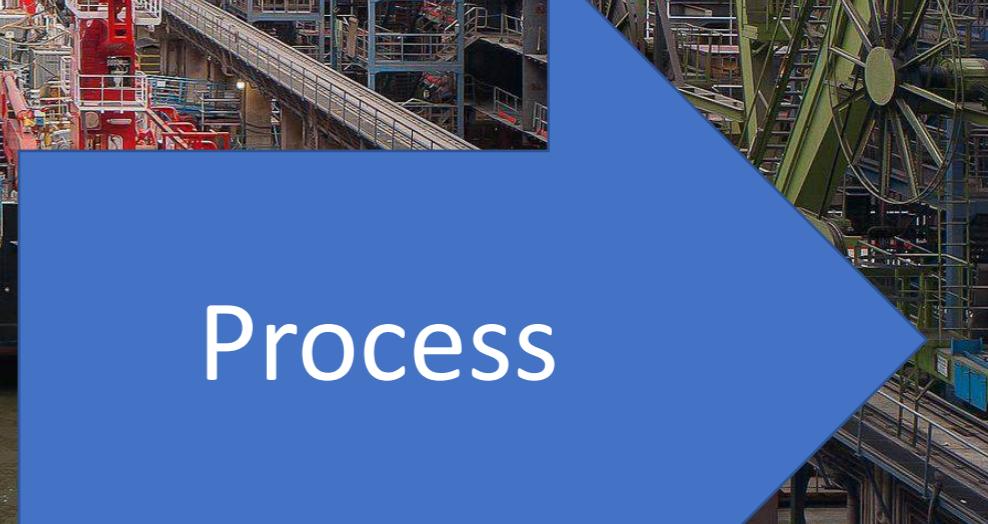
# Roadmap





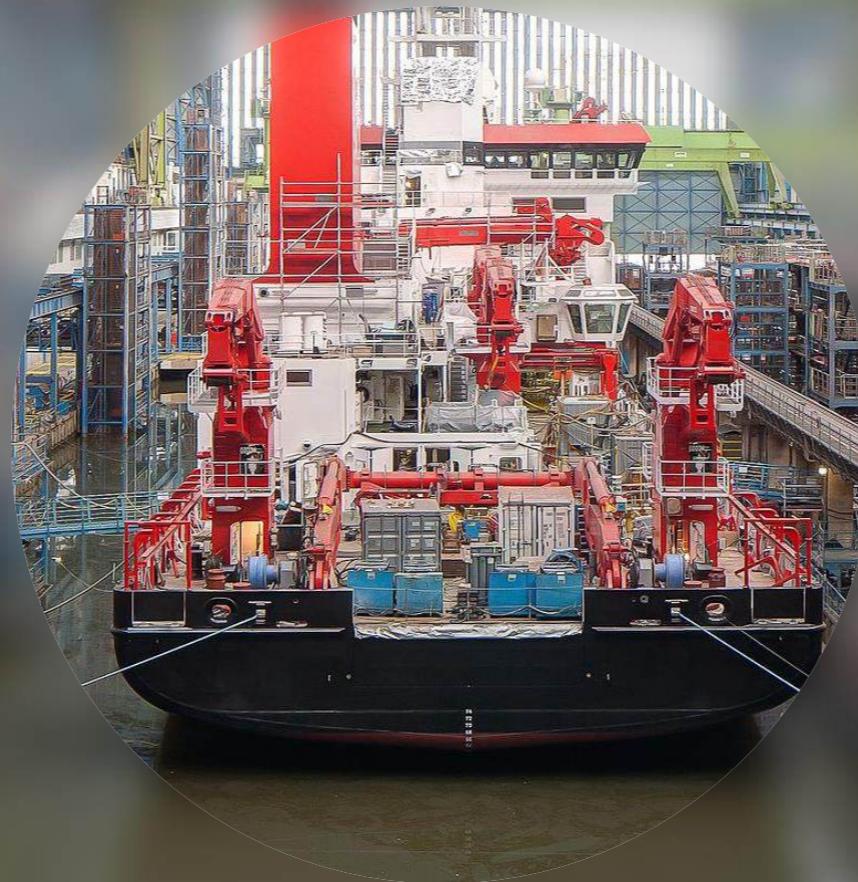
# Finding candidates

- GitHub search
- Package manager search
- Similar and competing products
  - Copyrights
  - Bundled libraries
  - String search
- Q&A forums



Product

Process





# Functionality

- Does it cover our needs?
- Does it cover more than our needs?

# License

Compatible with  
our business  
model?

Compatible with  
other software?

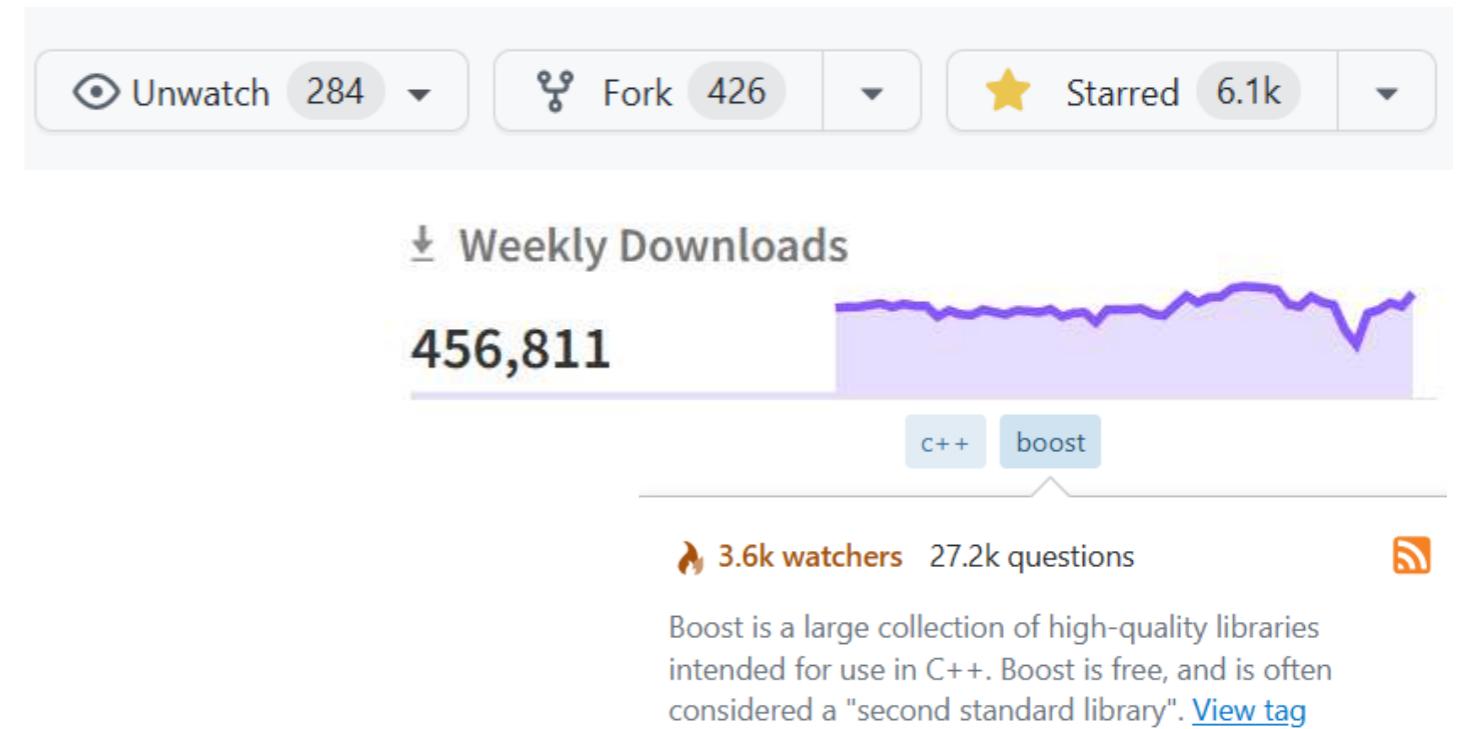
An abstract digital background featuring a 3D grid of binary digits (0s and 1s) in a dark blue color scheme. Several glowing nodes, represented by small circles in red, green, and blue, are connected by thin lines, forming a network-like structure that suggests data flow or connectivity.

# High-level nonfunctional properties

- Will it play along?
  - Processor architecture
  - Operating system
  - Middleware?
- Does it have the required performance?
  - Evaluate your actual workload
- (Now and in the future)

# Popularity

- Community to answer questions
- Volunteers to contribute fixes and enhancements
- A pool of new contributors
- Metrics
  - GitHub stars, forks, watchers
  - npm downloads
  - StackOverflow question tags
  - Google query results





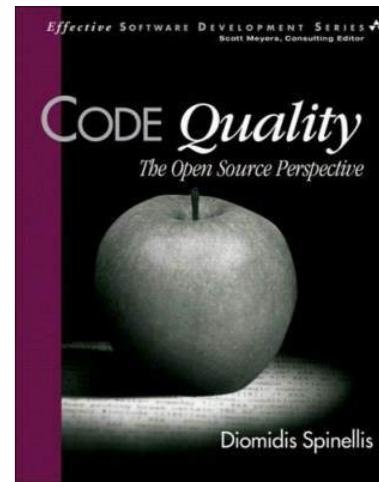
# Documentation

---

- “Use the ~~Force~~-Source, Luke” should be the last resort.
- Technical documentation
  - Installation
  - Maintenance
  - Troubleshooting
- User documentation
  - Tutorials
  - Reference manuals

# Source code

- Is the code's language known within your organization?
- Code quality matters
  - Bugs
  - Security vulnerabilities
  - Poor performance
  - Maintenance problems
- Quick quality check
  - Are files structured according to language's conventions?
  - Unit testing?
  - Non-source elements in the repo?
  - Short and readable routines?
  - Well-chosen identifiers?
  - Reasonably commented?
  - Consistent formatting?





# Build process

- Important
  - Insurance for bug fixes
  - Build-from-source policy
- Quick check
  - Documented build procedure?
  - Do we have the required tooling?
  - Can we build from source?
- Also examine dependencies



ChristianSchd, CC BY-SA 3.0, via Wikimedia Commons



ChristianSchd, CC BY-SA 3.0, via Wikimedia Commons

# Project governance

---

- Do-ocracy
  - “Just Do It”; “Shut up and code”; “Put your code where your mouth is”
- Founder-leader (BDFL)
- Self-appointed council or board
- Electoral
- Corporate-backed
  - Walled garden; CLA
- Foundation-backed
  - Can also act as an umbrella



Photo by Aditya Joshi on Unsplash

# Development workflow

- Does the project practice CI?
  - Look for .github/workflows, travis.yml, JenkinsFile, ...
- What does CI entail?
  - Unit testing
  - Static analysis
  - Test coverage analysis
  - Spell-check documentation
  - Check for dead links
- Also consider GitHub badges

build passing coverage 97%

frontend passing pre-commit passing api passing

	COMMENT	DATE
O	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
O	ENABLED CONFIG FILE PARSING	9 HOURS AGO
O	MISC BUGFIXES	5 HOURS AGO
O	CODE ADDITIONS/EDITS	4 HOURS AGO
O	MORE CODE	4 HOURS AGO
O	HERE HAVE CODE	4 HOURS AGO
O	AAAAAAA	3 HOURS AGO
O	ADKFJSLKDFJSDFKJ	3 HOURS AGO
O	MY HANDS ARE TYPING WORDS	2 HOURS AGO
O	HAAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.



# Code commits

---

- Regular commits?
- By how many committers?
- Are commits clearly labeled?
- Do they include descriptions?
- Do they employ meta-data conventions?
- Do they reference issues?
- Is there evidence of reviews and discussion?

# Project releases

---

- Regular?
- Recent?
- Evidence of maintenance?
- Is there a security / critical fix release channel?
- Is there a stable long term support (LTS) branch?
- Does it match your project's pace?

Protocol	Location	Latest Release
HTTP	<a href="https://www.kernel.org/pub/">https://www.kernel.org/pub/</a>	
GIT	<a href="https://git.kernel.org/">https://git.kernel.org/</a>	
RSYNC	<a href="rsync://rsync.kernel.org/pub/">rsync://rsync.kernel.org/pub/</a>	
		<b>6.1.11</b> 
mainline:	<a href="#">6.2-rc7</a>	2023-02-05 [tarball] [patch] [inc. patch] [view diff] [browse]
stable:	<a href="#">6.1.11</a>	2023-02-09 [tarball] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog]
stable:	<a href="#">6.0.19 [EOL]</a>	2023-01-12 [tarball] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog]
longterm:	<a href="#">5.15.93</a>	2023-02-09 [tarball] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog]
longterm:	<a href="#">5.10.167</a>	2023-02-06 [tarball] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog]
longterm:	<a href="#">5.4.231</a>	2023-02-06 [tarball] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog]
longterm:	<a href="#">4.19.272</a>	2023-02-06 [tarball] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog]
longterm:	<a href="#">4.14.305</a>	2023-02-06 [tarball] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog]
linux-next:	<a href="#">next-20230209</a>	2023-02-09 [tarball] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog] [browse]

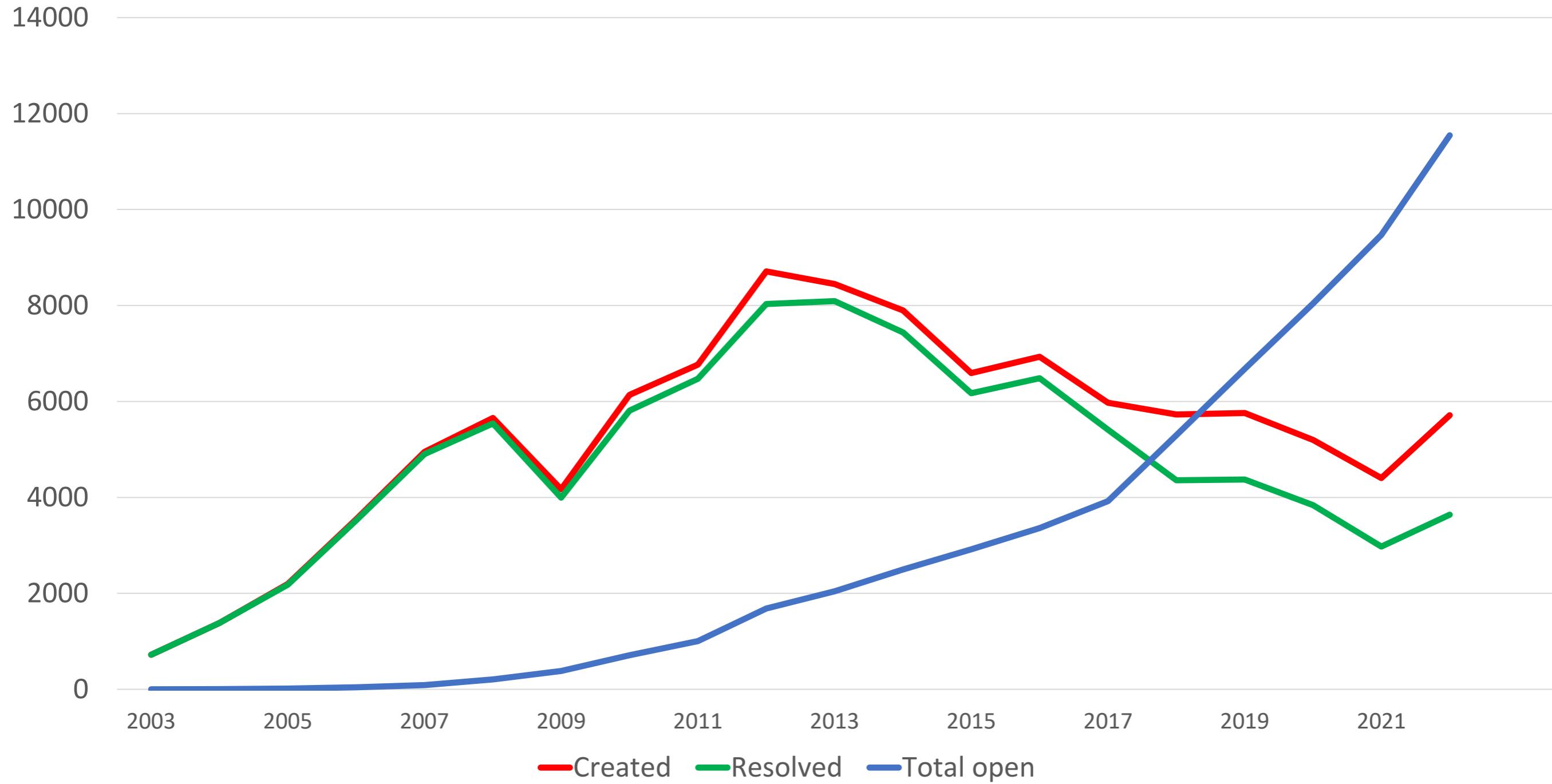


# Support

---

- Online forum?
- Mailing list?
- Chat group?
- Do answers arrive quickly?
- Are respondents supportive and friendly?
- What paid support options are available?
  - Companies, products, services, consultants

# Moodle LMS Issues





# Issue handling

---

- Dedicated issue management platform?
- Are issues resolved quickly?
- How many issues are lying open?
  - Are issues closed automatically?
- How serious are these to you?
- What's the ratio between open and closed issues?



# Contributions

---

- Outside contributions accepted?
- Contributor's guide?
- Contribution steps?
  - Signed agreement?
  - Code review?
- GitHub pull requests?
- Evidence of accepted / neglected contributions

# Putting it all together...

Dealbreakers

Indicators of needed in-house investment

Important for in-house development

# Potential dealbreakers

## Product

- Functionality
- Licensing
- Non-functional properties
- Popularity
- Documentation
- Code quality
- Build system

## Process

- Project governance
- Development workflow
- Code commits
- Project releases
- Support
- Issue management
- Acceptance of contributions

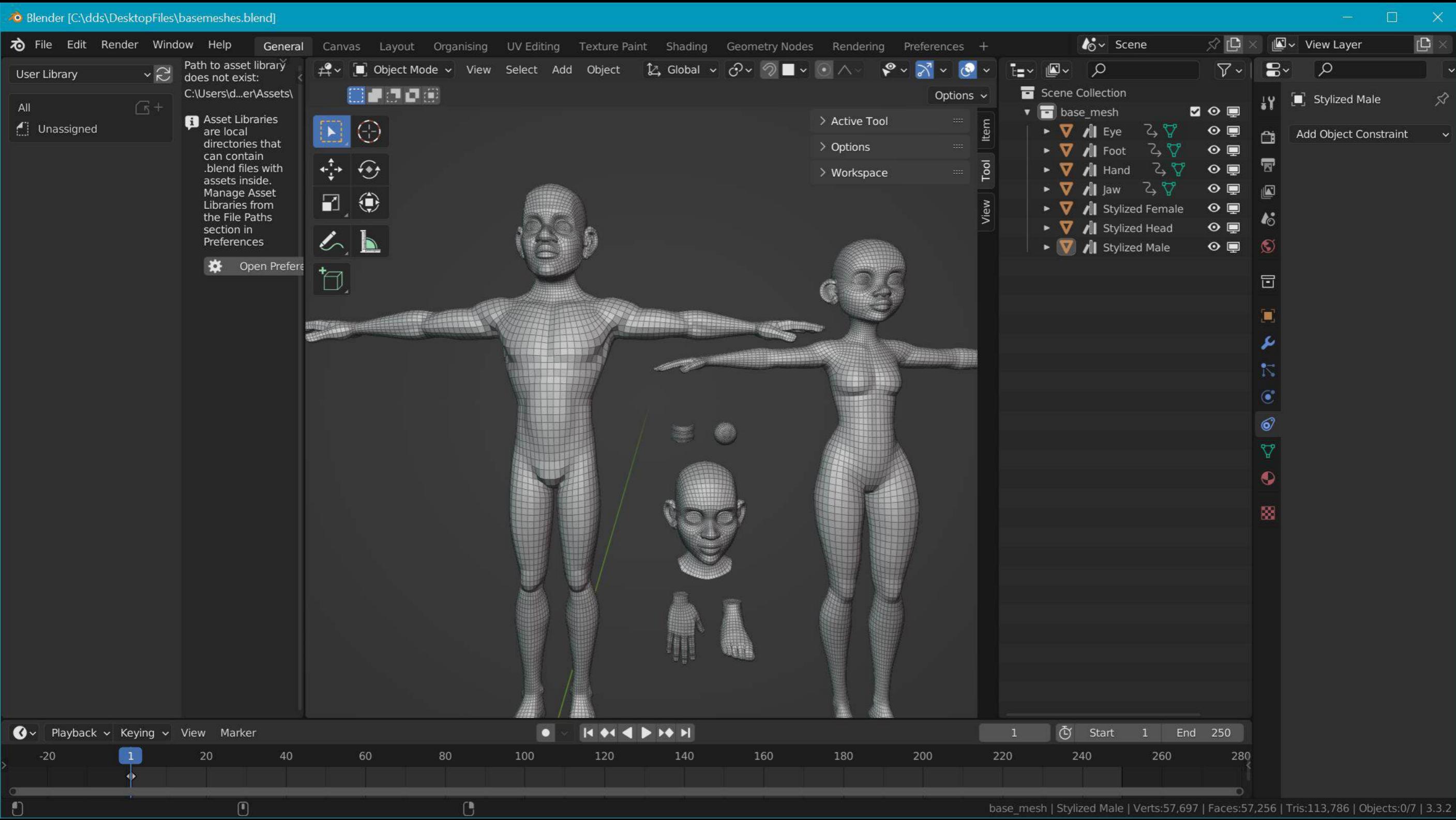
# Indicators of needed in-house investment

## Product

- Functionality
- Licensing
- Non-functional properties
- Popularity
- Documentation
- Code quality
- Build system

## Process

- Project governance
- Development workflow
- Code commits
- Project releases
- Support
- Issue management
- Acceptance of contributions





Please describe the blender cycles api



The Blender Cycles API is a set of programming interfaces that allow developers to create custom functionality and integrations with Blender's Cycles rendering engine. The API provides access to a wide range of features and functionalities within the Cycles engine, including the ability to create and manipulate objects, materials, and scenes; access and modify rendering settings; and create custom shaders and other rendering effects. The API is written in C++ and is designed to be easy to use for developers with experience in 3D graphics programming. It can be used to create custom add-ons for Blender, as well as to integrate Cycles into other 3D software applications.



Where is it documented?



The Blender Cycles API is documented in the Blender Manual, which is available on the Blender website. The manual contains detailed information about the various classes, functions, and other programming elements that make up the API, as well as tutorials and examples that show how to use the API to create custom tools and integrations with Cycles. You can access the Blender Manual by visiting the following URL: <https://docs.blender.org/manual/en/latest/render/cycles/index.html>.



This is the user documentation, not the API documentation.



You are correct, the information I provided earlier was about the user documentation for Blender's Cycles rendering engine, not the API documentation. The API documentation for the Blender Cycles API can be found in the Blender source code, which is available on the Blender GitHub page. The source code is organized into directories that correspond to the different components of the Blender software, and within each directory there are files that contain the documentation for the APIs associated with that component. For example, the API documentation for the Cycles rendering engine can be found in the `source/blender/render/cycles` directory.



The Blender repo /source/blender/render does not contain a cycles directory.



I apologize for the incorrect information. The directory structure for the Blender source code may have changed since I last looked at it. As a large language model trained by OpenAI, I don't have the ability to browse the internet or access up-to-date information about specific software projects. My responses are based on the information and knowledge that I was trained on, which has a fixed cutoff date.



Anyway, give me C++ code to render a cube using the Cycles renderer.



Here is an example of how you might use the Blender Cycles API to render a cube in C++:

rust

```
#include "cycles/cycles_api.h"

int main()
{
    // Initialize the Cycles API
    if (!ccl::CYCLES_API_INITIALIZE())
    {
        // Handle error
    }

    // Create a new scene
    ccl::Scene *scene = new ccl::Scene();

    // Create a cube mesh
    ccl::Mesh *cube = ccl::create_cube();

    // Create a material for the cube
    ccl::Material *mat = new ccl::Material();
    mat->set_diffuse(ccl::make_float3(1, 0, 0)); // Red color

    // Add the cube to the scene
    ccl::Object *obj = new ccl::Object();
    obj->add_mesh(cube, mat);
    scene->add_object(obj);

    // Set the scene for rendering
    ccl::RenderSettings settings;
    settings.set_resolution(1024, 768); // Set the output resolution
    settings.set_samples(1024); // Set the number of samples
    ccl::Renderer *renderer = new ccl::Renderer(scene, settings);

    // Render the scene
    renderer->render();
}
```

# Important for in-house development

## Product

- Functionality
- Licensing
- Non-functional properties
- Popularity
- Documentation
- Code quality
- Build system

## Process

- Project governance
- Development workflow
- Code commits
- Project releases
- Support
- Issue management
- Acceptance of contributions

# Roadmap





# You can use open-source software as...

---

- ... a (supported) product
- ... a service
  - SaaS through a cloud provider
  - Self-hosted
- ... a dependent package
  - Smooth(er) dependencies and updates
- ... code installed / built from source
  - Consider dependencies and updates
- ... copied file / routine / class / lines
  - Be even more careful regarding licensing



# Package Ecosystem

- Package management system
- Distribution network
- Contributors

# Package ecosystems are built around:

An operating system distribution

- GNU/Linux: Debian, Fedora, Ubuntu, ...
- FreeBSD, NetBSD, OpenBSD

Package repository / manager

- Maven / mvn, Gradle
- PyPI / pip
- NPM / npm, yarn
- CRAN
- RubyGems / gem



An activity

- Data science (Python Anaconda)

Software  
components  
and  
dependencies

---

Uploading

---

Distribution

---

Discovery

---

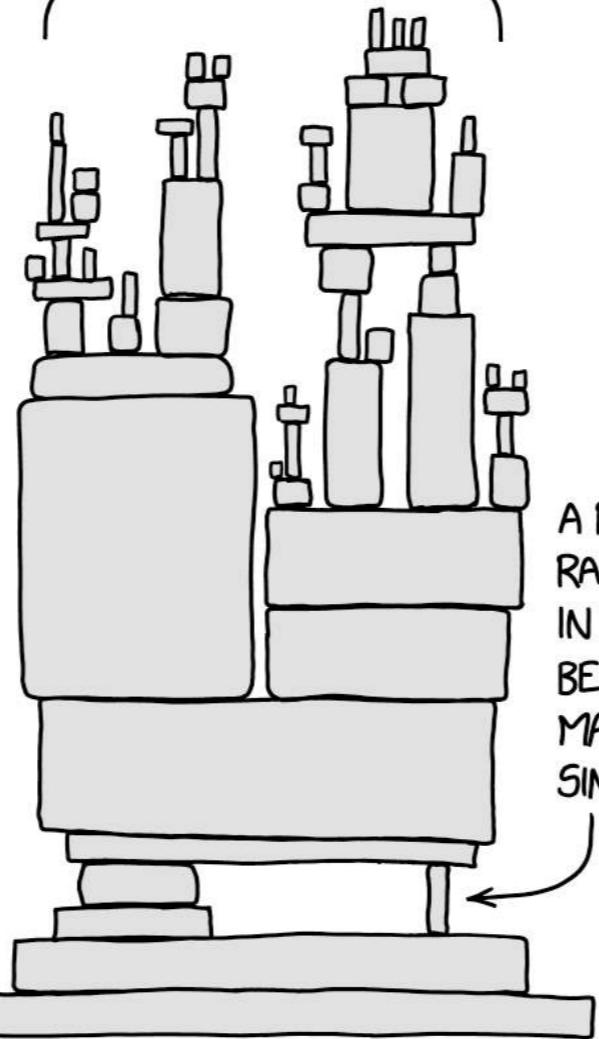
Installation

---

Upgrading

---

ALL MODERN DIGITAL  
INFRASTRUCTURE



A PROJECT SOME  
RANDOM PERSON  
IN NEBRASKA HAS  
BEEN THANKLESSLY  
MAINTAINING  
SINCE 2003

# Software Supply Chain Risks

## Endogenous

- Bugs
- Security vulnerabilities

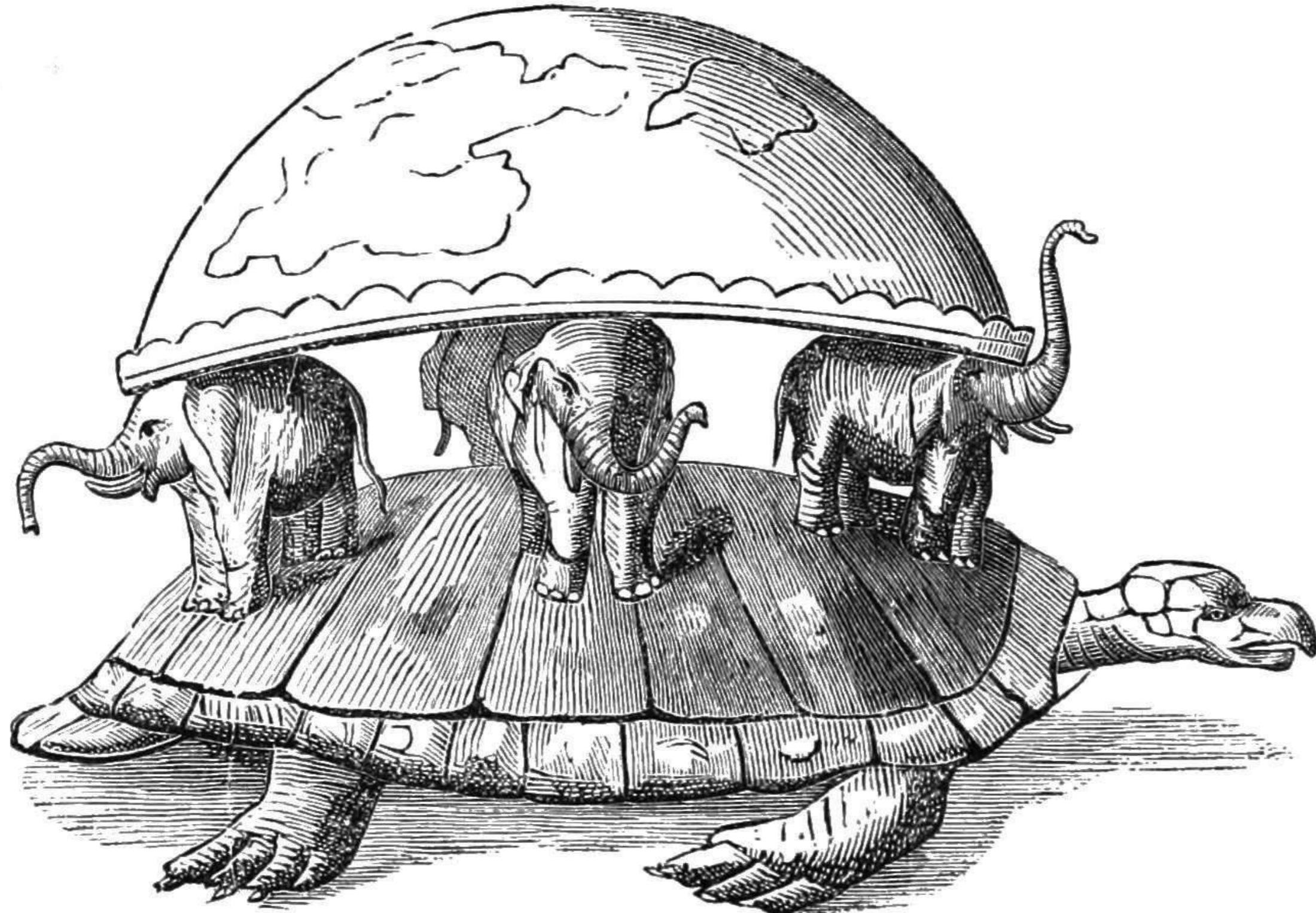
## Exogenous

- Malicious contributors
- Geopolitical rifts (FDPR)
- Critical single points of failure



# Examples

- Heartbleed SSL vulnerability
- XcodeGhost hack
- Equifax leak
- SolarWinds exploit
- Log4j vulnerability



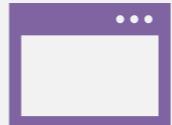
# Occurrence probability and impact



**80%** of code comes from  
libraries and frameworks



**25%** of library downloads have  
known security vulnerabilities



**77%** of web sites are using at  
least one vulnerable library

# Risk Management Tasks



## Assessment

Risk identification

Analysis

Prioritization



## Control

Risk planning

Resolution

Monitoring

# Roadmap



Open-source software

What it is and why to use it

Selecting

Look at the product and the process

Using

Ways and responsibilities

Adapting

Tread with care

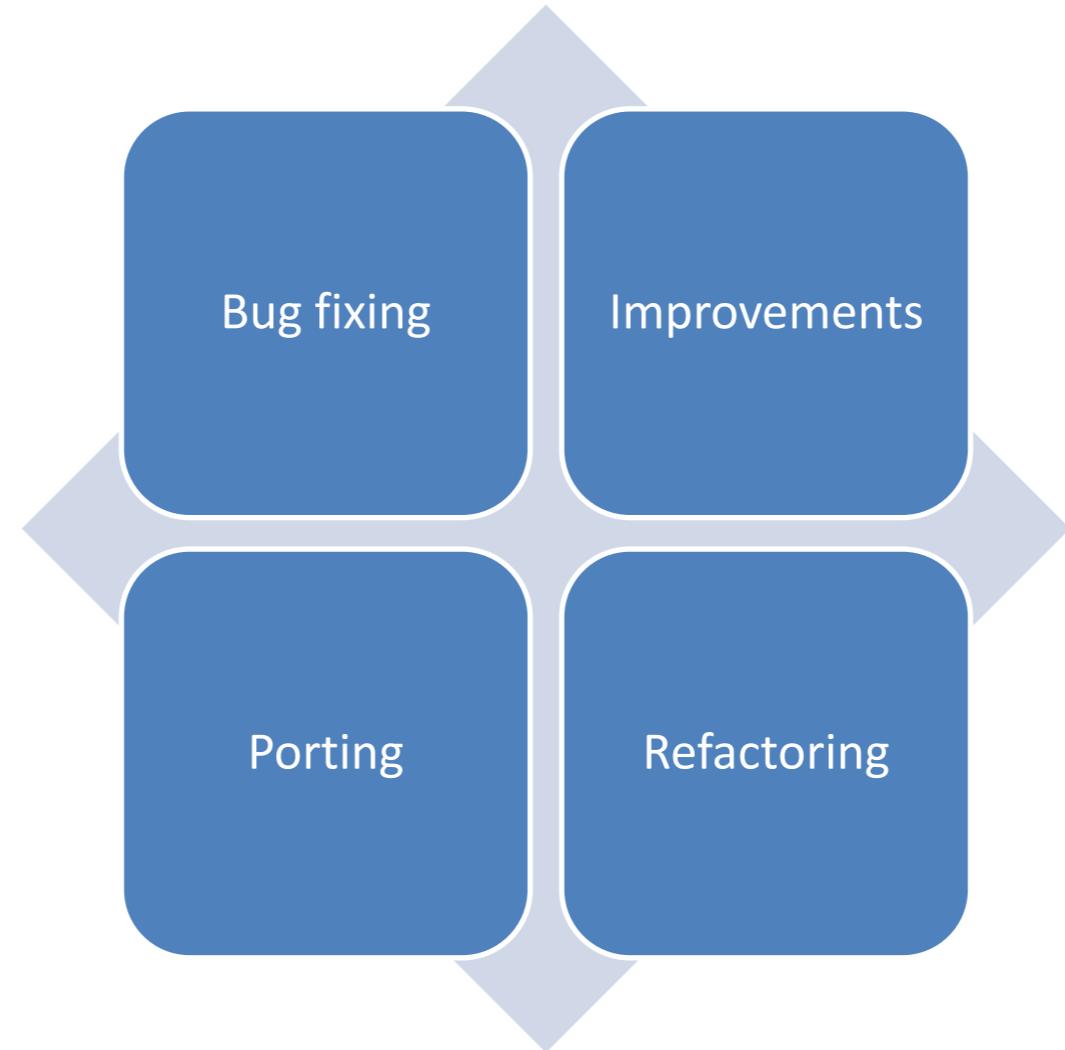
Contributing back

Why and how

# OSS adaptation methods

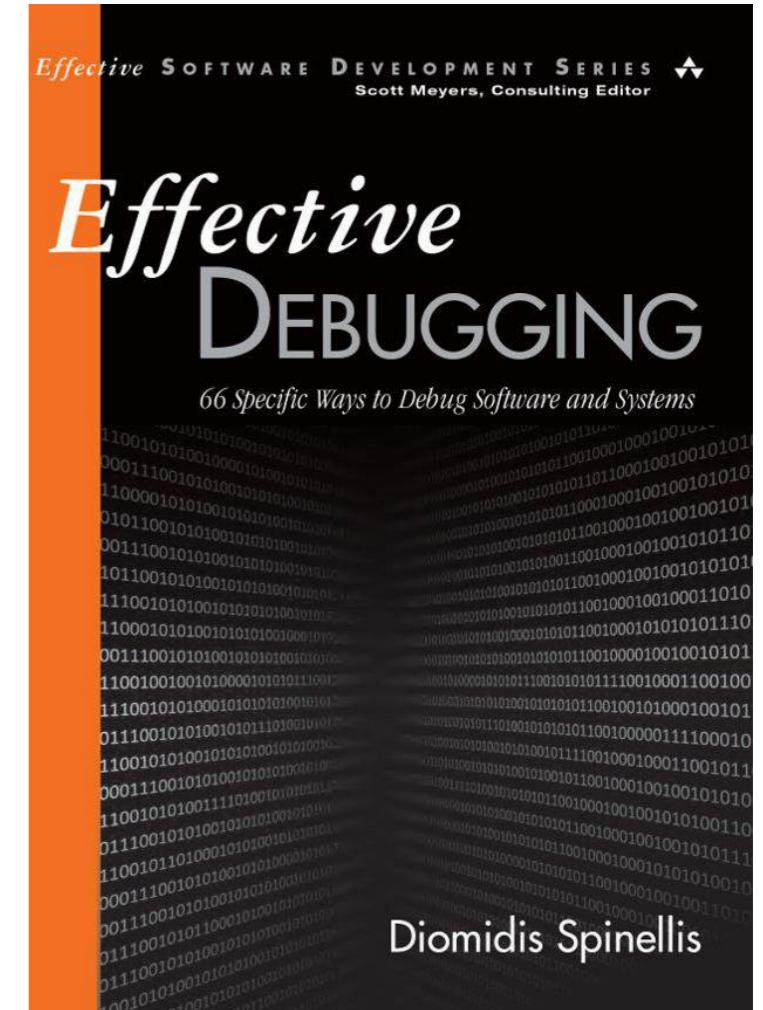
- Configuration
- Contributed change
- Internal change
- Shim
- Plugin
- Fork

# HOWTO



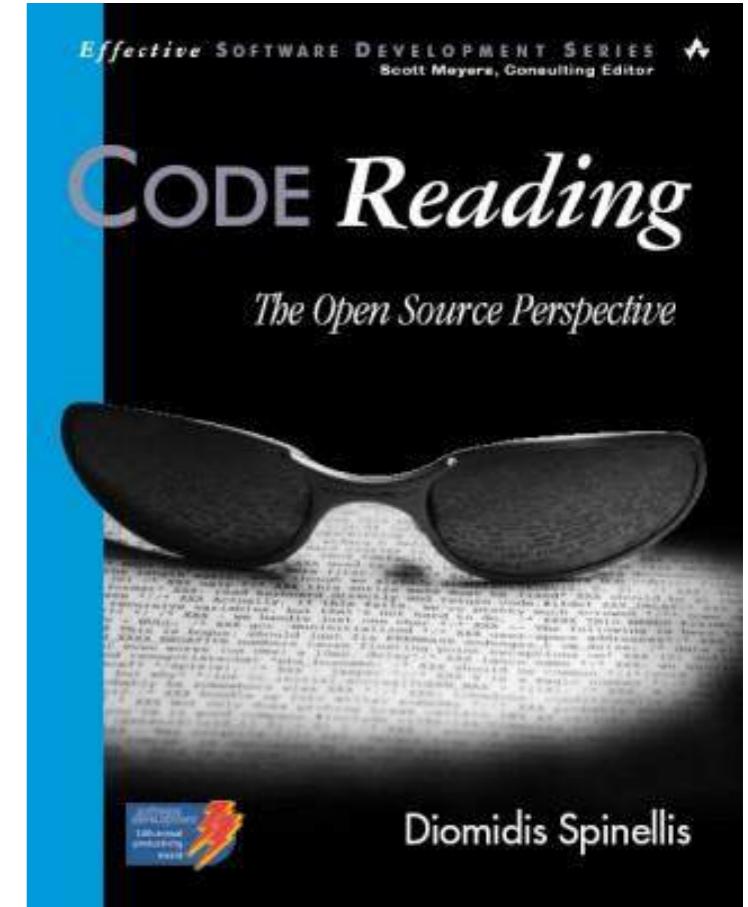
# Bug fixing

- Search for an existing issue — consider an update
- Look at recent changes (git-log / git-blame)
- Construct MWE / test case
- Examine log files; add logging statements
- Use tools
  - Debugger (stack tracing, code/data breakpoints, single stepping)
  - Static analysis tools (cc –Wall, SpotBugs, pylint, Coverity, ...)
  - System call tracing
  - Network packet monitoring



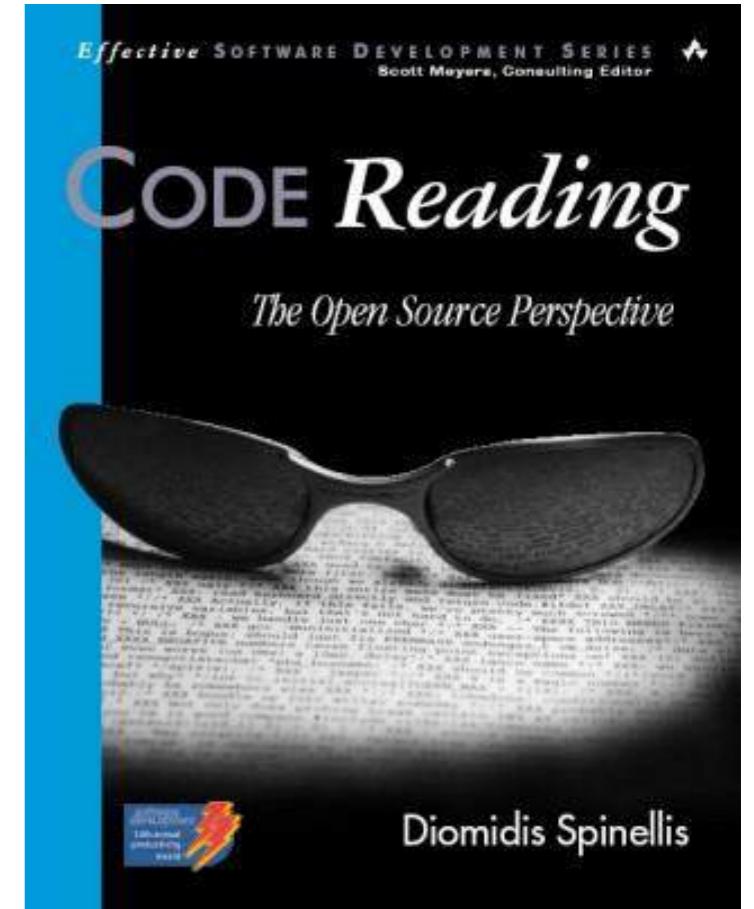
# Enhancements

- Don't panic!
- Locate the code you want to change
  - Search for message / comment / identifier
  - Break program execution in a debugger
  - Browse code top-down
- Understand and change the code in isolation
- [Adjust related parts]



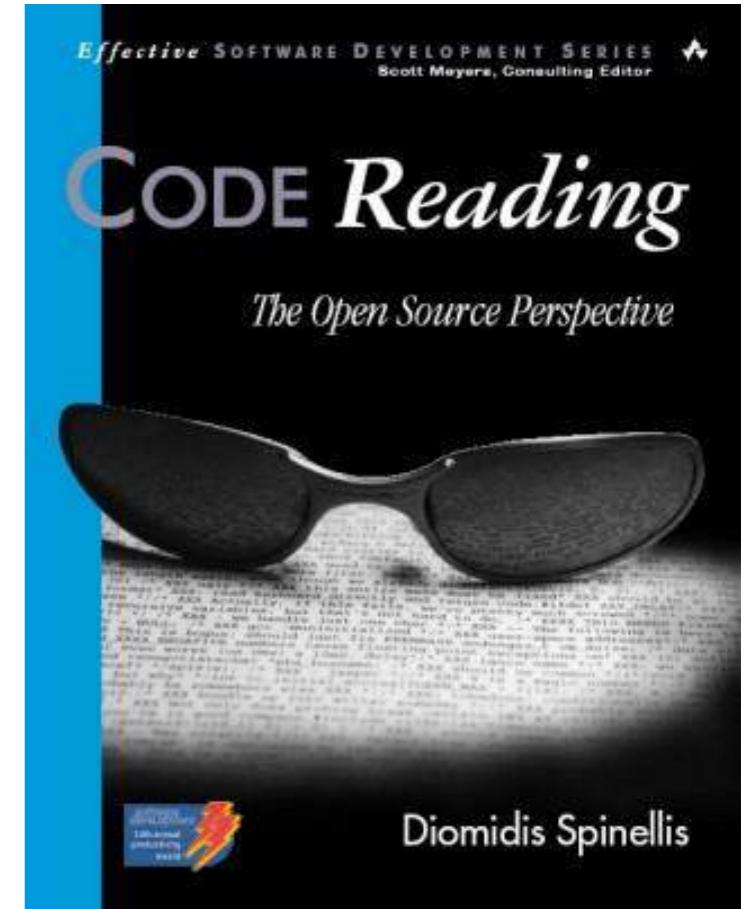
# Porting

- Add or create an interface layer
- Brute force:
  - Follow the compiler / linker error messages

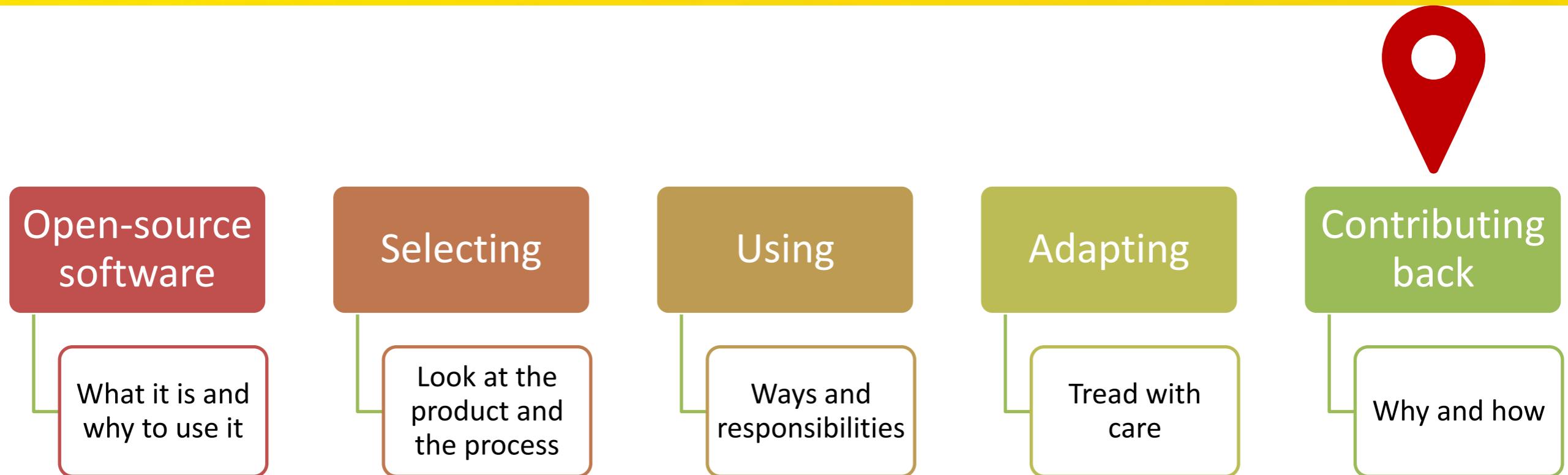


# Refactoring

- Prepare the ground
  - Ensure code is covered by unit tests
  - Ensure functionality is covered by integration tests
  - Add missing tests
- Action time!
  - Make one small improvement
  - Test
  - Commit
  - Repeat



# Roadmap



# Why contribute?



To integrate our  
changes with the  
OSS



For learning and  
professional  
advancement



To give back to  
the community

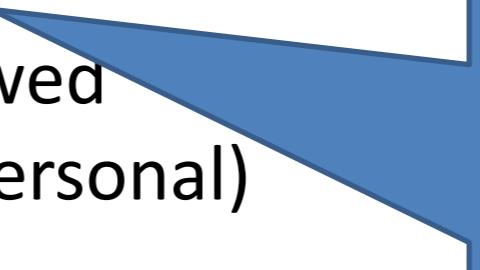
# Integrating fixes and improvements

**Eliminate** local configuration

**Benefit** from project's quality processes (CI, reviews)

**Ensure** maintenance of changes

# How to contribute

- Scratch your own itch
  - Start small
  - Communicate with team before making extensive changes
  - Open an issue (after checking existing)
  - One **clean** commit per issue
  - Don't mix style with functional changes
  - Follow the project's practices
  - Have your contribution reviewed
  - Be open to critique (it's not personal)
  - Be patient
- 
- Learn and follow the project's code style
  - Adopt the project's communication methods
  - Follow the project's integration guidelines
  - Create unit / regression tests
  - Measure performance

# Why contribute?



To integrate our  
changes with the  
OSS



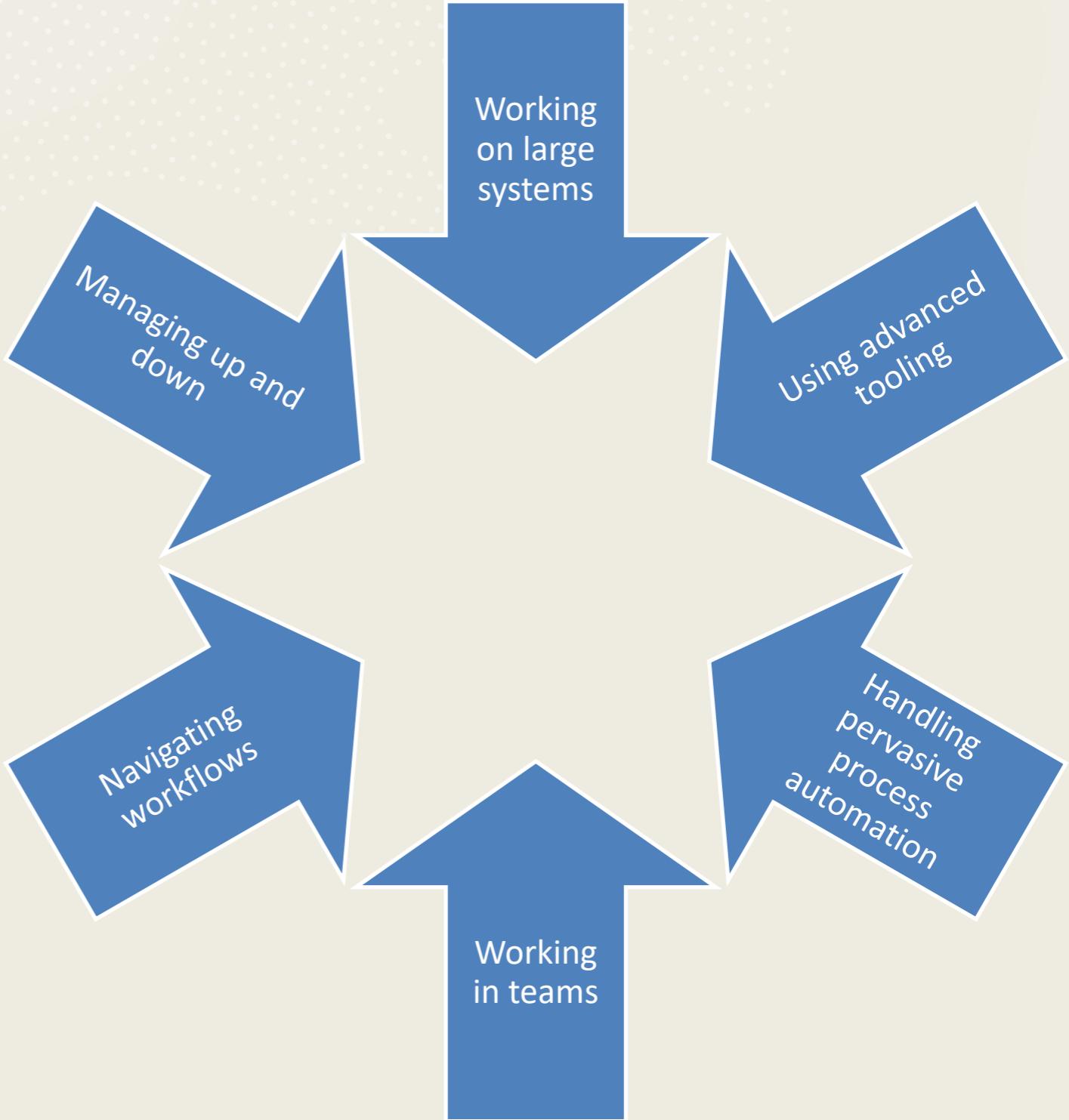
For learning and  
professional  
advancement



To give back to  
the community



# Modern software developers are not (just) programmers



# Contributing (to OSS) is the new coding

---

Social skills

Organizational  
skills

Technical  
knowledge  
beyond coding

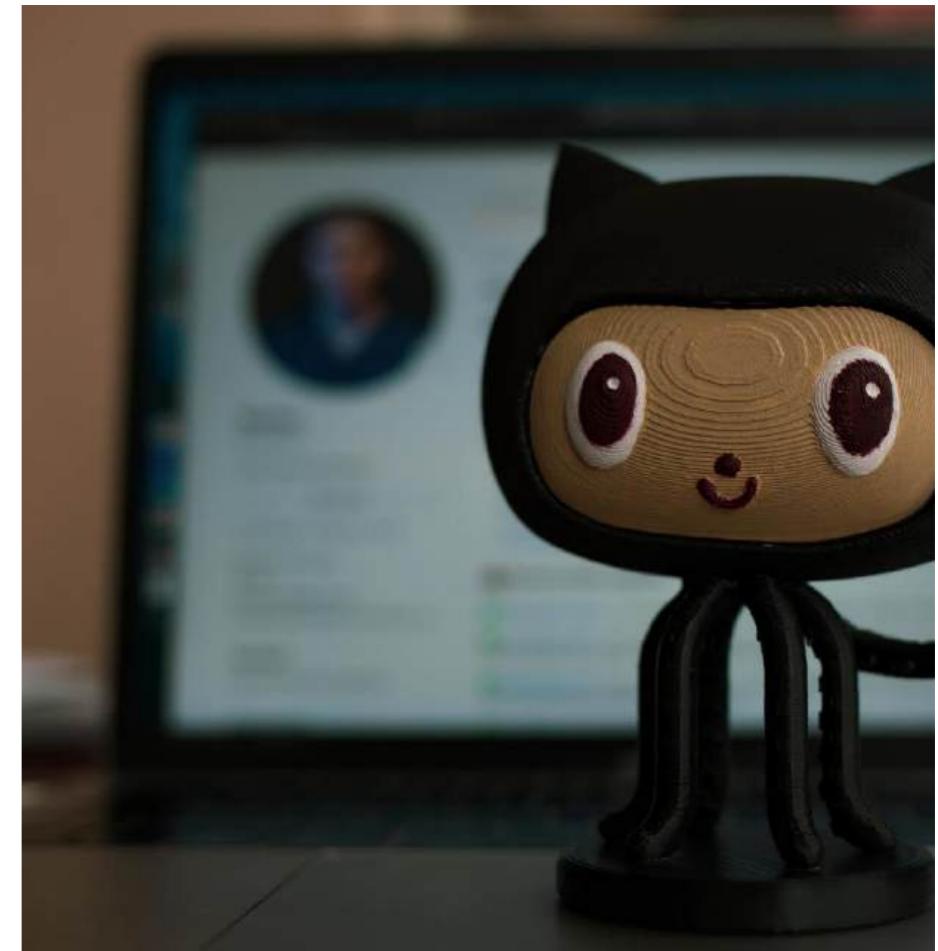


Photo by [Roman Synkevych](#) on [Unsplash](#)

# Social and organizational skills

---

- Developing a sense of context
- Interacting with a project's global and diverse community
- Negotiating feature requests, requirements, and implementation choices
- Dealing with communication problems
- Appreciating the software as a product
- Receiving, discussing, and addressing code review comments



# Planning lessons

---

- Navigating through a project's assets
- Evaluating swiftly the product and process quality of software systems or components
- Configuring, building, running, and debugging third-party code
- Setting up and running software intensive systems with diverse software and hardware requirements
- Choosing realistic contribution goals
- Reading third-party code to identify where their additions or fixes need to be made



Photo by [Brooke Cagle](#) on [Unsplash](#)

# Coding lessons

---

- Modifying a large third-party system by adding a new feature or fixing a bug
- Writing tests
- Working with software systems developed using multiple programming languages and tools
- Documenting their work, typically using a declarative markup language
- Following sophisticated version control workflows
- Passing pre-commit and CI checks and tests



Photo by [Brooke Cagle](#) on [Unsplash](#)

# Why contribute?



To integrate our  
changes with the  
OSS



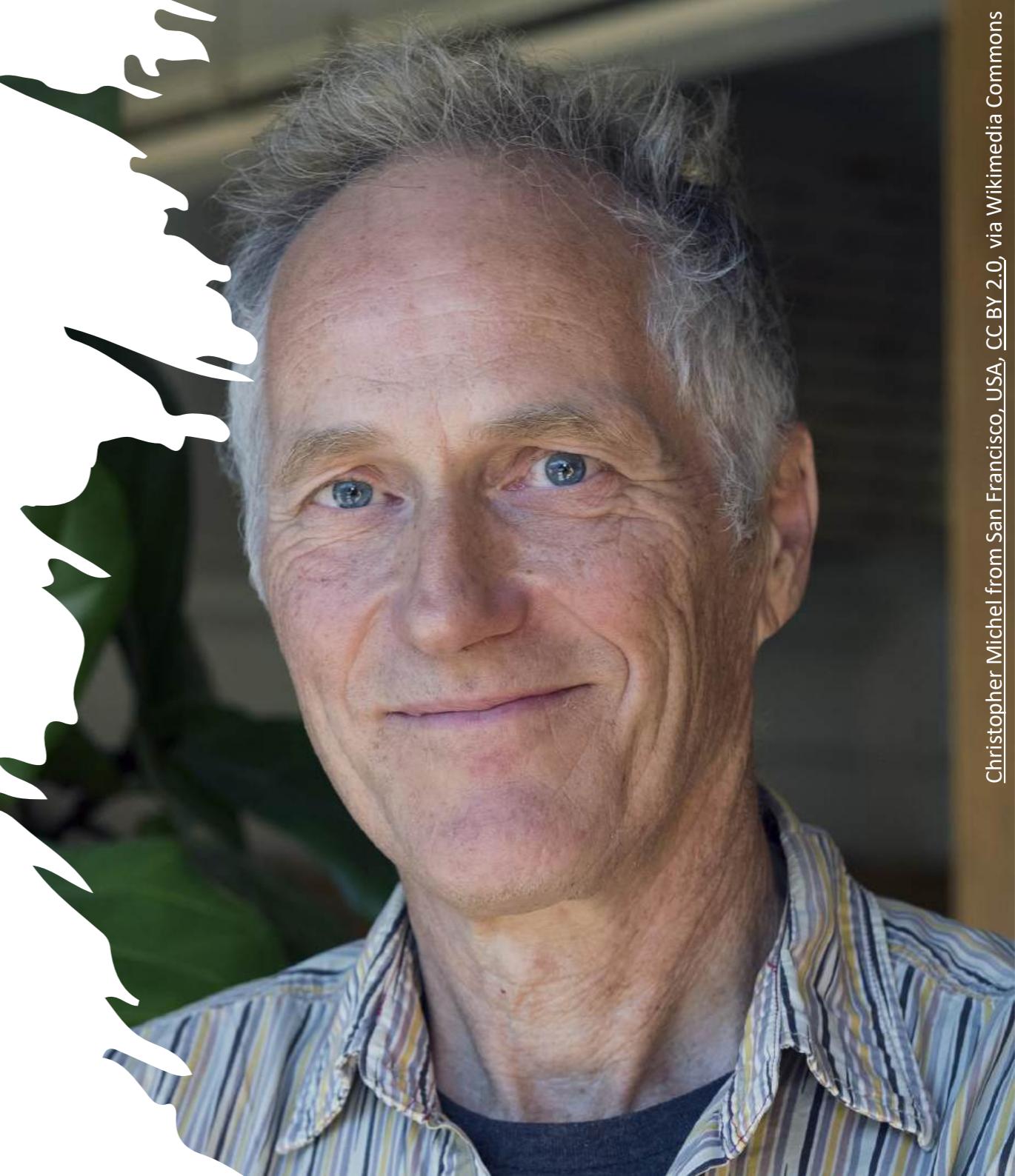
For learning and  
professional  
advancement



To give back to  
the community

“In any ecosystem,  
if you take more out  
than you put in, the  
ecosystem eventually  
depletes.”

— Tim O'Reilly



# Thank you!

[www.spinellis.gr](http://www.spinellis.gr)

X @CoolSWEng



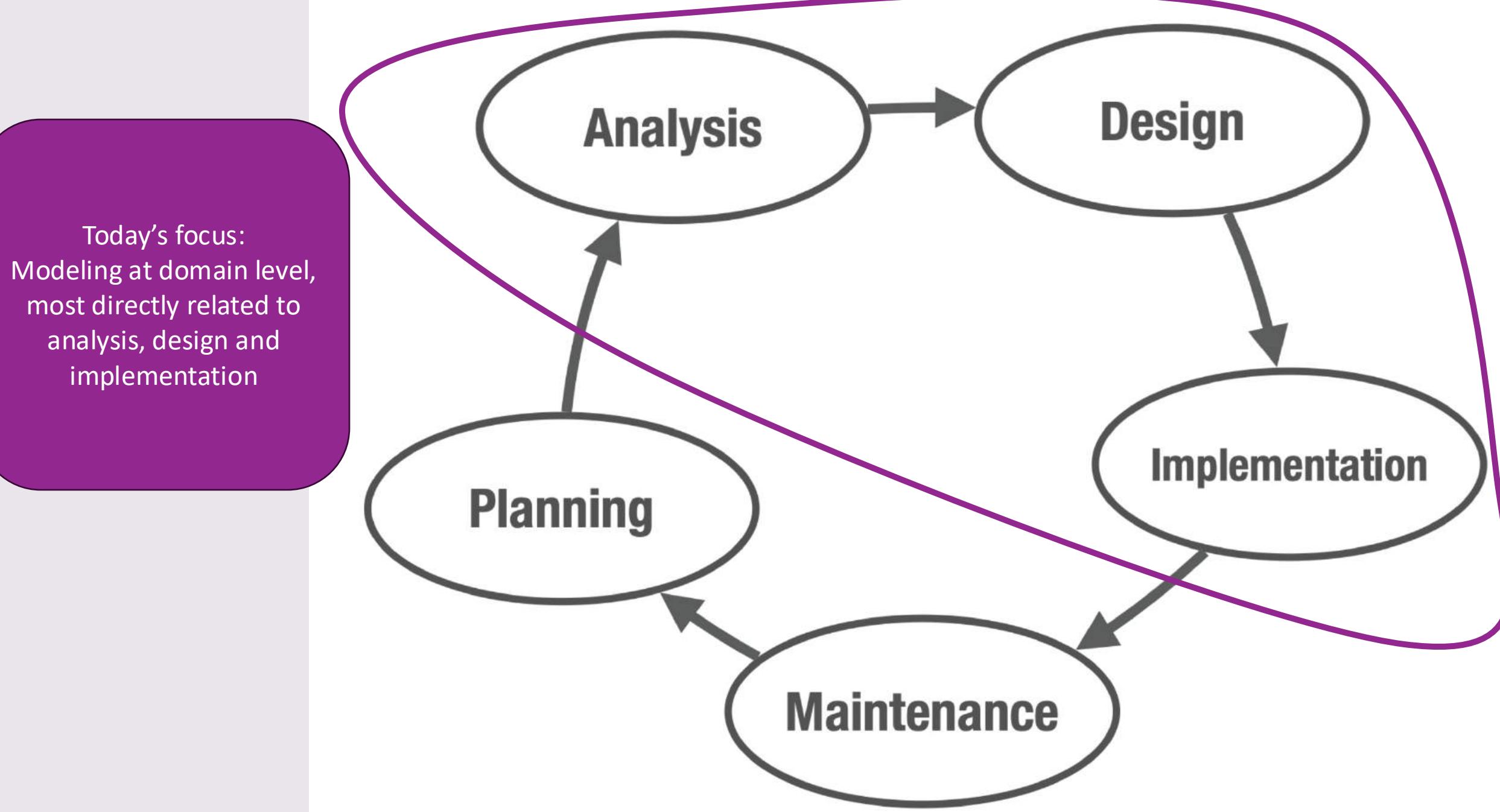
@CoolSWEng@mastodon.acm.org

# Domain-Driven Design

Arie van Deursen

Software Engineering Methods

September 2024



# Domain-Driven DESIGN

Tackling Complexity in the Heart of Software

Yet the most significant complexity of many applications is not technical. It is in the domain itself, the activity or business of the user. When this domain complexity is not handled in the design, it won't matter that the infrastructural technology is well conceived. A successful design must systematically deal with this central aspect of the software.

Eric Evans  
Foreword by Martin Fowler

# Domain-Driven DESIGN

Tackling Complexity in the Heart of Software



Eric Evans

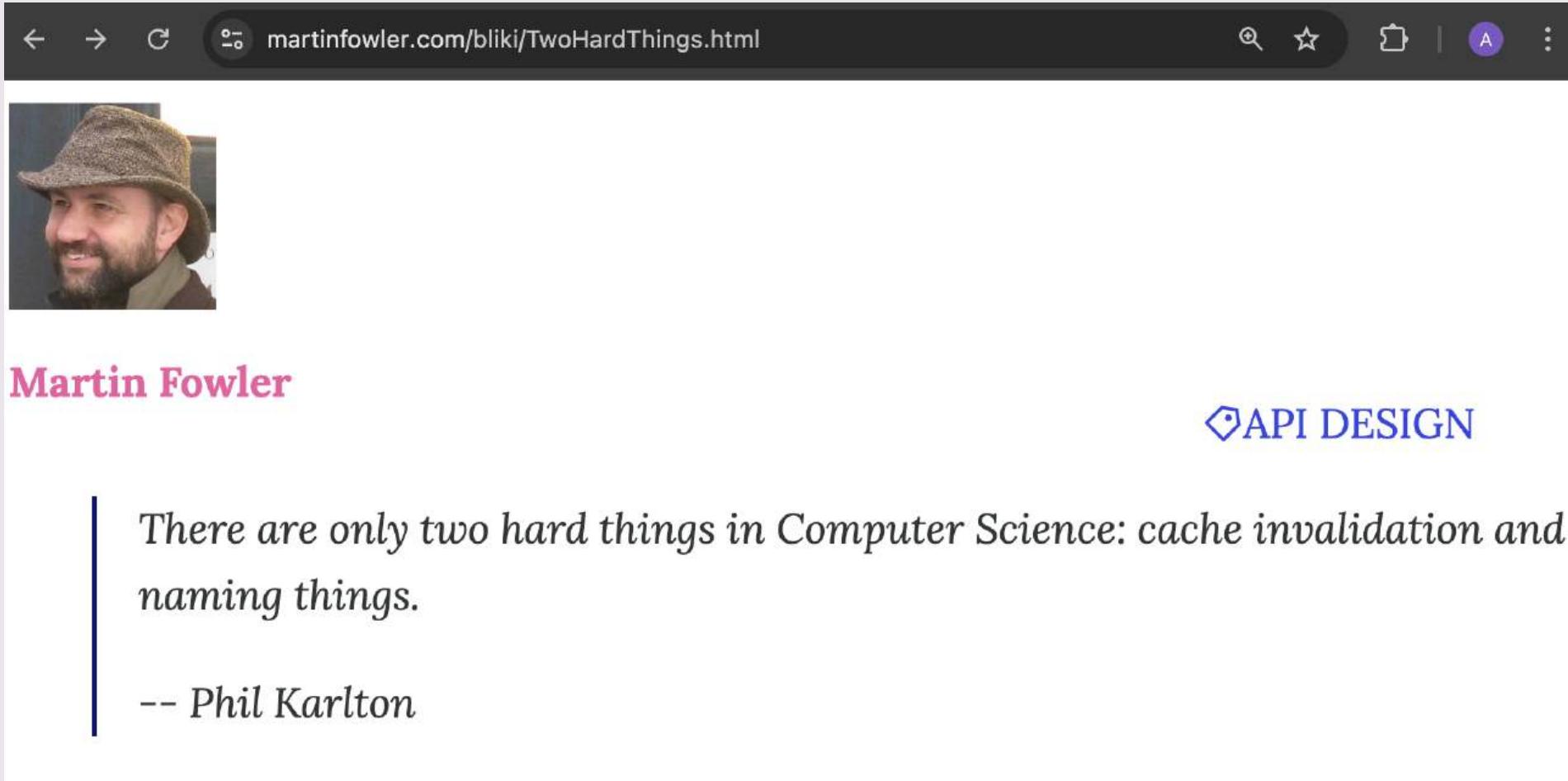
Foreword by Martin Fowler

## Explanatory Models

*"The thrust of this book  
is that one model  
should underlie  
implementation, design,  
and team communication.*

*Having separate models  
for these separate purposes poses a  
hazard."*

# Naming Things



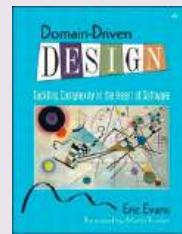
The screenshot shows a web browser window with the URL [martinfowler.com/bliki/TwoHardThings.html](http://martinfowler.com/bliki/TwoHardThings.html) in the address bar. The page content features a portrait of Martin Fowler wearing a fedora hat, followed by his name in pink text, and a quote in black text. A blue link to 'API DESIGN' is also visible.

**Martin Fowler**

 API DESIGN

*There are only two hard things in Computer Science: cache invalidation and naming things.*

-- Phil Karlton



# Ingredients of Effective Modeling

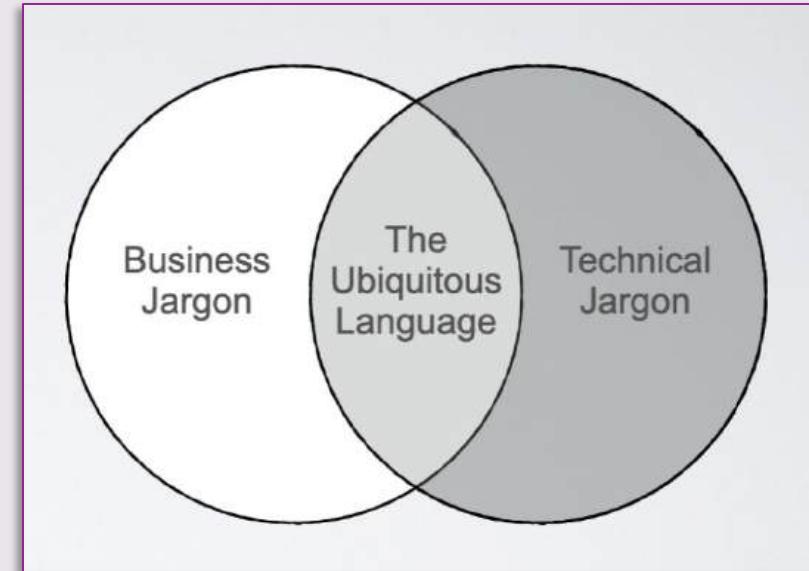
Ch.1

1. Binding the model and implementation
2. Cultivating a language based on the model
3. Developing a knowledge-rich model
4. Distilling the model (less is more)
5. Brainstorming and experimenting

It is the creativity of brainstorming and massive experimentation, leveraged through a model-based language and disciplined by the feedback loop through implementation, that makes it possible to find a knowledge-rich model and distill it. This kind of *knowledge crunching* turns the knowledge of the team into valuable models.

# The Ubiquitous Language

- Carefully crafted terminology used across the team for domain concepts: “team-ubiquitous”
- Deeply grounded in **domain knowledge**
- Supported by **models** defining key aspects of domain concepts
- Refined and improved based on constant (domain-level!) **refactoring**



# Growing the Ubiquitous Language

- Study domain literature, engage with experts
- Write down scenarios
- Model concepts and relationships in UML class diagrams  
(or “conceptual models” / Entity-Relationship Diagrams)
- Navigating the class diagram should correspond to the language used to describe scenarios
- Bind to implementation using tests
- A continuous process of “**knowledge crunching**”

# Domain-Driven DESIGN

Tackling Complexity in the Heart of Software



Eric Evans  
Foreword by Martin Fowler

## Continuous Learning

*“When we set out  
to write software,  
we never know enough.*

[...]

*This ignorance can lead us to make  
false assumptions.”*

# Our Running Example

- BookWurm / GoodReads
- Platform for book lovers to share books, reviews, reading progress, reading plans, book lists, ...
- GoodReads: Owned by Amazon
- BookWurm: Federated, based on ActivityPub (like Mastodon)

https://bookwurm.social

30%

Search for a book

Username  password  Forgot your password?  Log in

**BookWurm**  
Social Reading and Reviewing

DECENTRALIZED   
FRIENDLY   
ANTI-CORPORATE

**Join BookWurm**

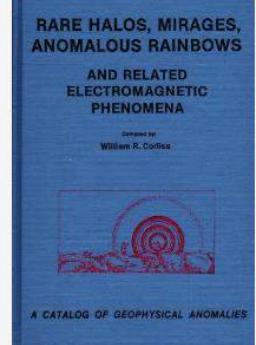
Username:   
Choose wisely! Your username cannot be changed.

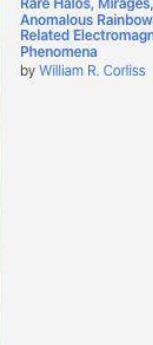
Email address:

Password:

Sign Up

  
Talk about books, track your reading, and find out what your friends are reading, on your own terms. BookWurm is ad-free, anti-corporate, and federated. You can seamlessly follow and interact with users on other BookWurm instances, and on services like Mastodon.

  
Rare Halos, Mirages, Anomalous Rainbows, and Related Electromagnetic Phenomena  
by William R. Corliss  
A CATALOG OF GEOPHYSICAL ANOMALIES  
No rating

  
ROUTINIEN DES VERGESSENS  
ROUTINIEN DES VERGESSENS  
by Raphaëla Edelbauer  
★★★★☆  
No rating

  
Le Chien du Forgeron  
by Camille Leboulanger  
Approchez, approchez ! Alors que tombe la nuit froide, laissez-moi vous divertir avec l'histoire de Cuchulainn, celui que l'on nomme le Chien du Forgeron ; celui qui s'est rendu dans l'Autre Monde plus de fois qu'on ne peut le compter sur les doigts d'une main, celui qui a repoussé à ...

  
★★★★☆  
監書密令  
by Rachel Caine

  
禁書陷阱  
by Rachel Caine  
★★★★☆

  
LE CHIEN DU FORGERON  
by Camille Leboulanger  
★★★★☆

bookwyrm.social/list/2266/s/software-architect

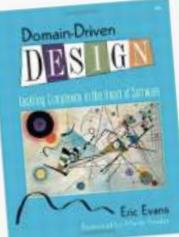
# Software Architecture

Created and curated by Arie van Deursen

Lists / Software Architecture

List of books on the general topic of software architecture. Suggestions for additions welcome.

1

 Domain-Driven Design by Eric Evans

★★★★★ ★  
"Eric Evans has written a book that matches your software match your domain." - Arie van Deursen

Read ✓

Add notes

Added by Arie van Deursen

List position

2

 Building Microservices by Martin Newman

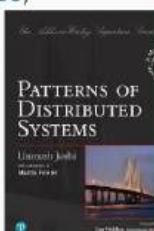
★★★★★ ★  
Read ✓

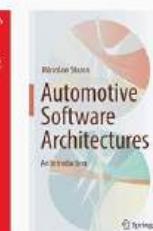
Add notes

Activity    Reviews and Comments    Reading Goal    Groups    Lists    Books

To Read (View all 39)

 Designing Data-Intensive Applications by Martin Kleppmann

 Patterns of Distributed Systems by Werner Kuehl

 Automotive Software Architectures by Bernd Hartmann

Currently Reading

 Patterns, Principles, and Practices of Domain-Driven Design by Scott Addie with Scott W. Ladd

 Implementing Domain-Driven Design by Vaughn Vernon

 Lost in Math by Sabine Hossenfelder

Read (View all 39)

Arie van Deursen reviewed Domain-Driven Design by Eric Evans

Nov 30 · edited Aug 10 ·

### Wisdom on Every Page

★★★★★ ★

The main reason I love this book is because it is so excellently written. It makes you want to believe that rigorously focusing on the "domain" and its "ubiquitous language" will solve all your software design problems. The book is divided into four parts.

- Part I introduces the overall concepts of Domain-Driven Design, including the development of the "ubiquitous language".
- Part II focuses on model design within an isolated context. It distinguishes entities, value objects, and modules, as well as aggregates, factories, and repositories for managing domain objects.
- Part III emphasizes the incremental process of building a proper domain model, through a series of refactoring steps.
- Part IV steps out of the bounded context, and shows where interaction needs to take place between bounded contexts, calling for, e.g., anti-corruption layers and common kernels.

# Eric Evans

## Author details

Born: Jan. 24, 1962

## External links

[View on OpenLibrary](#) [Load data](#)

### Books by Eric Evans

 Domain-Driven DESIGN  
 Domain-Driven DESIGN

clayton finished reading G-Man: J. Edgar Hoover and the American Century by Beverly Gage 17 hours ago

G-Man: J. Edgar Hoover and the Making of the American Century by Beverly Gage Hoover's life and career, from his birth in 1895 to a Washington civil-service family through his death in 1972. Gage ...

Want to read

mouse started reading The girl in the tower by Katherine Arden 19 hours ago

The Girl in the Tower by Katherine Arden Want to read

mouse finished reading Artificial Condition by Martha Wells (The Murderbot Diaries, #2) 19 hours ago

Artificial Condition by Martha Wells (The Murderbot Diaries, #2) It has a dark past - one in which a number of humans were killed. A past that caused it...

# Scenarios / Use Cases (I)

## Posting about books

- Compose reviews, with or without ratings, which are aggregated in the book page
- Compose other kinds of statuses about books, such as:
  - Comments on a book
  - Quotes or excerpts
- Reply to statuses
- View aggregate reviews of a book across connected BookWyrm instances
- Differentiate local and federated reviews and rating in your activity feed

# Scenarios / Use Case (II)

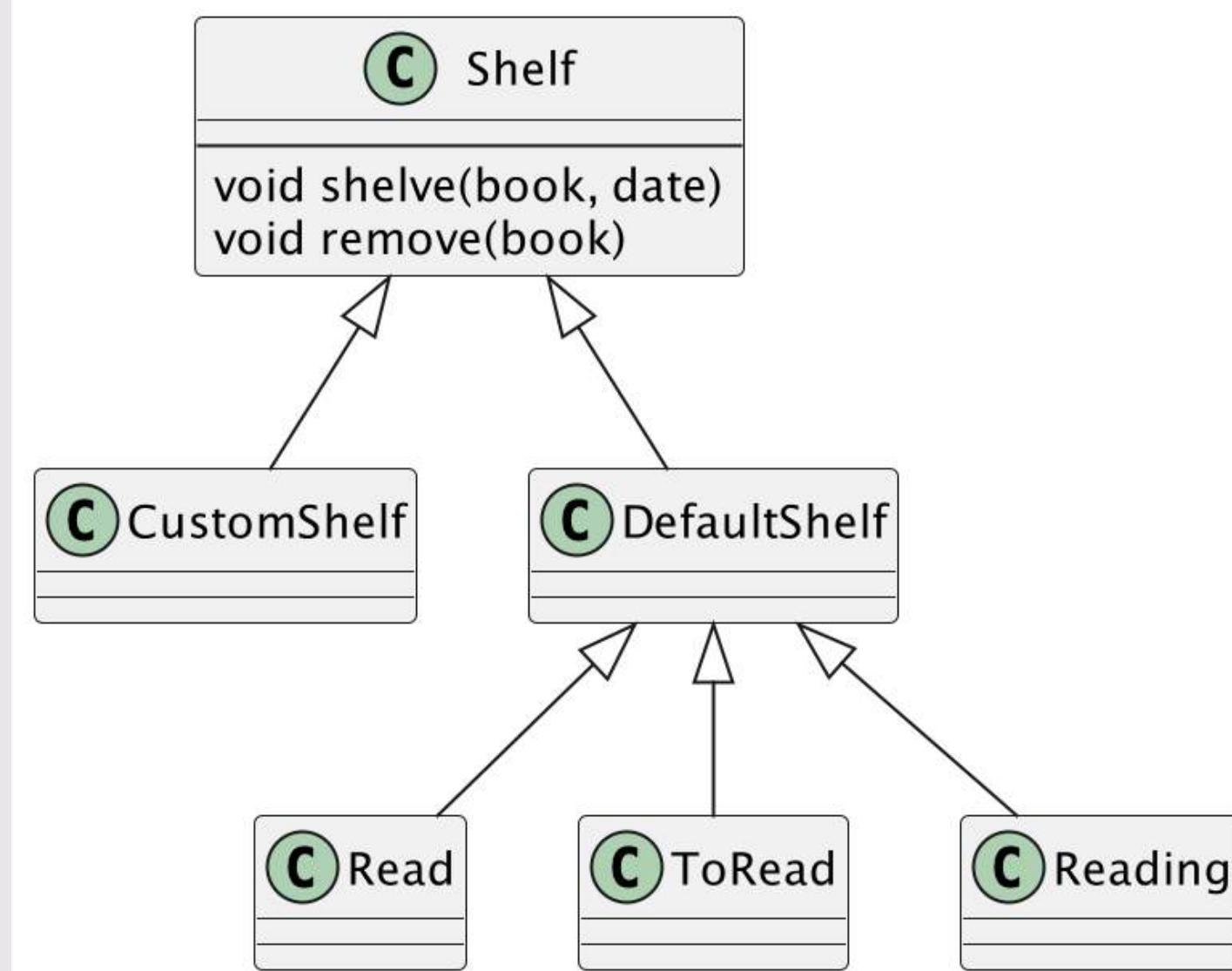
Can you draw relevant UML diagrams?

## Track reading activity

- Shelve books on default "to-read," "currently reading," and "read" shelves
- Create custom shelves
- Store started reading/finished reading dates, as well as progress updates along the way
- Update followers about reading activity (optionally, and with granular privacy controls)
- Create lists of books which can be open to submissions from anyone, curated, or only edited by the creator

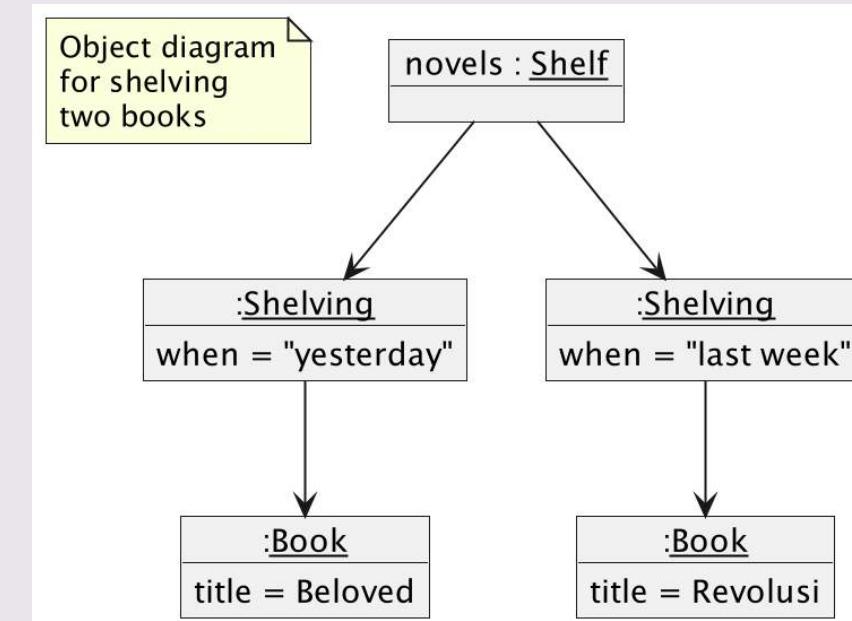
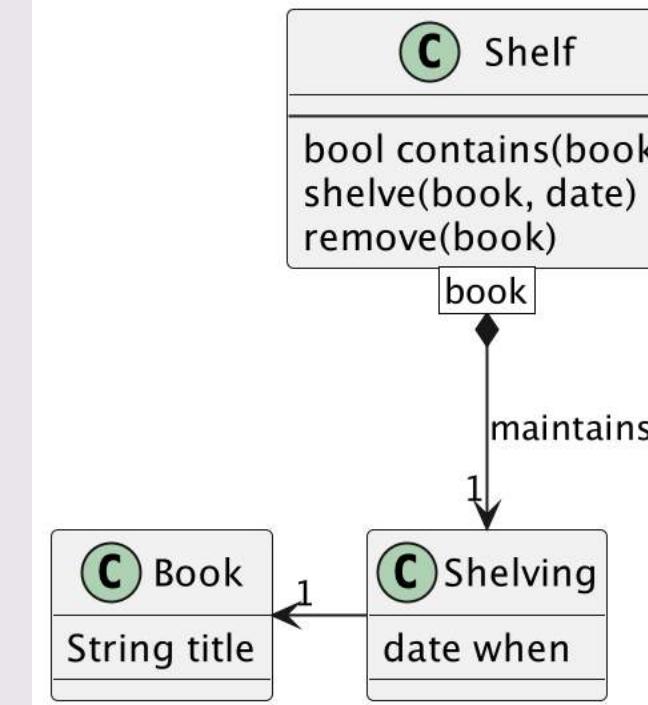
# Sorts of Shelves

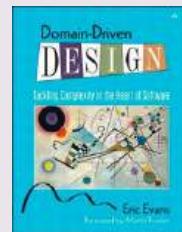
- Custom shelves are user created and are like tags
- Default shelves have special meaning
- Keep track of books a user has read, wants to read, or is currently reading.



# Shelving Books

- We remember the date when a book was put on the shelf
- A shelf maintains “shelvings”, one per book  
[ UML qualified association ]
- Each shelving has a date and a book.





# Binding to Implementation

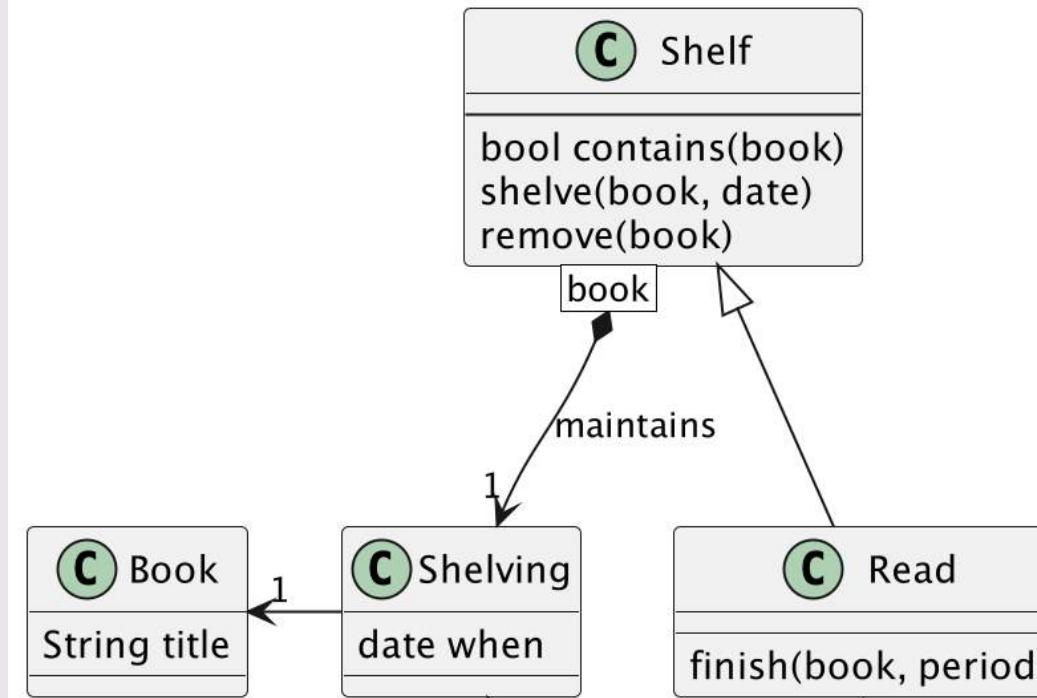
Ch. 1

“After a couple more part-time days of this, I felt I understood enough to attempt some code. I wrote a very simple prototype, driven by an automated test framework.”

```
@Test  
public void testAddBookToShelf() {  
    var novels = new Shelf();  
    var beloved = new Book();  
    novels.shelve(beloved, LocalDateTime.now());  
    assertThat(novels.contains(beloved)).isTrue();  
}
```

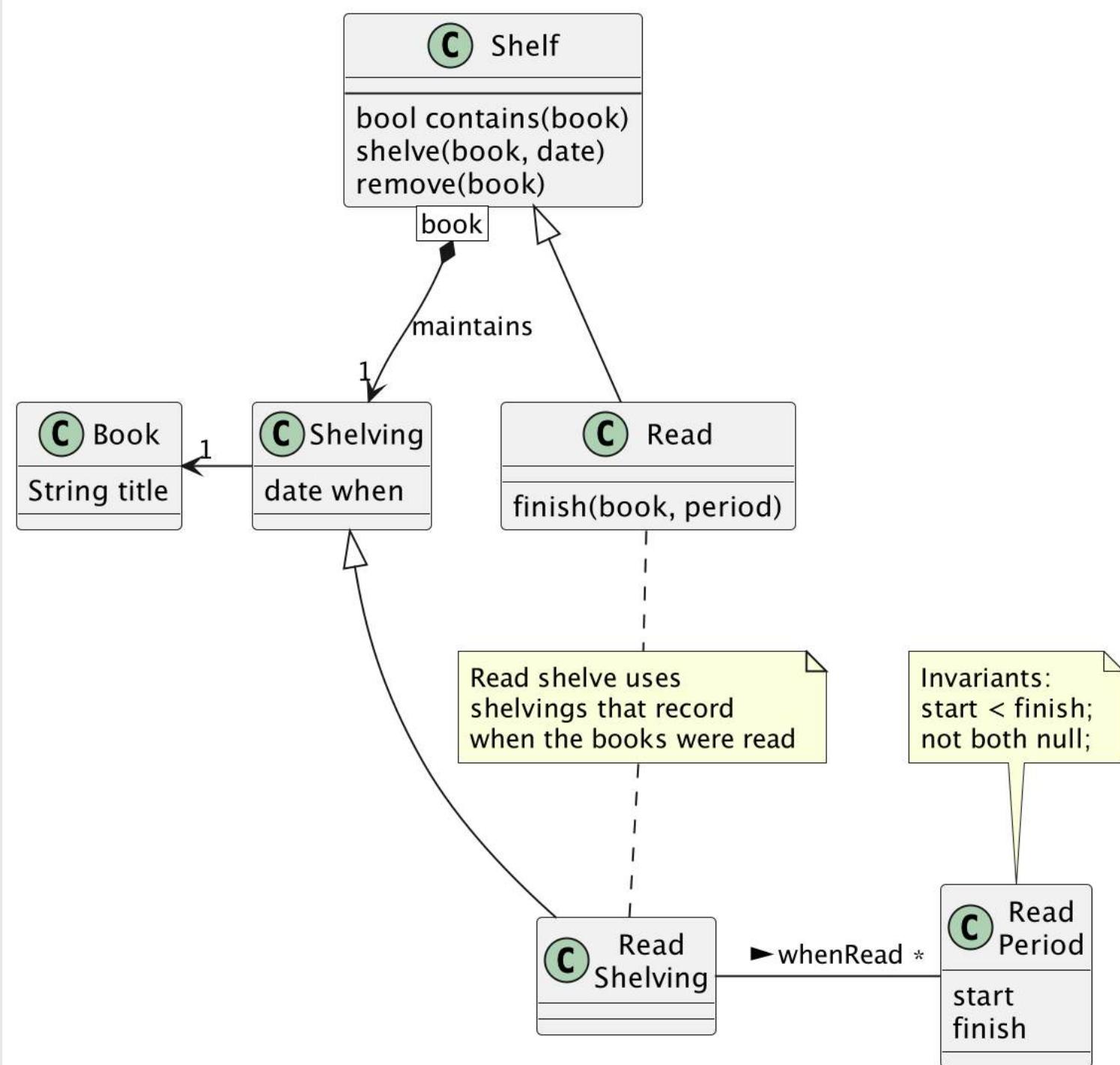
# Reading Periods

- The special shelf ‘Read’ keeps track of start and finish dates



# Reading Periods

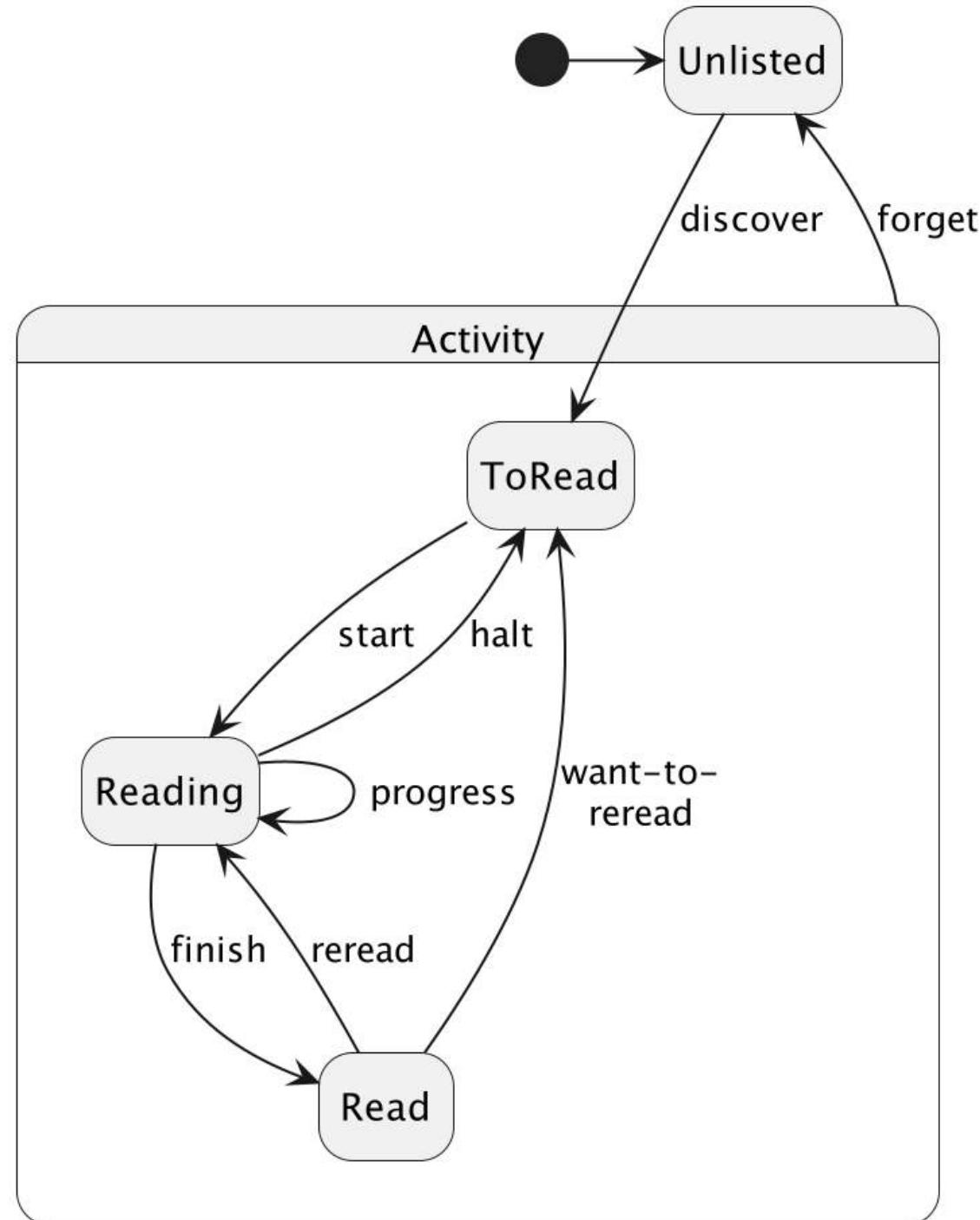
- The special shelf ‘Read’ keeps track of start and finish dates
- It uses special ‘Shelvings’ to keep track of periods when book was read
- A book can be read multiple times



# Reading Activity

- A language for tracking reading activity
- The happy flow is to discover a book, start reading it, making progress, and finishing it.
- Great books are re-read
- A book can always be forgotten

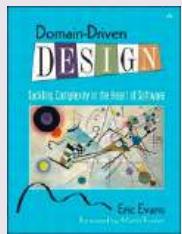
[ UML state diagram ]



# Tests Shape Language

Tests follow and influence the ubiquitous language

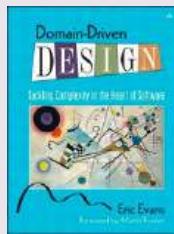
```
@Test  
public void testFinishBook() {  
    var read = new ReadShelf();  
    var beloved = new Book();  
    read.shelve(beloved, LocalDateTime.now());  
    assertThat(read.contains(beloved)).isTrue();  
  
    var start = LocalDateTime.now();  
    var finish = LocalDateTime.now();  
    var p1 = new ReadingPeriod(start, finish);  
    read.finish(beloved, p1);  
  
    assertThat(read.whenRead(beloved)).isNotEmpty();  
    var p2 = read.whenRead(beloved).getFirst();  
    assertThat(p2.getFinish()).isEqualTo(finish);  
}
```



# Language Creation

Ch. 2

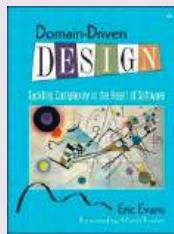
- Use the model as the backbone of a language.
- Commit the team to exercising that language relentlessly
- Use the same language in diagrams, writing, and especially speech.
- Iron out difficulties by experimenting with alternative expressions, which reflect alternative models.
- Then refactor the code, renaming classes, methods, and modules to conform to the new model.



# Documenting? Diagrams?

Ch. 2

- “The trouble comes when people feel compelled to convey the whole model or design through UML.”
- “Diagrams are a means of communication and explanation, and they facilitate brainstorming. They serve these ends best if they are minimal.”
- “The vital detail about the design is captured in the code.”
- *Documents Should Complement Code and Speech*
- *Documents Should Work for a Living and Stay Current*



# Hands-On Modelers

Ch. 3

**Any technical person contributing to the model must spend some time touching the code, whatever primary role he or she plays on the project. Anyone responsible for changing code must learn to express a model through the code. Every developer must be involved in some level of discussion about the model and have contact with domain experts. Those who contribute in different ways must consciously engage those who touch the code in a dynamic exchange of model ideas through the UBIQUITOUS LANGUAGE.**

# [ UML in this Course ]

- You need to master UML
- PlantUML and “UML @ Classroom” provide rich information
- Make sure you can use it correctly and meaningfully
- In your professional life UML will be a key tool for clearly communicating design ideas
- Storing / maintaining a huge collection of UML diagrams in a project is likely counter productive

# Domain Isolation: Layered Architecture

**UI:** interaction with user

**Application:** jobs the software  
is supposed to do

**Domain:** business concepts  
and rules

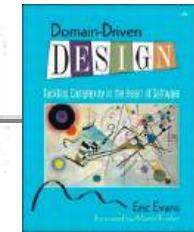
**Infra:** messaging, persistence,  
...

**User  
Interface**

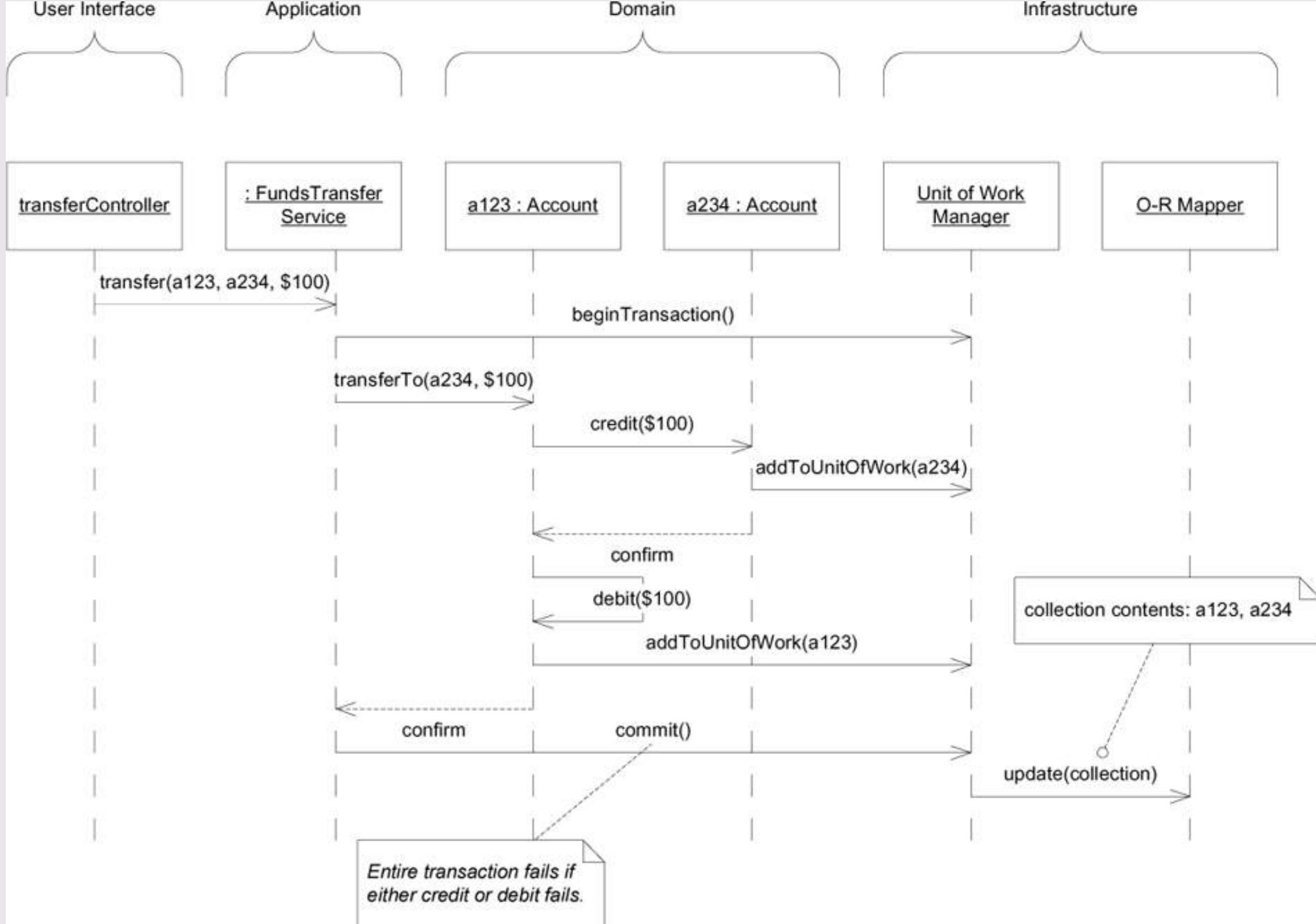
**Application**

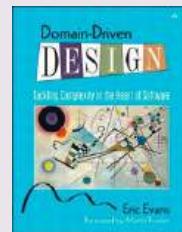
**Domain**

**Infrastructure**



Ch. 4

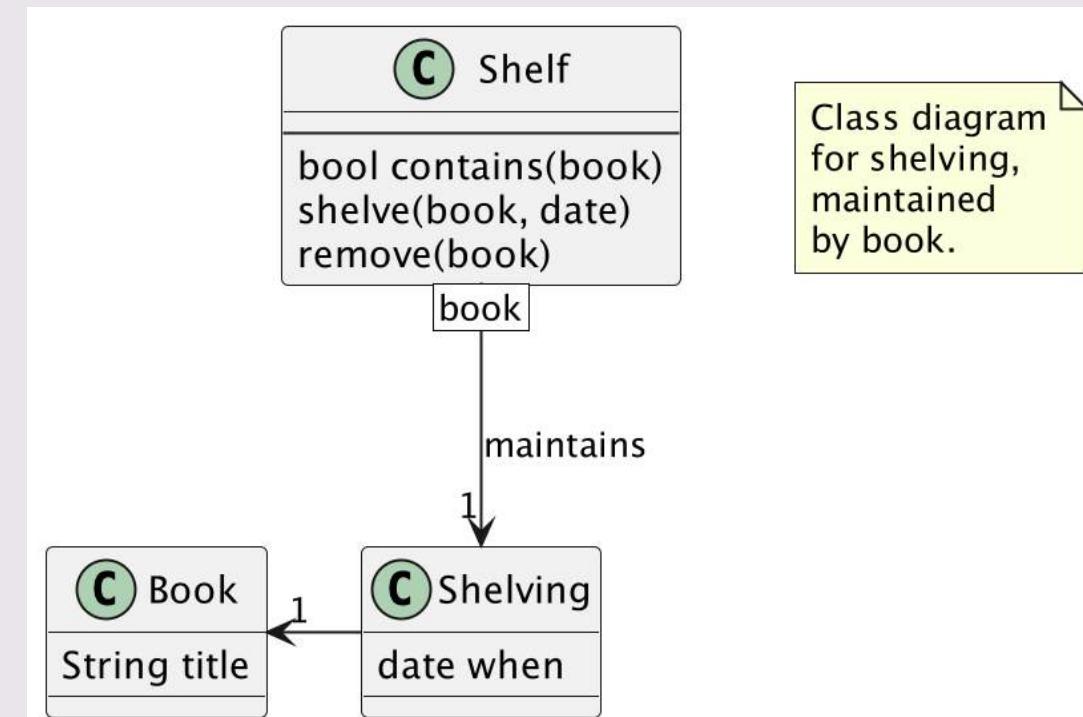


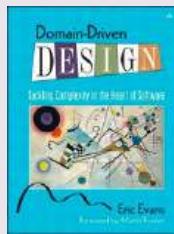


Ch. 5

# Rethinking (Many-to-Many) Associations

- In real life, associations are often many-to-many
- Their correct implementation and maintenance can easily become a nightmare
- Therefore:
  - Impose a traversal direction
  - Add a qualifier to reduce multiplicity
  - Eliminate non-essential associations





# Entities vs Value Objects

Ch. 5

## Entities

- Many objects have a “life cycle”
  - a thread of events happening to them
- These objects *remain the same as they change*:  
they have **identity**

## Value Objects

- Objects that cannot change
- If any of their fields would change, you’d have a new object
- Date, name, age, ...
- “What”, but not “who” or “which”

Some concepts can have identity in one context, and no identity in another.  
Ultimately, it is a *design decision* whether you treat a class as an *entity* or *value object*.

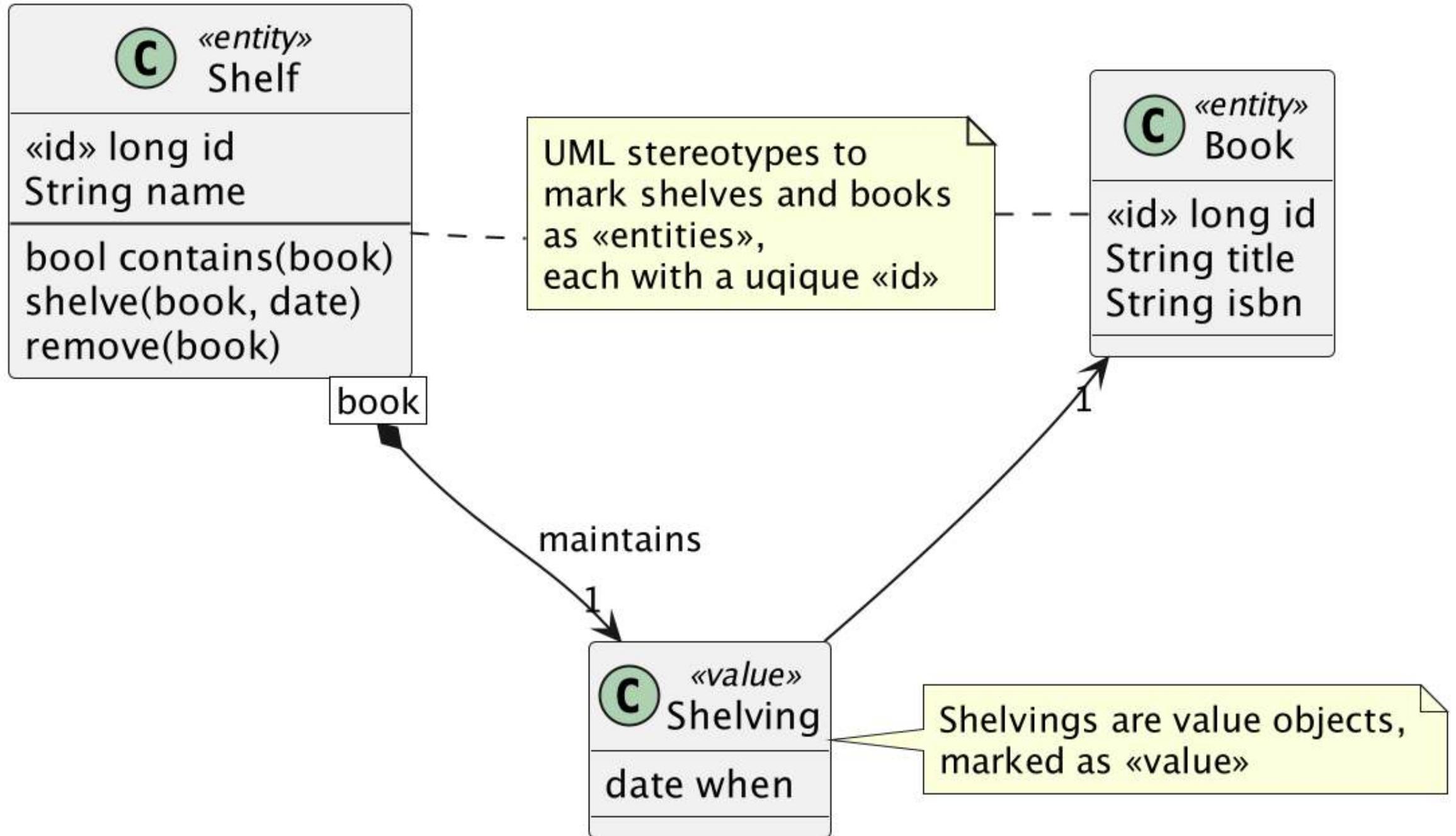
# BookWurm Entities and Value Objects

## Entities

- User
- Book
- Shelf
- Publisher
- Author

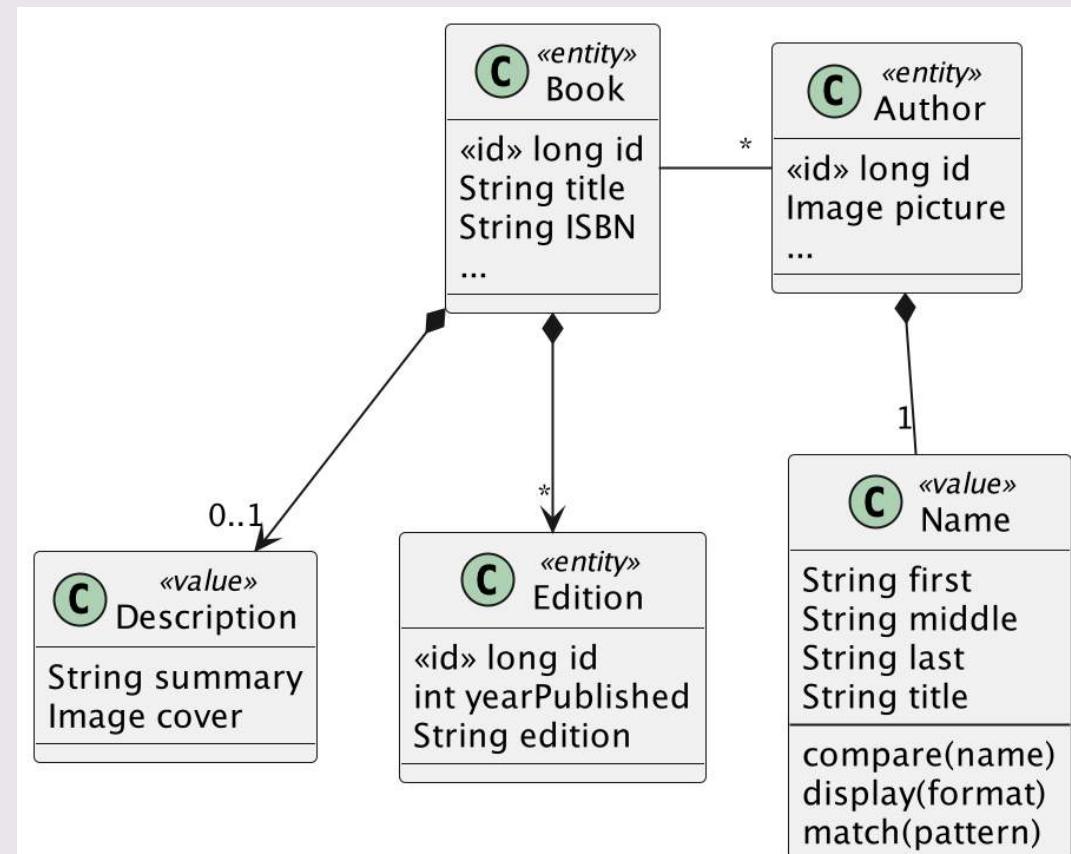
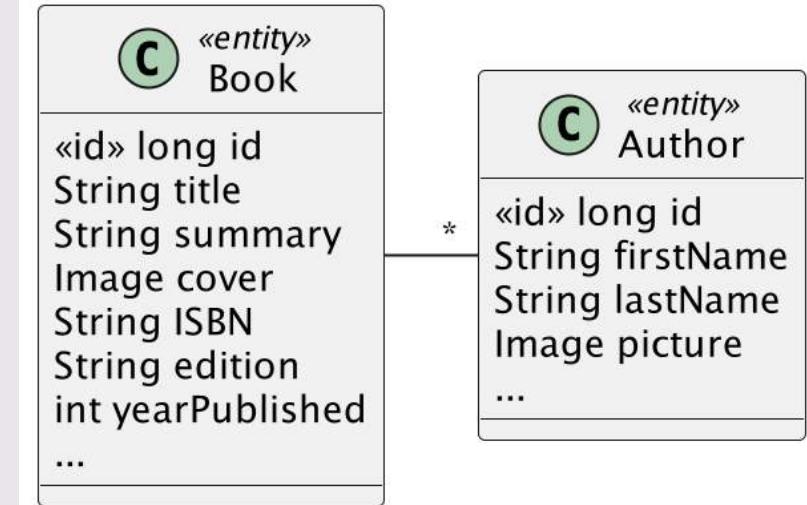
## Value Objects

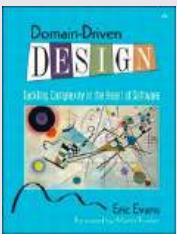
- Reading period
- Reading progress
- Book rating
- Shelving
- (Author) names



# Entity Design

- Keep your entities simple
- Focus on behavior
- Expose information need to find it
  - ISBN, authors, title, year, publisher, ...
- Delegate behavior / attributes to other (associated) classes
  - May be entities or value objects





# Services

Ch. 5

- Some domain operations don't fit an object
- It's OK to create a dedicated “service” class
  - Mostly behavior, little (no!) local “state” / attributes
- “A SERVICE is:
  - an operation
  - offered as an interface
  - that stands alone in the model,
  - without encapsulating state”

# Service Characteristics

A good SERVICE has three characteristics.

1. The operation relates to a domain concept that is not a natural part of an ENTITY or VALUE OBJECT.
2. The interface is defined in terms of other elements of the domain model.
3. The operation is stateless  
(clients can use service without regard for service history)

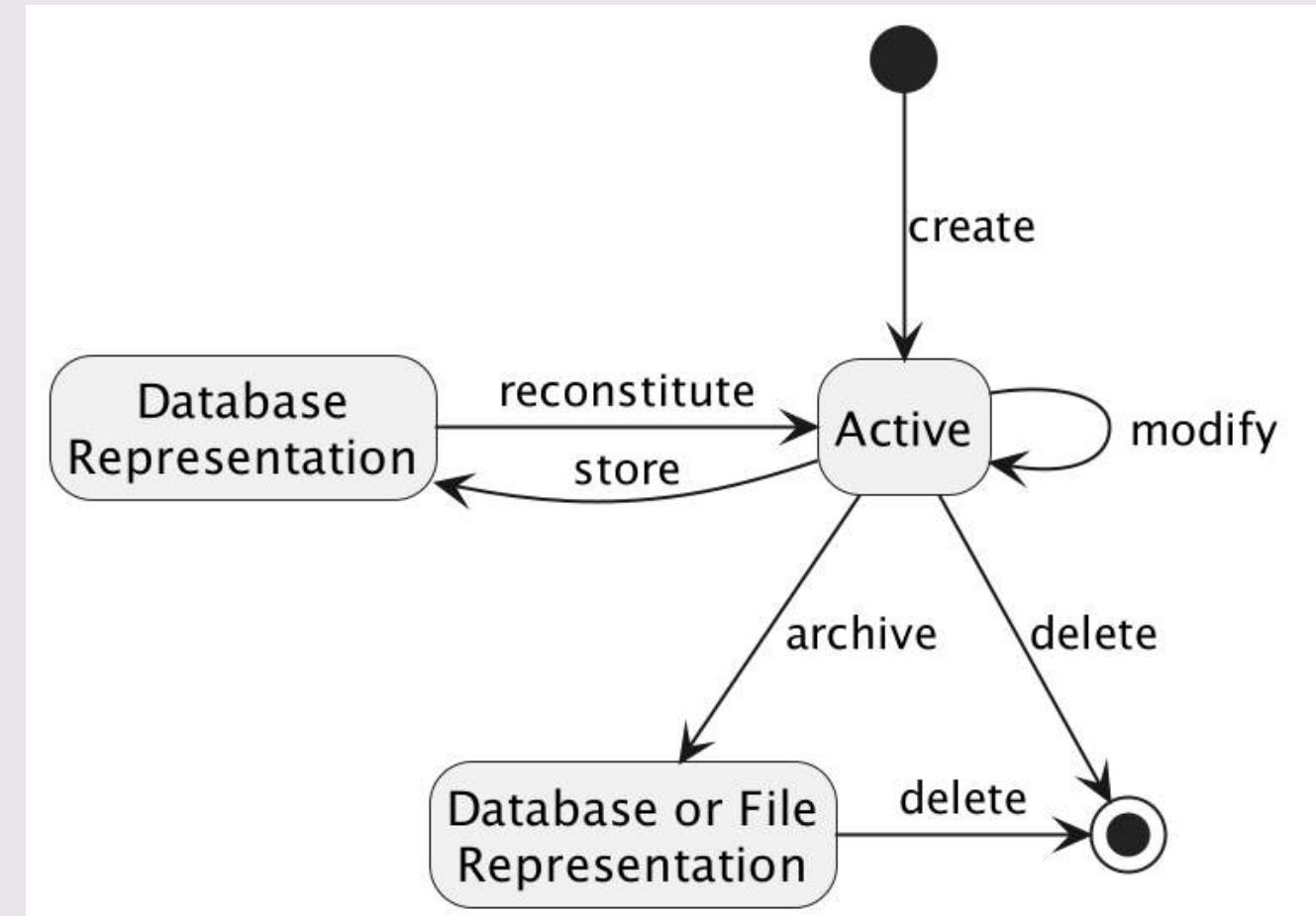
# BookWurm Services?

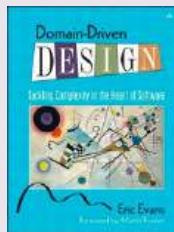
- Import / export
  - From / to GoodReads
  - From / to Excel
- (Periodic) indexing for search
- Finding duplicate books
- (Periodic) synchronization with external libraries

# A Domain Object's Lifecycle

Ch. 6

- Objects have a life
- Relevant objects will need to be stored (and reconstituted)
- Poorly managing object life cycles can kill a system (performance, integrity, maintenance)



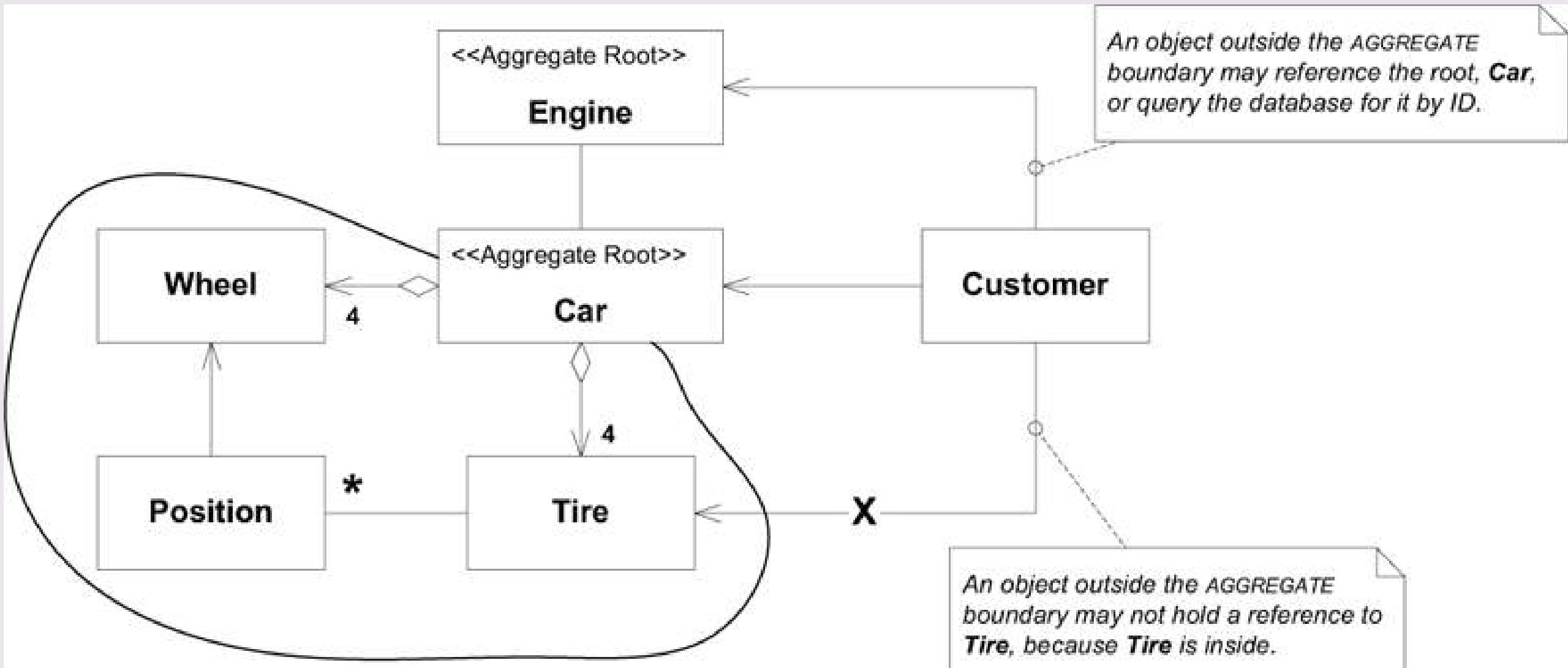


# Aggregates

Ch. 6

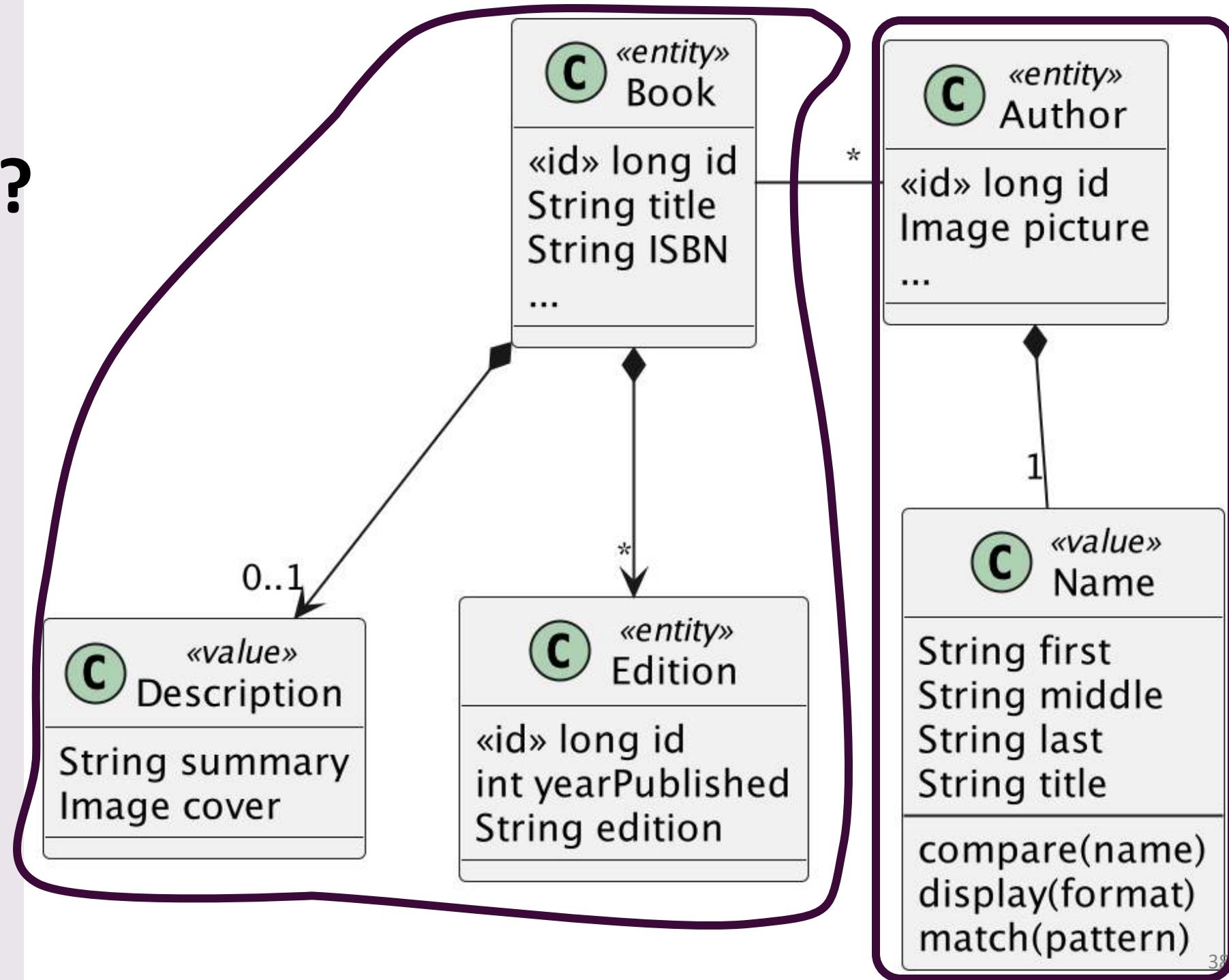
- *A cluster of associated objects treated as a unit for the purpose of data changes.*
- The aggregate **boundary** defines what is inside it.
- The aggregate **root** is a single, specific entity contained in it.
  - The root is the only member of the aggregate that outside objects are allowed to hold references to
  - Objects within the boundary may hold references to each other.
  - Entities other than the root have local identity
  - No outside object can ever see it out of the context of the root entity.

# Aggregates



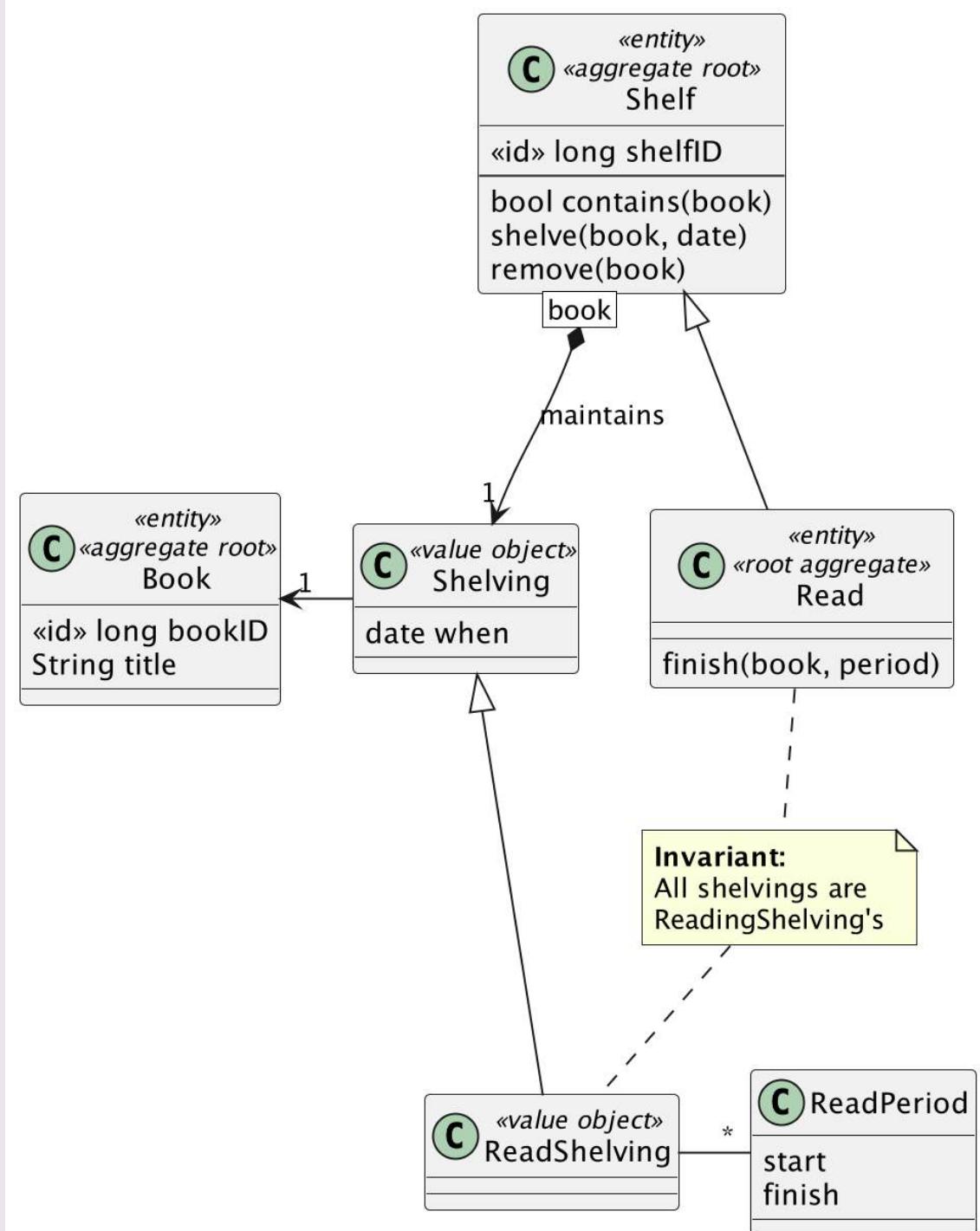
# BookWurm Aggregates?

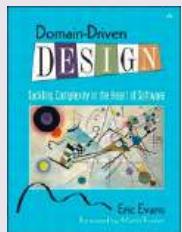
- Several options
- Book (root)
  - + Editions
  - + Description
- Author (root)
  - + Name



# Shelf Aggregates

- Shelves are aggregates
  - With their shelvings
- Shelf subclasses are aggregates too!
- Read shelf safeguards **invariant** on shelvings.
- Books are separate aggregates





# Factories

Ch. 6

- Some objects will be hard to create
  - May require substantial implementation knowledge
- Aggregates need to be created as a whole
  - Invariants must be ensured
- *Factories* offer services for the creation of complex objects
  - [ not necessary for *all* objects ]
- Factories may also be able to reconstitute stored objects
- [ See also GoF creational design patterns ]

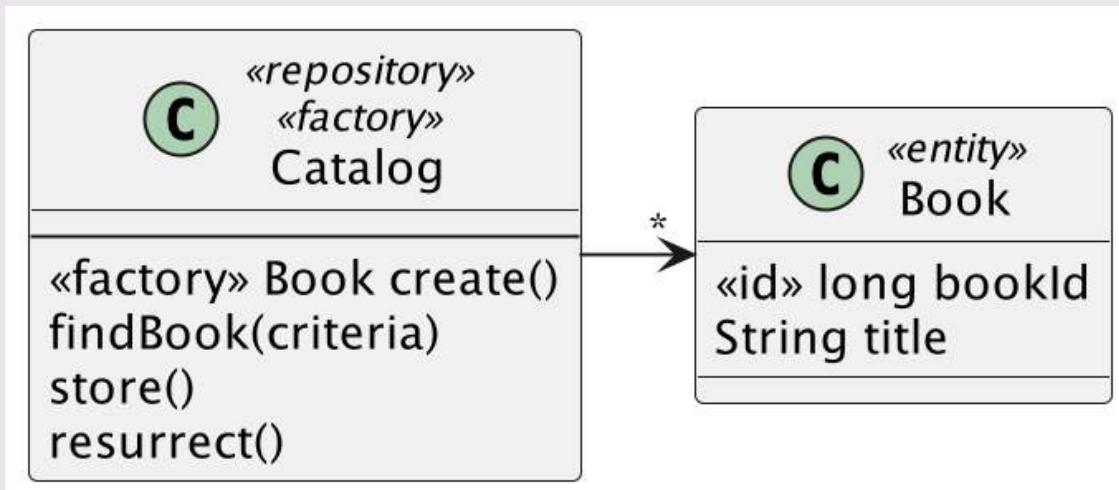
# Repositories

A repository:

- Represents all objects of a certain type as a conceptual set
- Offers more elaborate querying capability.
- Can add / remove objects. The machinery behind the repository inserts or deletes them from the database.
- Offers cohesive set of responsibilities for providing access to the roots of aggregates from early life cycle through the end.

# BookWurm Repositories?

- Catalog of books
- List of users
- Authors
- Shelves

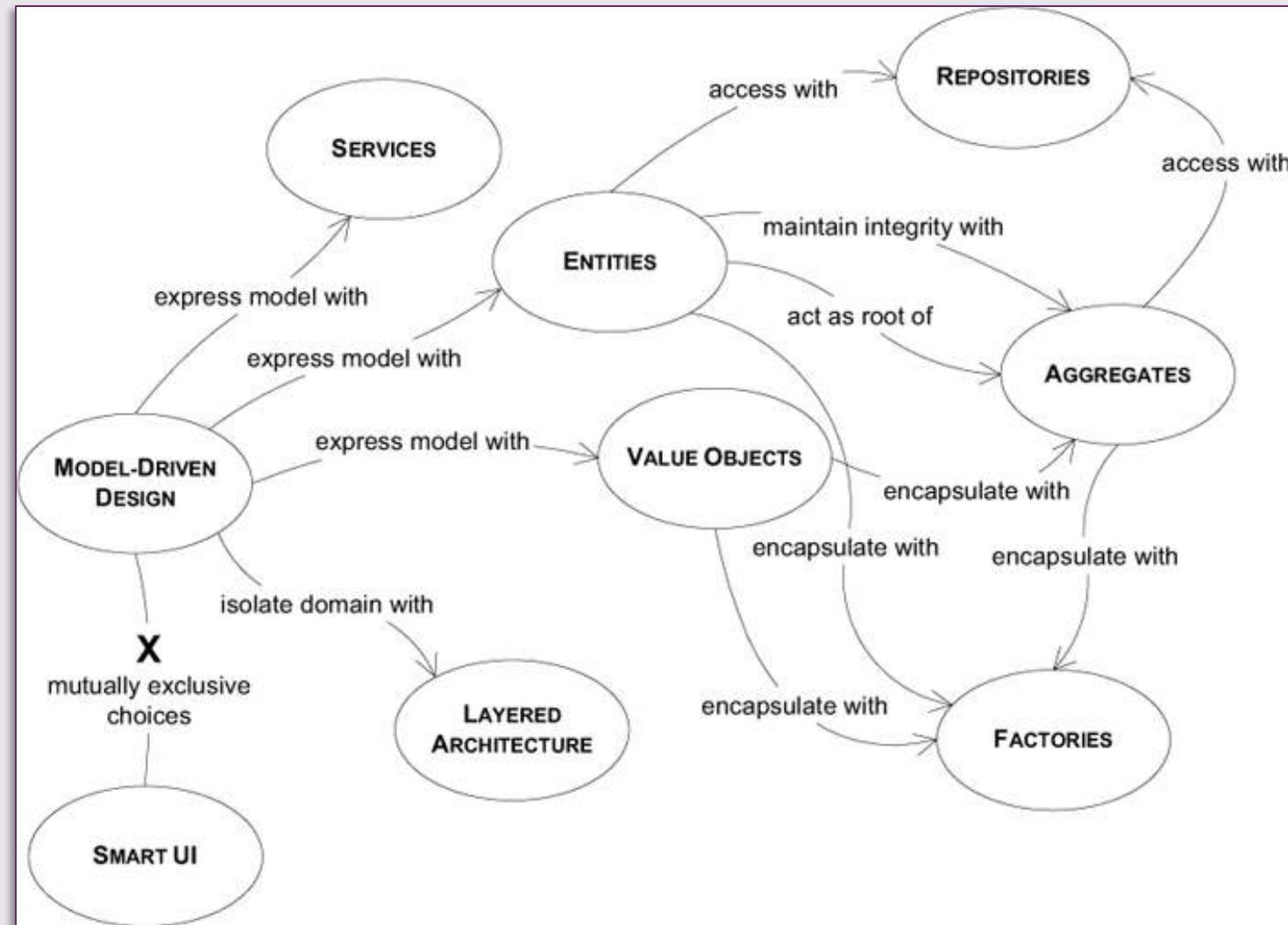


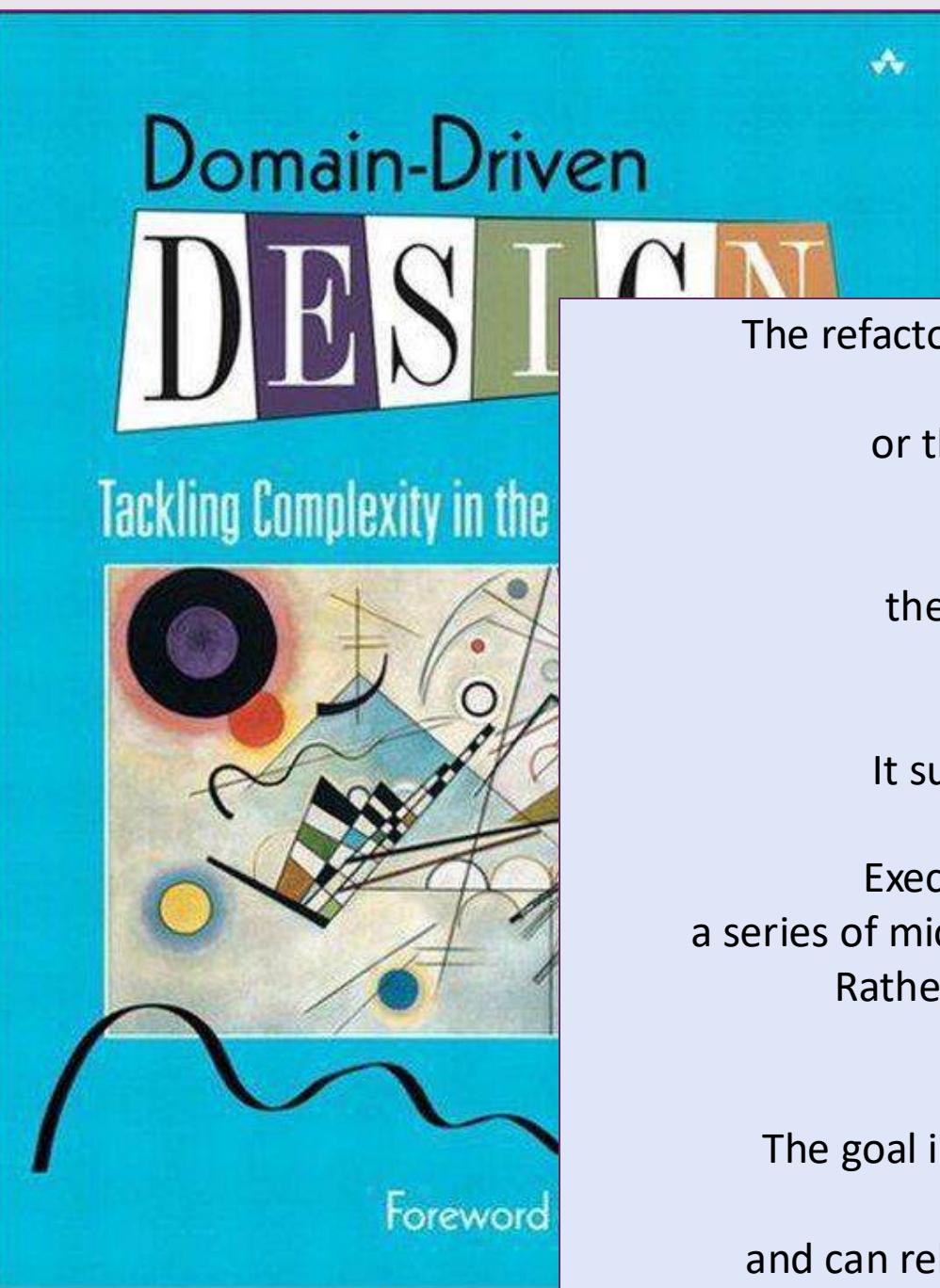
```
@BeforeEach
public void setUp() {
    aCatalog = new Catalog();
    dunkelblum = aCatalog.createBook();
    dunkelblum.setTitle("Dunkelblum");
    barbarossa = aCatalog.createBook();
    barbarossa.setTitle("Barbarossa");
}

@Test
public void testBooksAreUnique() {
    assertThat(dunkelblum.getId())
        .isNotEqualTo(barbarossa.getId());
}
```

# DDD Tactical Patterns

Ch. 5





# Domain-Driven DESIGN

## Model Refactoring

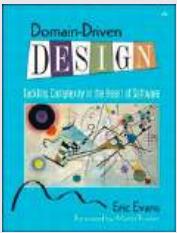
The refactorings that have the greatest impact on the viability of the system are those motivated by new insights into the domain or those that clarify the model's expression through the code.

This type of refactoring does not in any way replace the refactorings to design patterns or the micro-refactorings, which should proceed continuously.

It superimposes another level: refactoring to a deeper model.

Executing a refactoring based on domain insight often involves a series of micro-refactorings, but the motivation is not just the state of the code. Rather, the micro-refactorings provide convenient units of change toward a more insightful model.

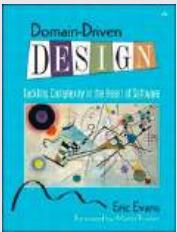
The goal is that not only can a developer understand what the code does; they can also understand *why* it does what it does and can relate that to the ongoing communication with the domain experts



# One Domain to Rule Them All!

Ch. 14

- No!!
- A meaningful domain must be well-scoped
- A domain model lives within a **bounded context**
- A system may require a multitude of bounded contexts
- Some concepts may re-occur in different contexts:
  - Each bounded context can use its own tailor-made variant
  - Such concepts will need some translation when exchanged



Ch. 14

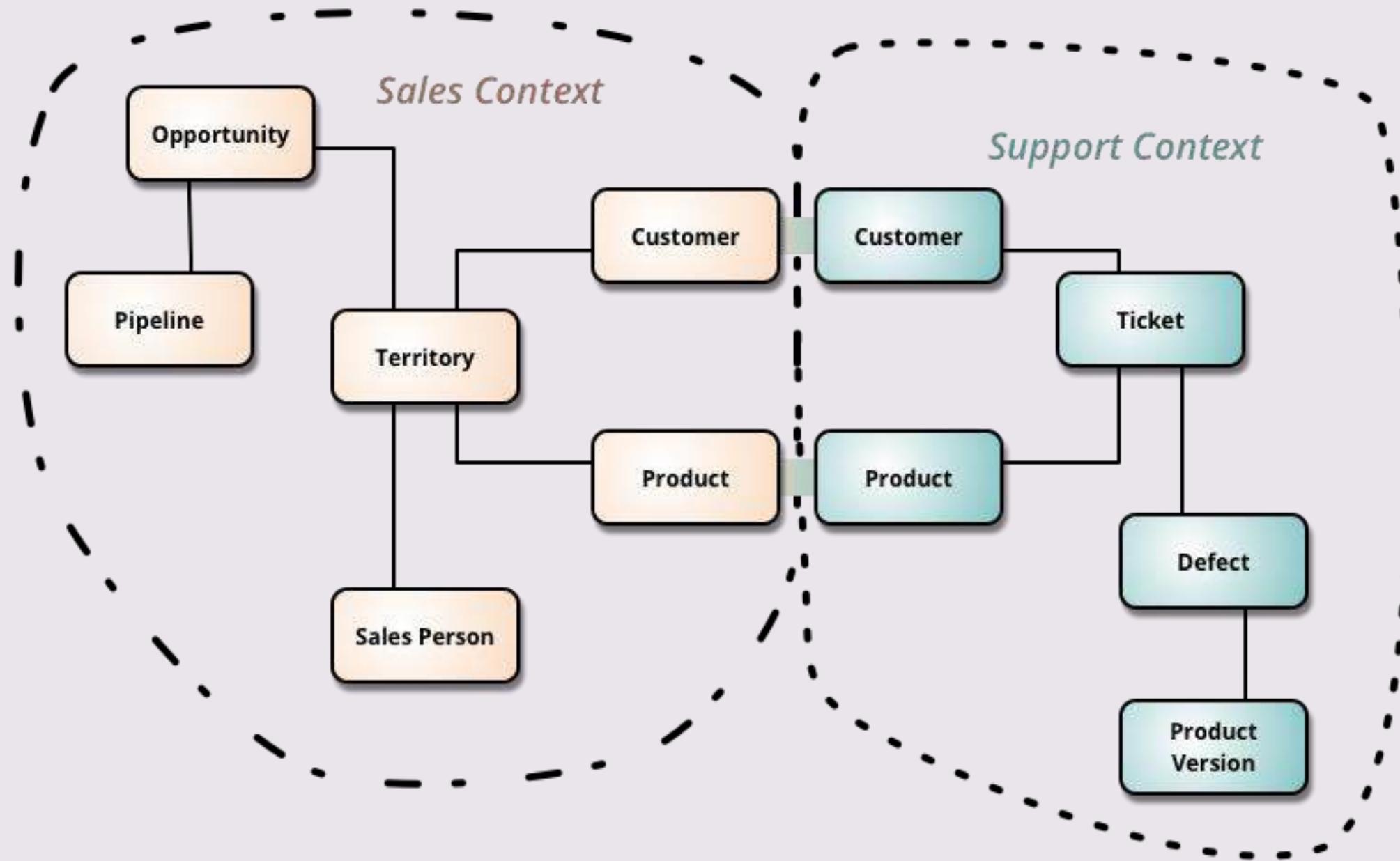
# The Bounded Context

A BOUNDED CONTEXT delimits the applicability of a particular model so that team members have a clear and shared understanding of what has to be consistent and how it relates to other CONTEXTS.

Within that CONTEXT, work to keep the model logically unified, but do not worry about applicability outside those bounds.

In other CONTEXTS, other models apply, with differences in terminology, in concepts and rules, and in **dialects** of the UBIQUITOUS LANGUAGE.

By drawing an explicit boundary, you can keep the model pure, and therefore potent, where it is applicable.

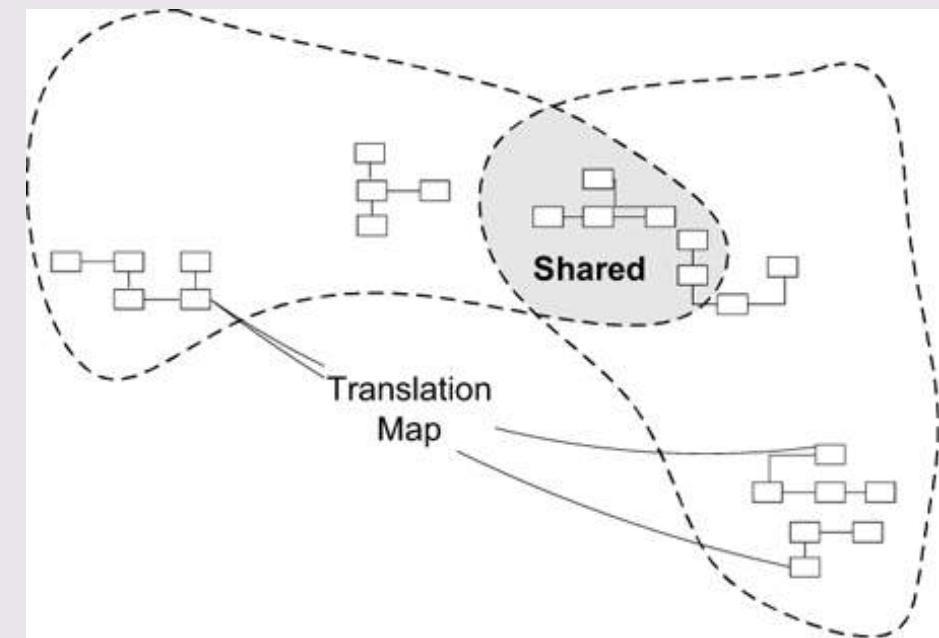


# BookWurm Bounded Contexts?

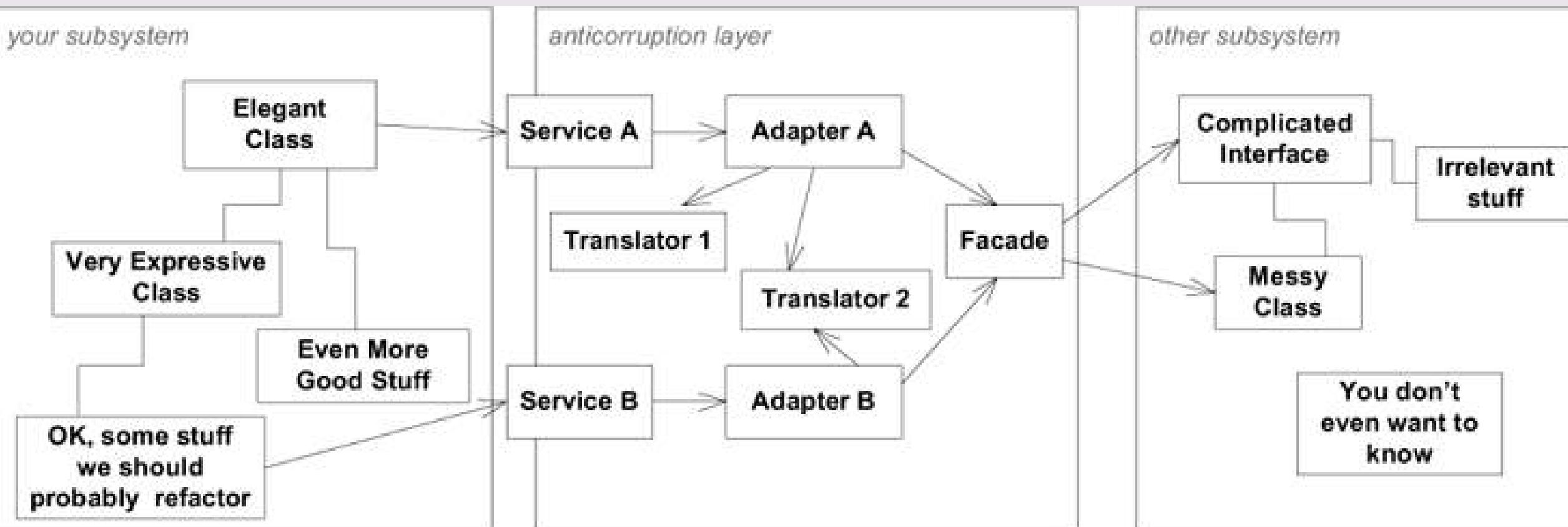
- Users
- Books
- Reading progress
- Reviews
- Feeds
- Friends and connections
- ...
- External: ActivityPub
- External: Different book catalogs / libraries
- External: Import/export of GoodReads

# Relationships Between Bounded Contexts C1 and C2

- **Shared kernel:**  
C1 and C2 share concepts
- **Customer C1 / supplier C2:**  
Upstream C1 uses downstream C2
- **Conformist:**  
C1 follows all changes of C2
- **Anti-corruption layer:**  
C1 hides the mess of C2



# The Anti-Corruption Layer



ACL: see also Adapter Design Pattern

# Context Map

Visualize bounded context dependencies in a context map



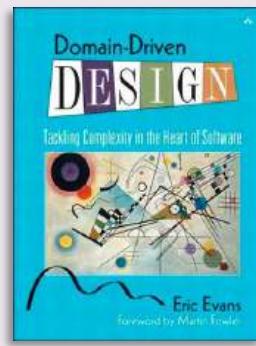
# Distillation: Reduction to the Core

- Models quickly become too large
- Focus on the **core**, captured in a *domain vision statement*
- Separate off **subdomains, generic or supporting**

# “Tactical” vs “Strategic DDD

- Tactical:
  - Develop domain model within single bounded context
- Strategic:
  - Manage connections between different bounded contexts
  - “Context map” highlights connections and translations
  - “Distillation” highlights need to reduce model to its core

# DDD Book



- Extremely well-written book arguing for domain modeling
- Coherent collection of actionable patterns that will help you
- Rich book about organizing domain knowledge
  - Yet also very close to concrete code
- Immensely popular, also because of one popular use:
  - One context == one micro-service == one team

# Why Event Storming?



Domain Experts

Understand the domain

How does domain  
knowledge become  
software?



Software Engineers

Structure knowledge  
to become software

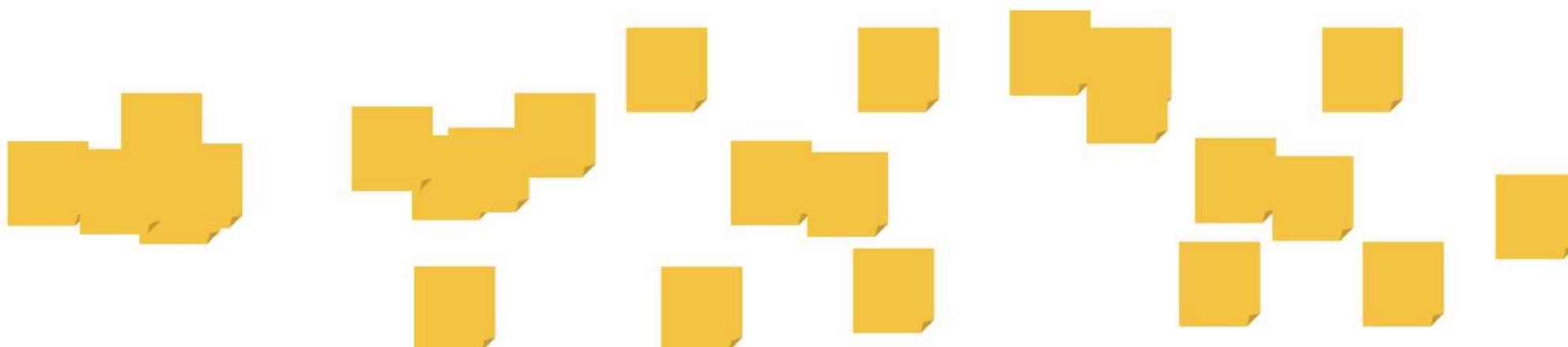
# What is Event Storming?

- Event in the past
- At least noun + verb
- Verb in past tense
- Write event on orange sticky

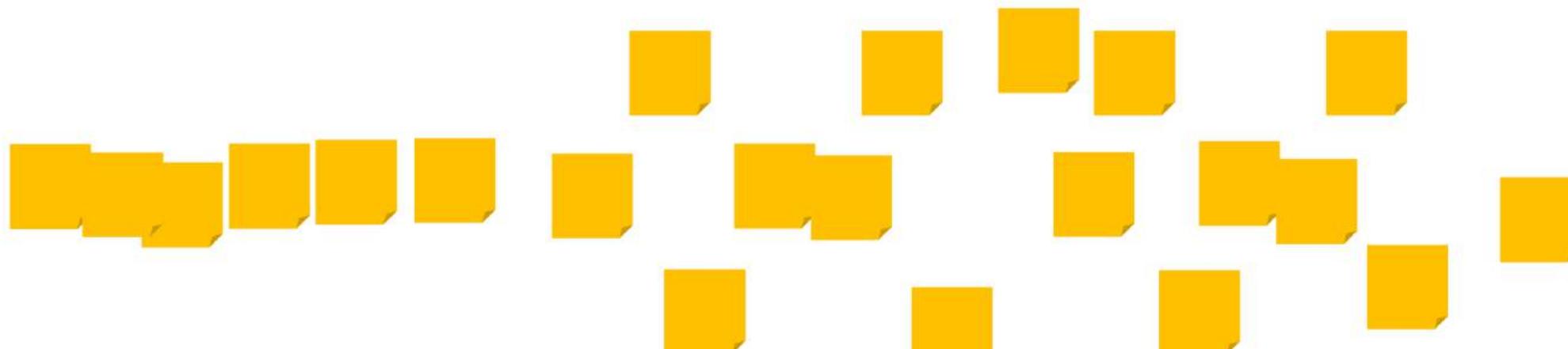
Order  
accepted

# Phase: Chaotic Exploration

- Create as many events as possible

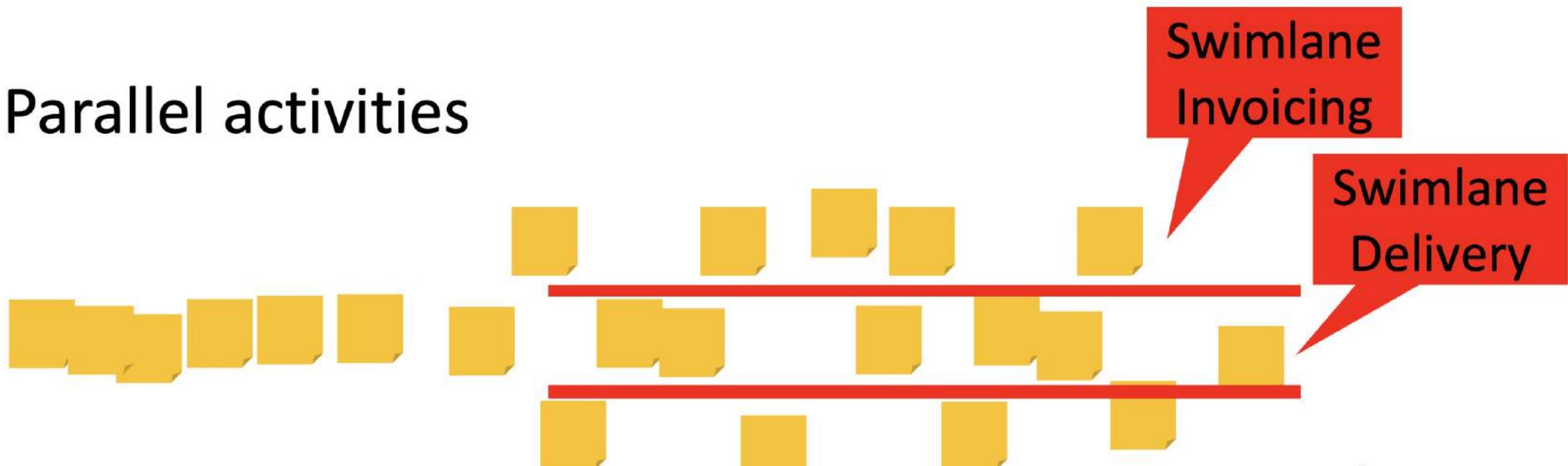


# Phase: Enforce the timeline



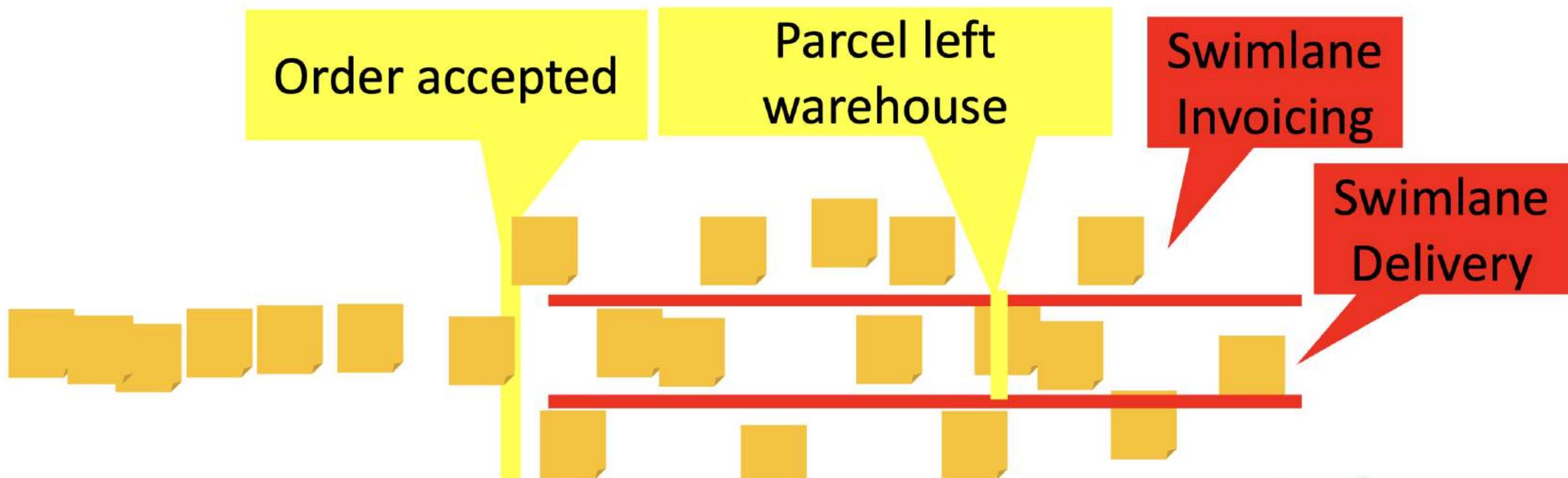
# Phase: Identify Swim Lanes

- Parallel activities



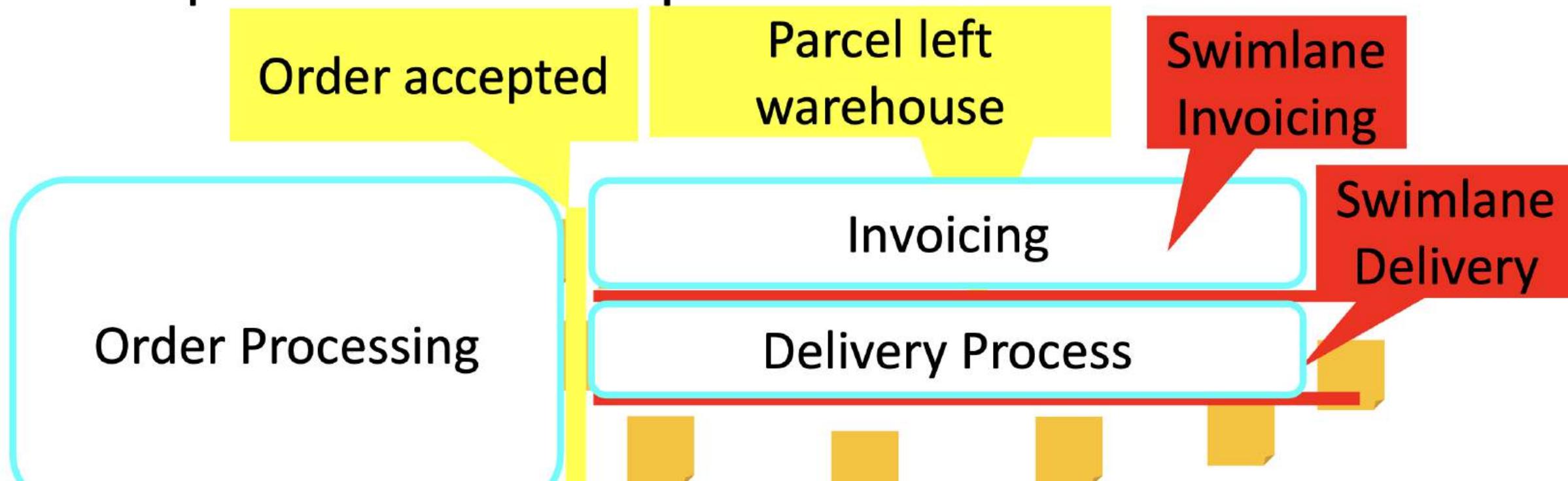
# Phase: Identify Pivotal Events

- Afterwards the world is different



# Identify Candidates for Bounded Contexts

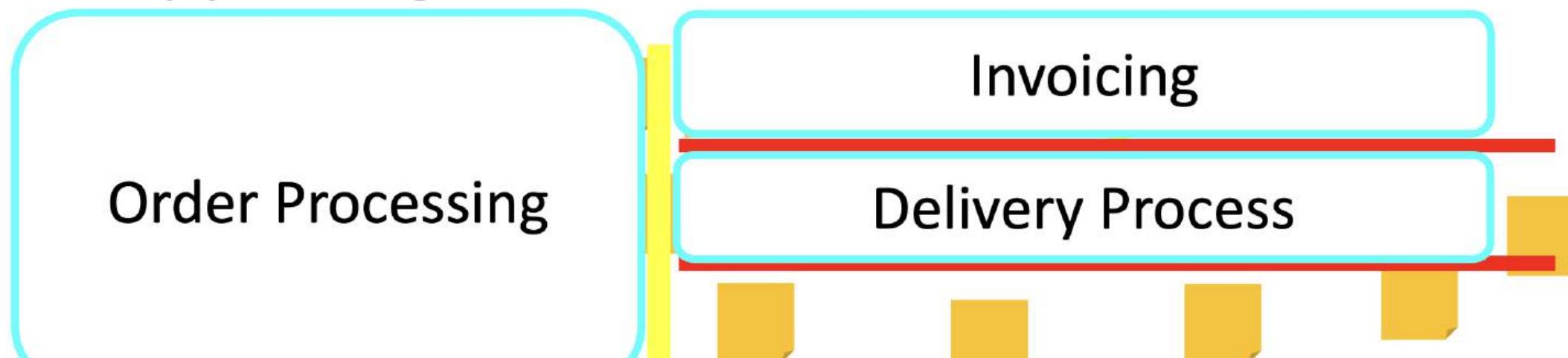
- Areas between swim lanes and pivotal events are good candidates for Bounded Contexts
- Not all pivotal events separate bounded contexts



# What is the Core Domain?

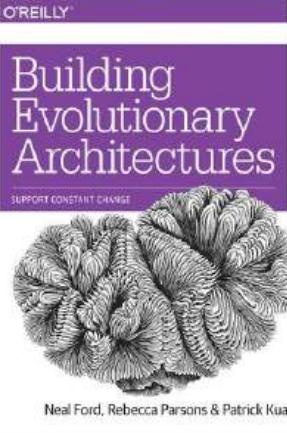


- We delivery quickly and reliably
- So: core domain = delivery process
- Rest: generic subdomain  
...or supporting subdomain



# BookWurm Even Storming?

- What are meaningful events?
- What domain boundaries will emerge?

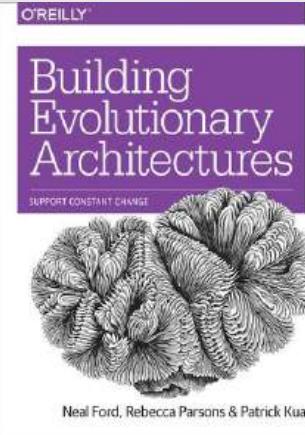


# The Bounded Context (I)

*“Eric Evans’s book *Domain-Driven Design* has deeply influenced modern architectural thinking. [...]”*

*“DDD defines the bounded context, where everything related to the domain is visible internally but opaque to other bounded contexts. [...]”*

*“DDD influenced several modern architectural styles, along with related factors like team organization.”*



# The Bounded Context (II)

*Before DDD, developers sought holistic reuse across common entities within the organization. Yet, creating common shared artifacts causes a host of problems, such as coupling, more difficult coordination, and increased complexity.*

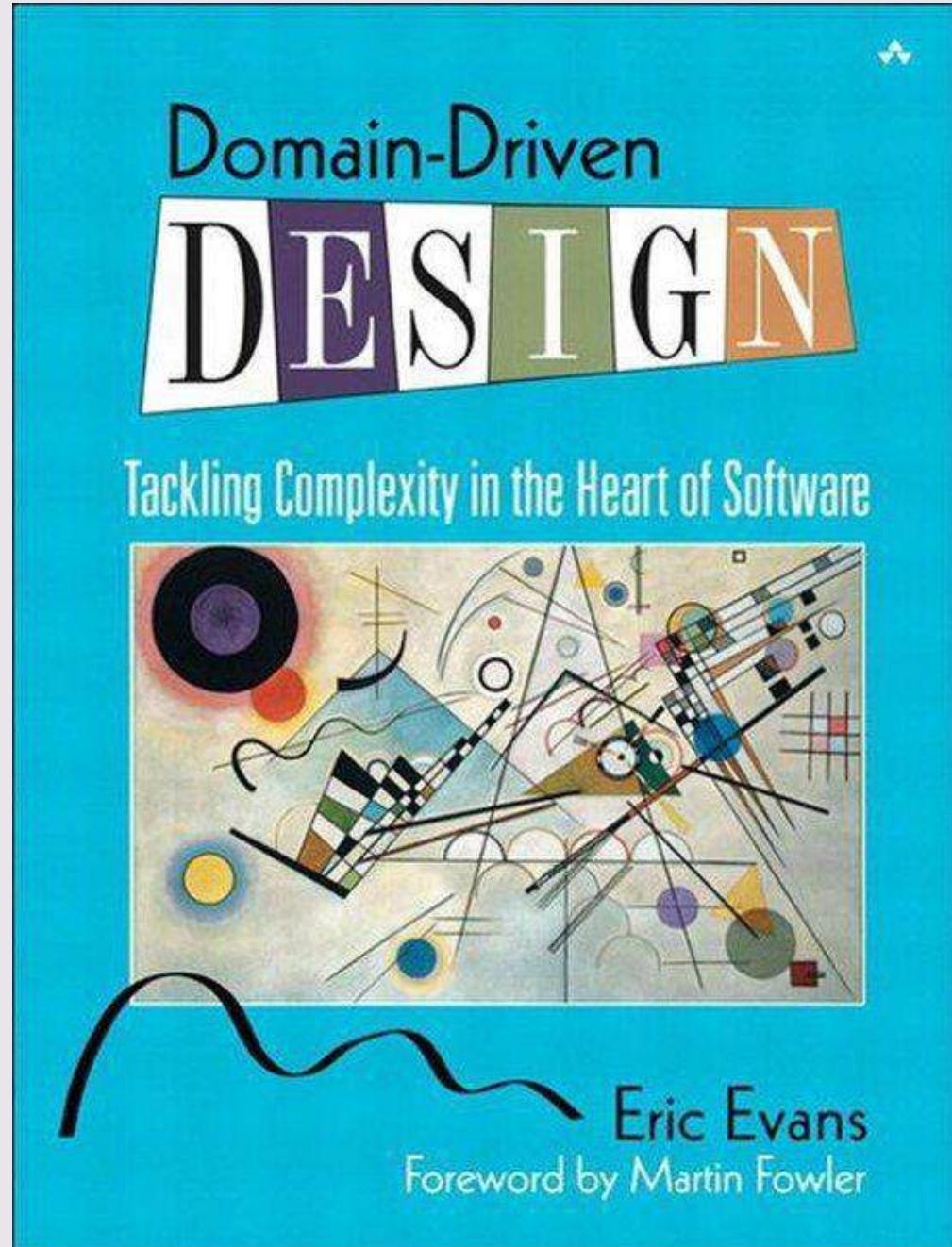
*The bounded context concept recognizes that each entity works best within a localized context.*

*Thus, instead of creating a unified Customer class across the entire organization, each problem domain can create their own, and reconcile differences at integration points.*

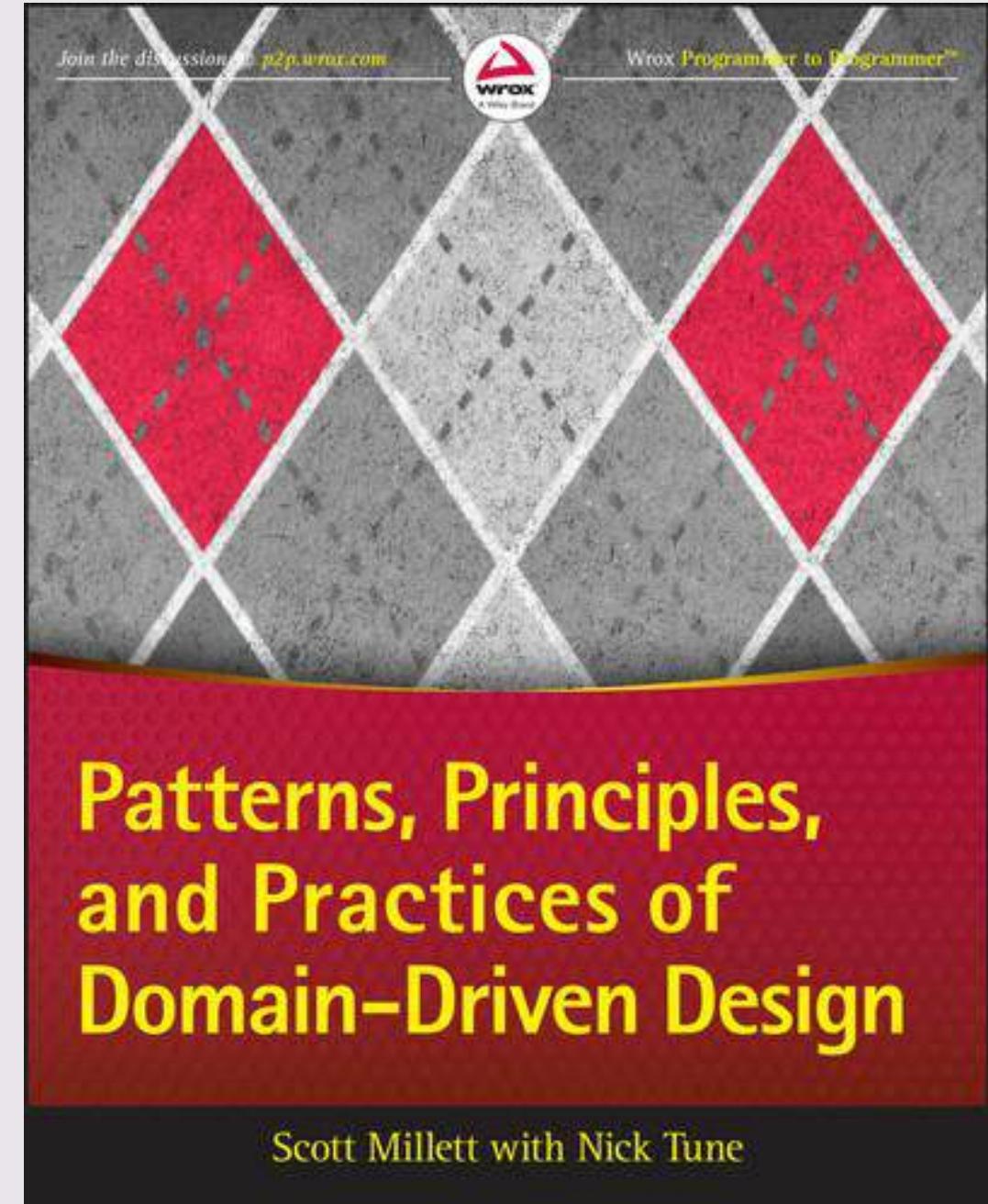
# Constant Refactoring for “Supple Design”

- Our domain understanding grows as we work on the system
- New features will require new domain concepts, often across bounded contexts
- To keep the design “supple” we need to constantly refactor *at the level of the domains*





2003



2015

# Preparing for the exam

- Eric Evans, chapters:
  - 1, 2; the ubiquitous language, role of domain model, ...
  - 5, 6; entities, aggregates, repositories ...
  - 14: bounded contexts, context maps, anti-corruption, ...
- + additional material on slides (event storming)
- Understand core concepts in chapters and on slides
- Ability to apply them to example design problems
- Ability to capture model essence in UML diagrams

# Domain-Driven DESIGN

Tackling Complexity in the Heart of Software



Eric Evans  
Foreword by Martin Fowler

# Continuous Learning

*“To create a supple,  
knowledge-rich design  
calls for a versatile,  
shared team language,  
and a lively  
experimentation with language that  
seldom happens  
on software projects”*

# CSE2115: SE Methods Professional Agility at Scale

Lecture 11 | prof.dr.ir. Rini van Solingen

26-9-2024





# Rini van Solingen

Part-time full professor in SW Engineering group



d.m.vansolingen@tudelft.nl

[www.rinivansolingen.com](http://www.rinivansolingen.com)

Author, speaker, professor and entrepreneur.

Expertise: distributed work, sw-startups. Speed, impact and agility of people and organizations.

Prowareness, Momo Medical, TU-Delft, Nyenrode, .....



# Rini van Solingen

Part-time full professor in SW Engineering group



# Learning objectives on Scaling Agility

1. Understand why iterative and incremental development is a solution for VUCA settings
2. Understand that iterative and incremental development is about high-frequency delivery, continuous (and thus automated) verification and validation, and continuous improvement
3. Know the most important agile approaches for teams (Scrum, KanBan, ScrumBan, eXtreme Programming and Obeya)
4. Knowing what the most important prerequisites are for large-scale agile and the main barriers to large-scale agile
5. Know key agile scaling frameworks (SAFe, Spotify, LESS, Scrum@Scale)
6. Be able to reason about strengths and weaknesses of Scrum and SAFe

# Agenda | Part I (10:45-11:30) and part II (11:45-12:30)

**1.1**

Why agile?

- VUCA
- Complexity

**1.2**

How to tackle?

- Iterative delivery
- Inspect & Adapt

**1.3**

Frameworks

Scrum, KanBan,  
ScrumBan, XP,  
Obeya...

**1.4**

Scaling Agile

Discussion

**1.5**

BREAK

optional Q&A

**2.1**

Why scale?

- Speed
- VUCA

**2.2**

How to scale?

- Principles
- Feedback & learning

**2.3**

Frameworks

SAFe,  
Scrum@Scale,  
Spotify, Less,  
Nexus...

**2.4**

Challenges

- dependencies
- no feedback/  
learning loop
- over-processing

**2.5**

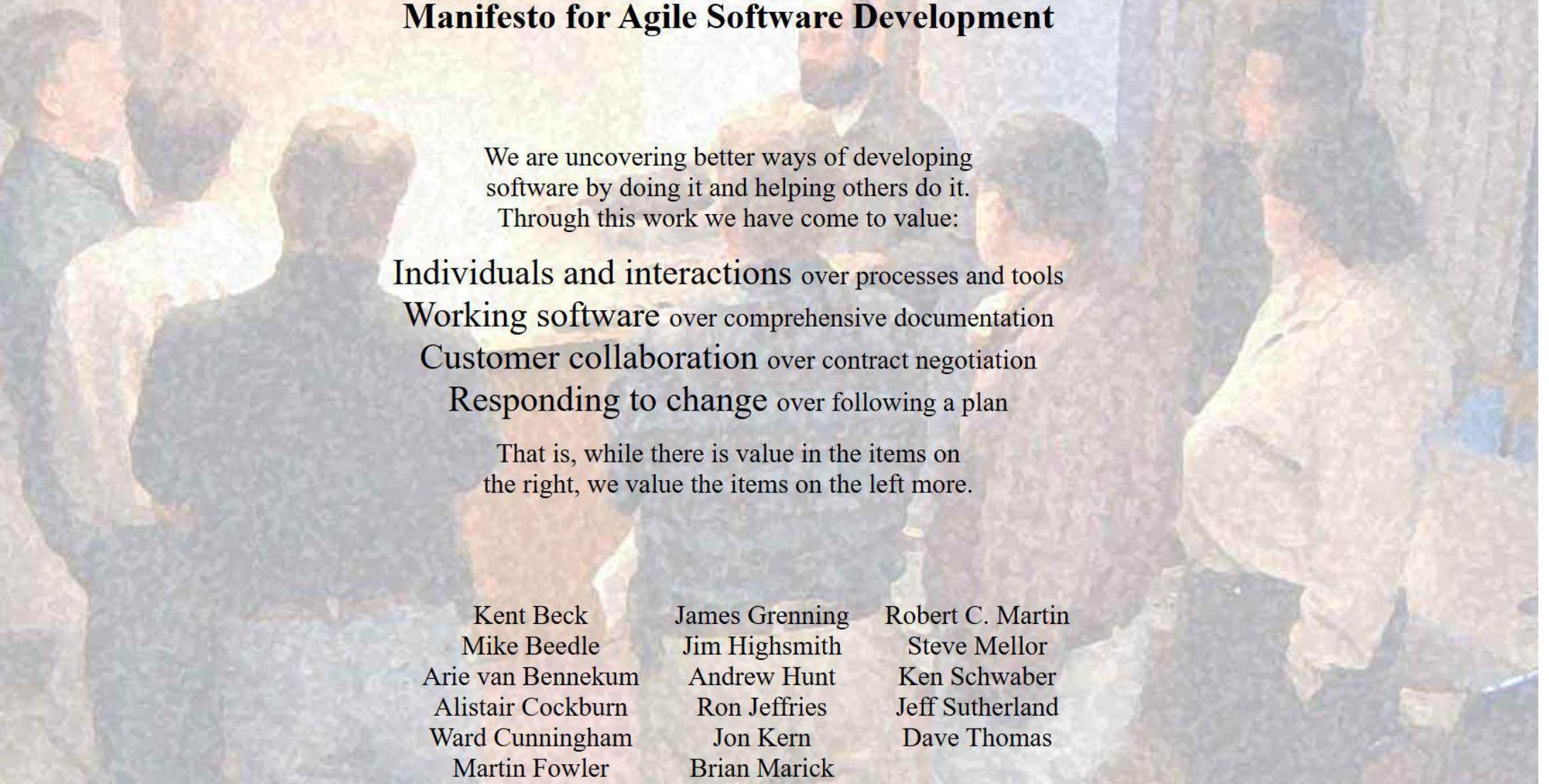
Quiz

# What is agile?



# What is agile? – The Values:

## Manifesto for Agile Software Development



We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

**Individuals and interactions** over processes and tools

**Working software** over comprehensive documentation

**Customer collaboration** over contract negotiation

**Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck

Mike Beedle

Arie van Bennekum

Alistair Cockburn

Ward Cunningham

Martin Fowler

James Grenning

Jim Highsmith

Andrew Hunt

Ron Jeffries

Jon Kern

Brian Marick

Robert C. Martin

Steve Mellor

Ken Schwaber

Jeff Sutherland

Dave Thomas

# What is agile? – The Principles:

## Principles behind the Agile Manifesto

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able  
to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and  
adjusts its behavior accordingly.



# Agile in Team(s)

Basics under agile

# Agenda | Part I (10:45-11:30) and part II (11:45-12:30)

**1.1**

Why agile?

- VUCA
- Complexity

**1.2**

How to tackle?

- Iterative delivery
- Inspect & Adapt

**1.3**

Frameworks

Scrum, KanBan,  
ScrumBan, XP,  
Obeya...

**1.4**

Scaling Agile

Discussion

**1.5**

BREAK

optional Q&A

**2.1**

Why scale?

- Speed
- VUCA

**2.2**

How to scale?

- Principles
- Feedback & learning

**2.3**

Frameworks

SAFe,  
Scrum@Scale,  
Spotify, Less,  
Nexus...

**2.4**

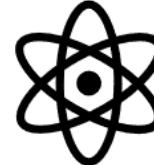
Challenges

- dependencies
- no feedback/  
learning loop
- over-processing

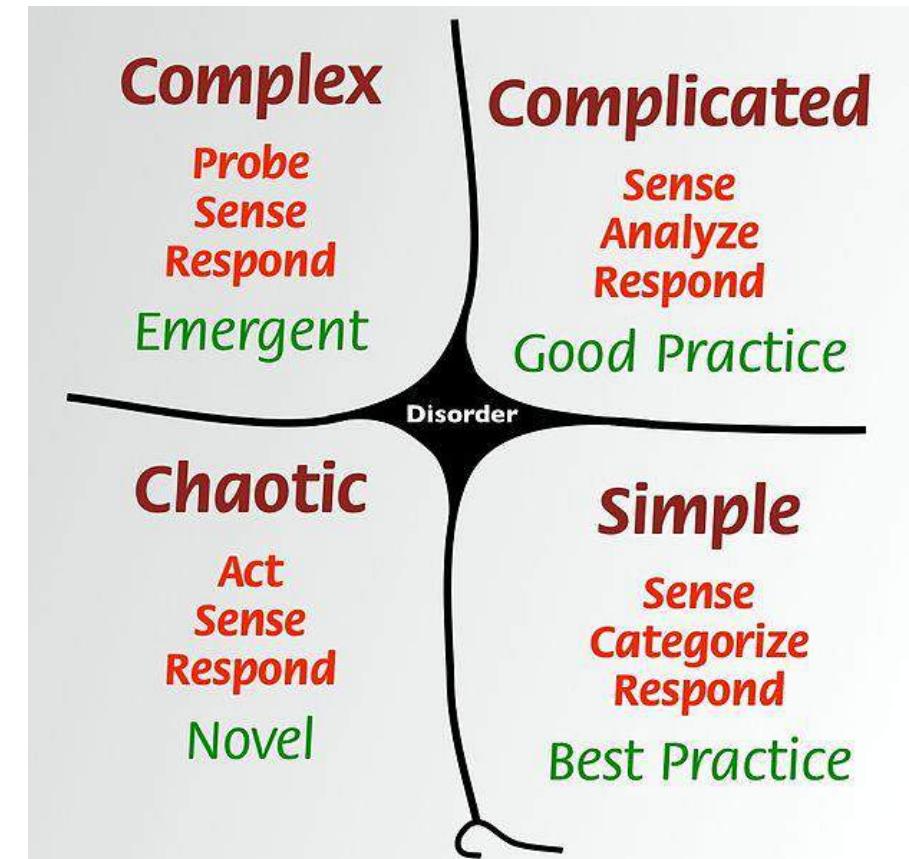
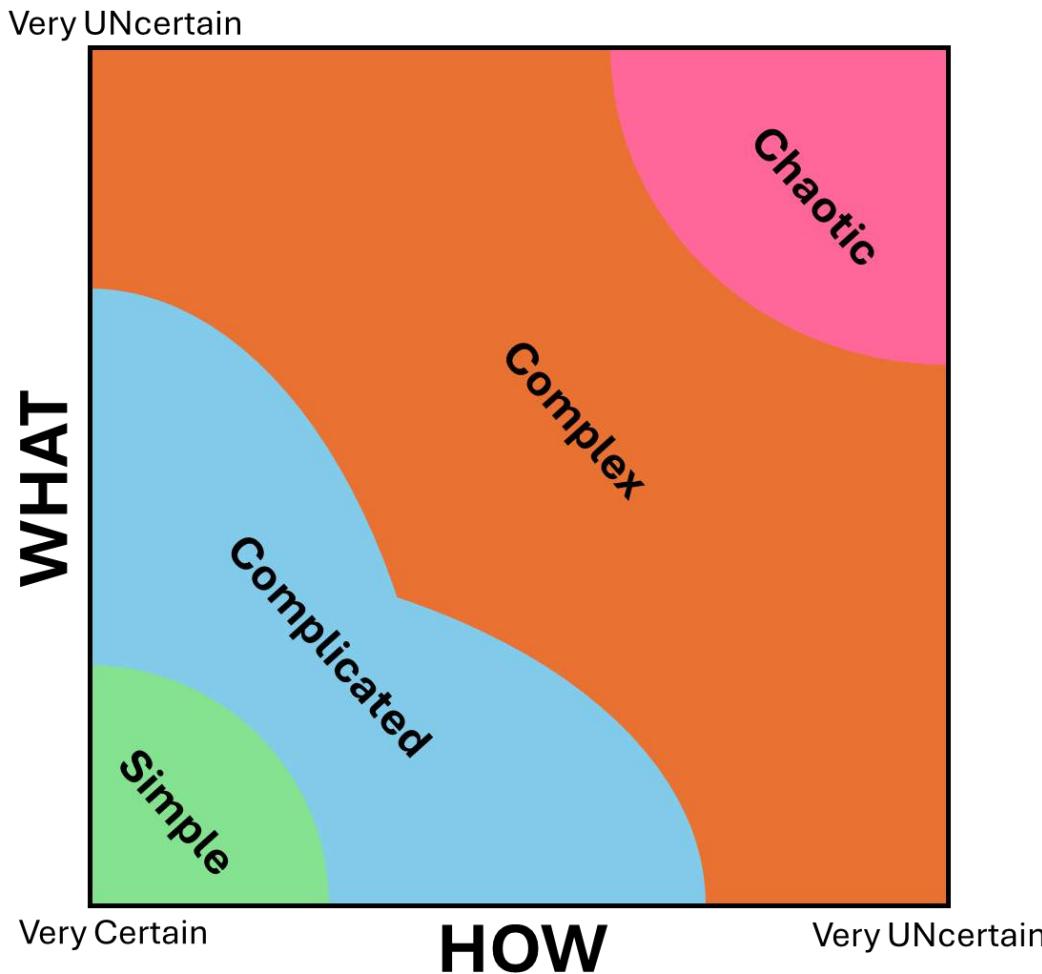
**2.5**

Quiz

# VUCA is dominating in many fields

V	U	C	A
Volatility	Uncertainty	Complexity	Ambiguity
			ЕпЭш
rapid and seemingly random fluctuations	incomplete information about future trends	high cross-linking density and interactions	many possible interpretations

# More and more work is executed in Complexity



Ralph Stacey model (left) Cynefin framework (right)

# Agenda | Part I (10:45-11:30) and part II (11:45-12:30)

**1.1**

Why agile?

- VUCA
- Complexity

**1.2**

How to tackle?

- Iterative delivery
- Inspect & Adapt

**1.3**

Frameworks

Scrum, KanBan,  
ScrumBan, XP,  
Obeya...

**1.4**

Scaling Agile

Discussion

**1.5**

BREAK

optional Q&A

**2.1**

Why scale?

- Speed
- VUCA

**2.2**

How to scale?

- Principles
- Feedback & learning

**2.3**

Frameworks

SAFe,  
Scrum@Scale,  
Spotify, Less,  
Nexus...

**2.4**

Challenges

- dependencies
- no feedback/  
learning loop
- over-processing

**2.5**

Quiz



# Exercise

Complexity example:

- Developing iPad app
  - Within 1 working week
  - For the flag-ship store that sells sunglasses
  - Guaranteed ready on time (end of day 5)
  - Customer and users are happy
- 
- No idea what the app should do
  - No idea who the users will be
  - No idea what the biggest challenge is

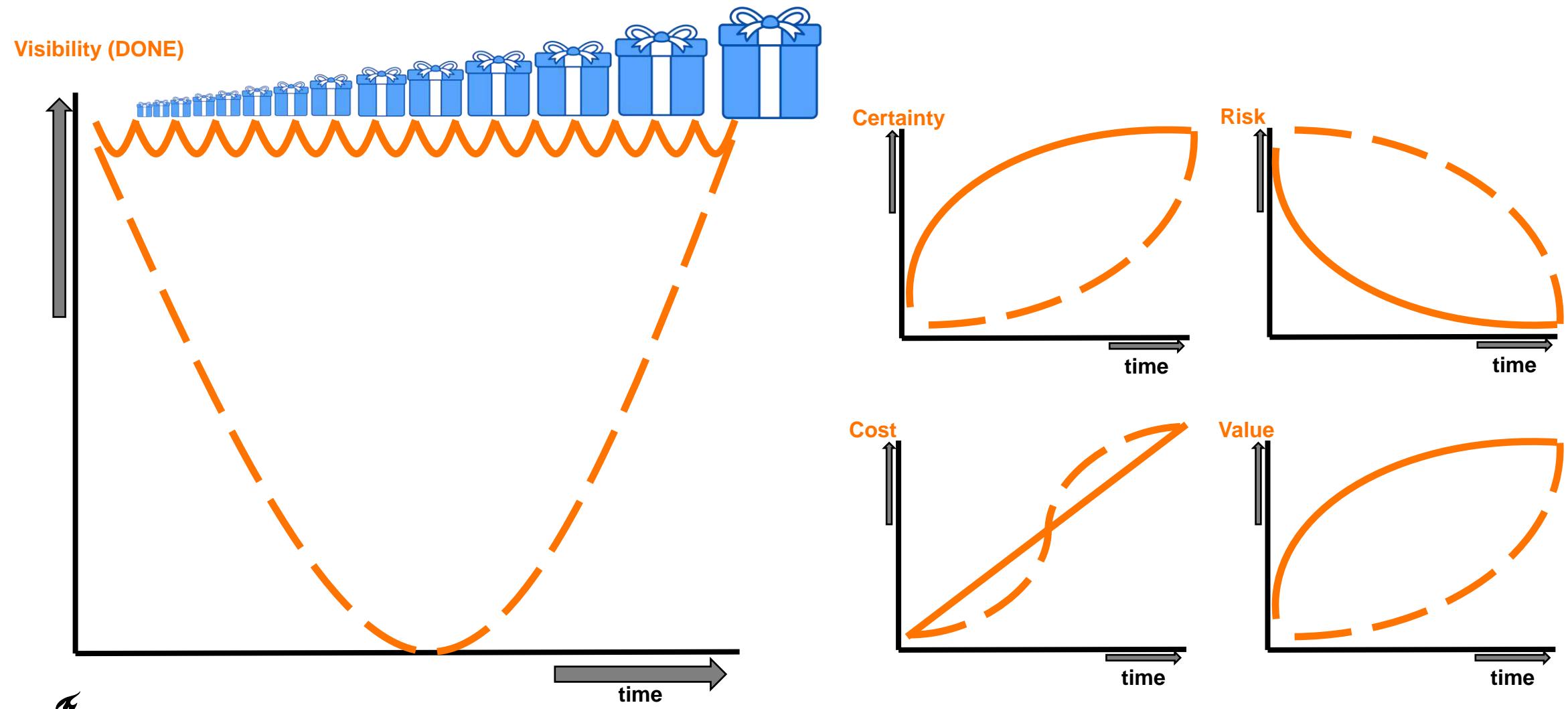
# Video



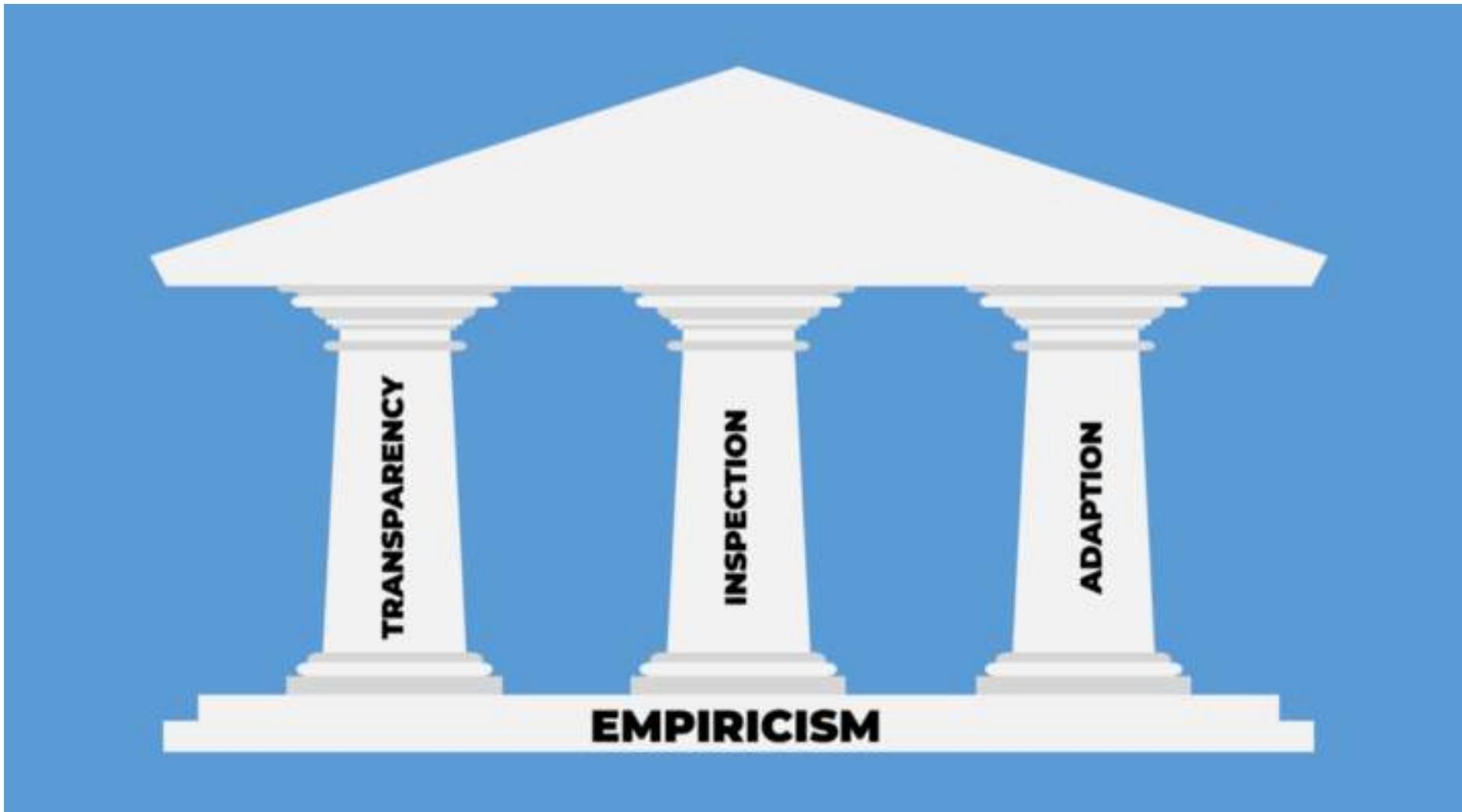
# What they are doing/have available:

1. In context: close to the customer/together with customers
2. Very short iterations – finishing and delivering very quickly
3. Continuous validation – what is customer value and what not?
4. Finishing and delivering the most valuable first
5. There is constantly a product that is really DONE (possibility to stop)
6. Cross-functional teams with all the necessary competencies present
7. Lots and intense human contact and interaction
8. Safe environment in which experimentation, discovery and learning is allowed
9. Visual management to coordinate the work

# Iterative delivery as antidote to complexity



# Short-cyclic empiricism: deliver and learn



# Agenda | Part I (10:45-11:30) and part II (11:45-12:30)

**1.1**

Why agile?

- VUCA
- Complexity

**1.2**

How to tackle?

- Iterative delivery
- Inspect & Adapt

**1.3**

Frameworks

Scrum, KanBan,  
ScrumBan, XP,  
Obeya...

**1.4**

Scaling Agile

Discussion

**1.5**

BREAK

optional Q&A

**2.1**

Why scale?

- Speed
- VUCA

**2.2**

How to scale?

- Principles
- Feedback & learning

**2.3**

Frameworks

SAFe,  
Scrum@Scale,  
Spotify, Less,  
Nexus...

**2.4**

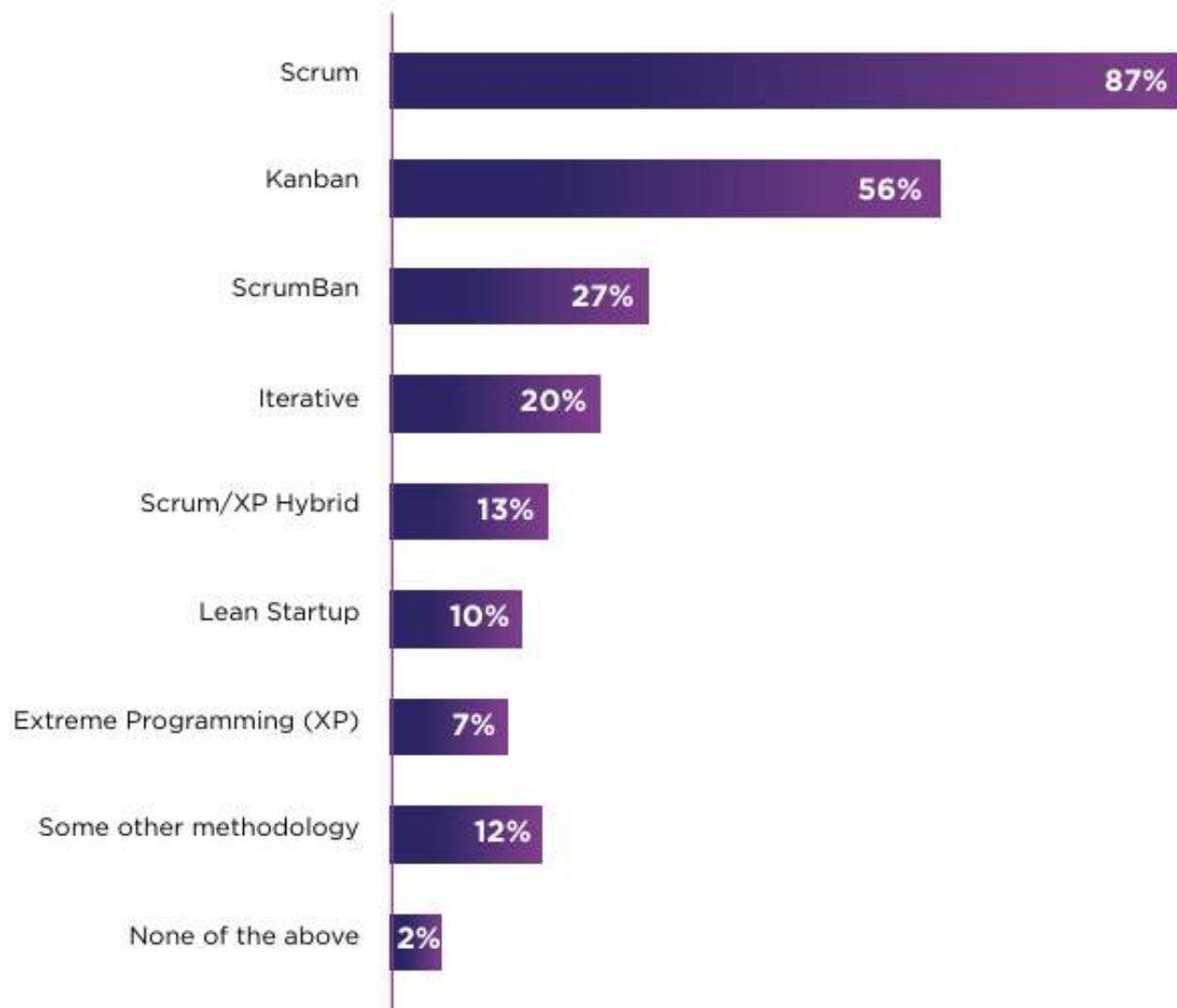
Challenges

- dependencies
- no feedback/  
learning loop
- over-processing

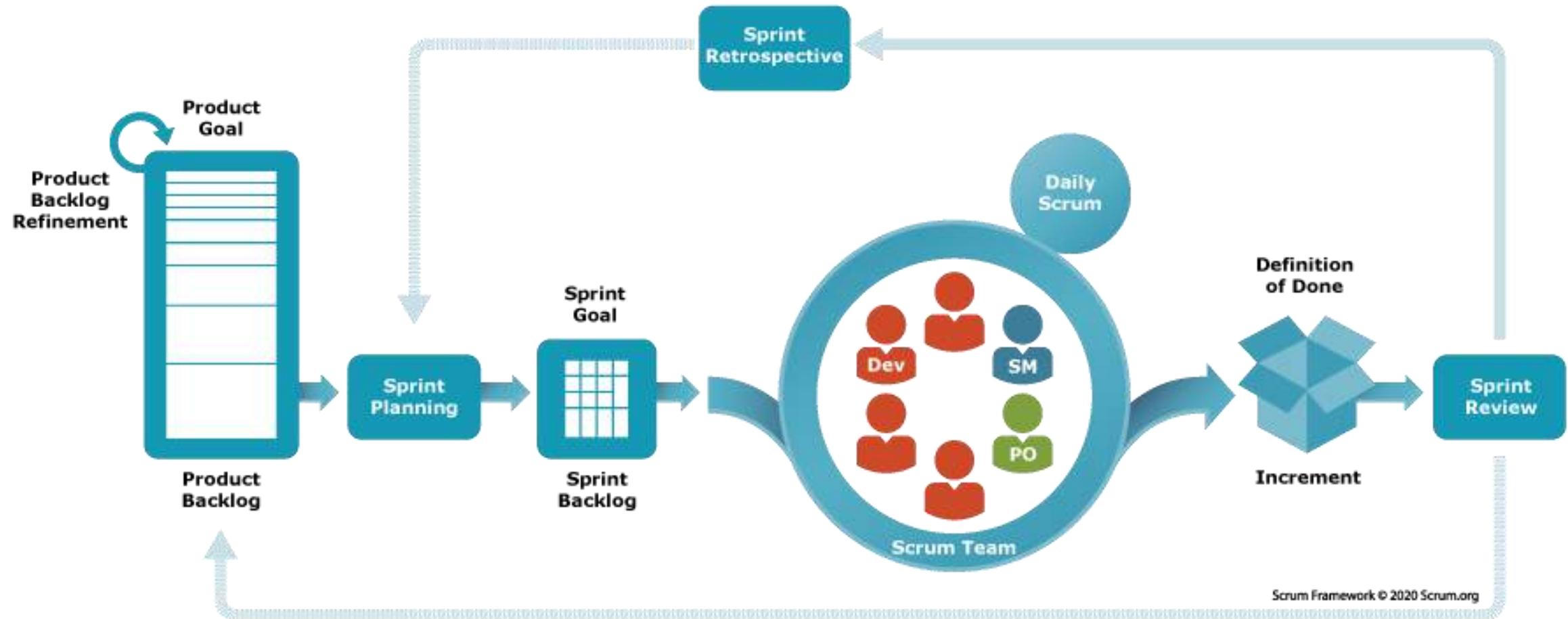
**2.5**

Quiz

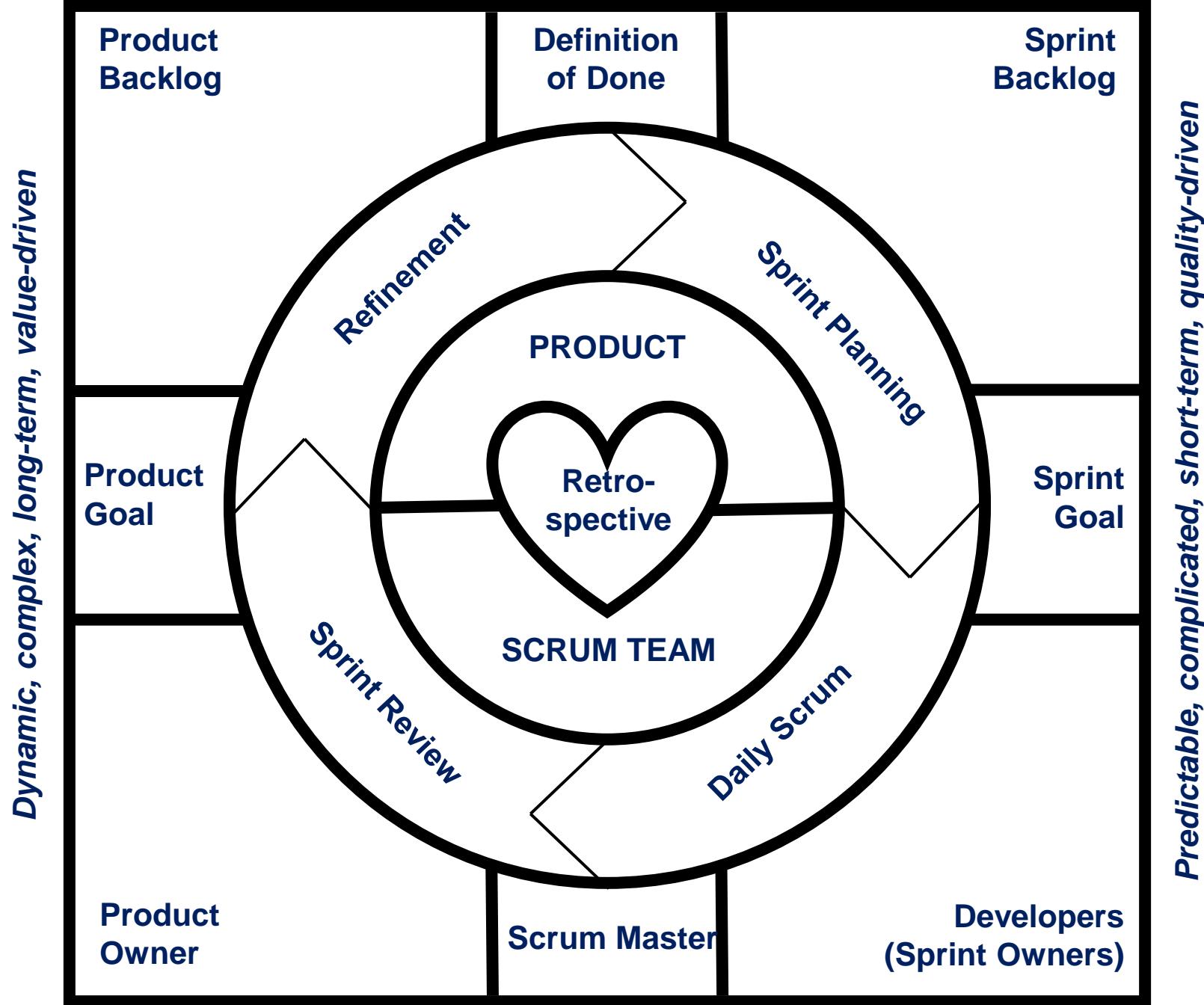
# Frameworks at team level in practice



# Scrum



# Scrum

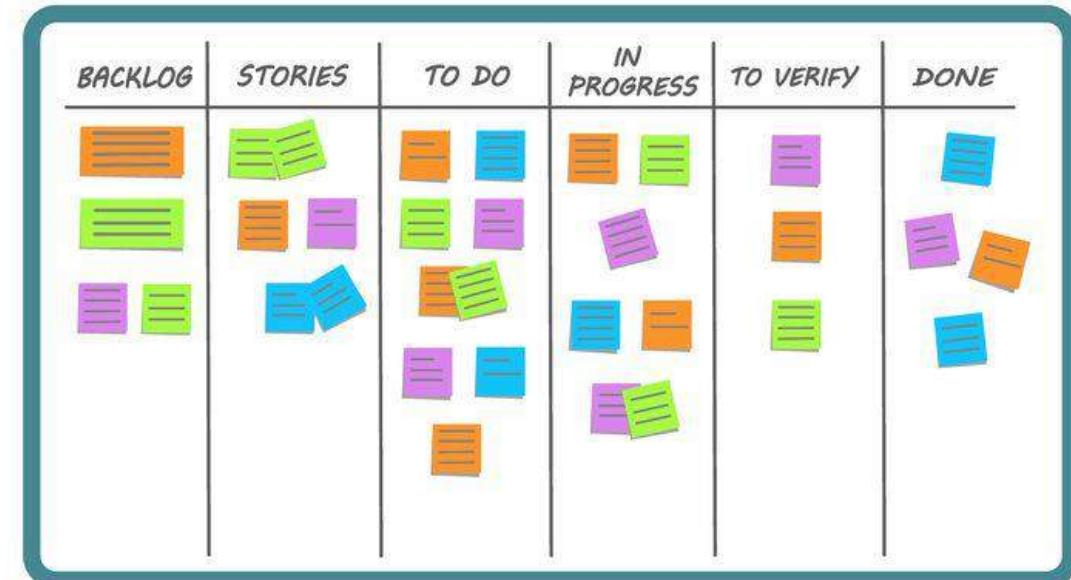


# KanBan

## 4 KANBAN PRINCIPLES

The four principles of Kanban are *visualizing the work, limiting the work in progress, focusing on flow, and continuous improvement*. Each of these principles help to create a successful Kanban Board and ensures that the team is always working on the most valuable tasks.

- 1** Visualize the Workflow
- 2** Limit the Work in Progress (WIP)
- 3** Focus on Flow
- 4** Continuous Improvement

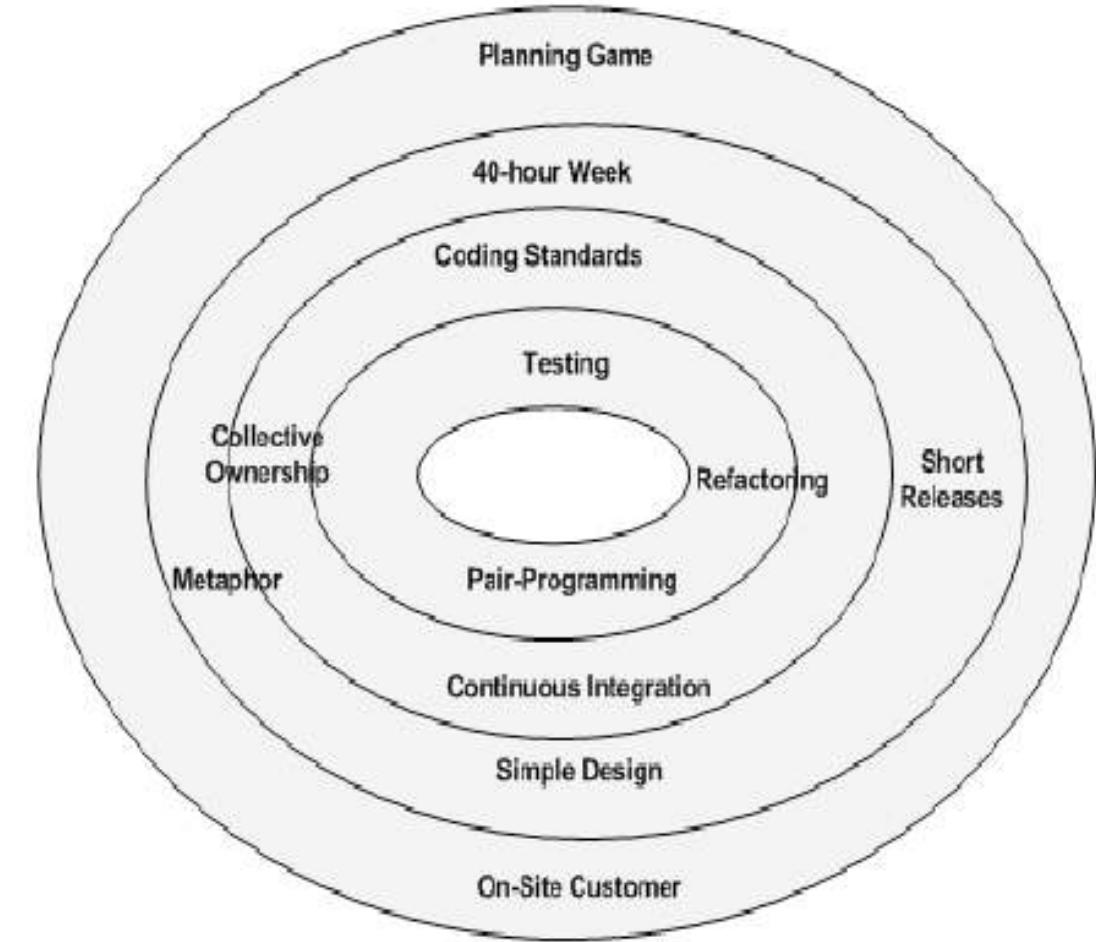
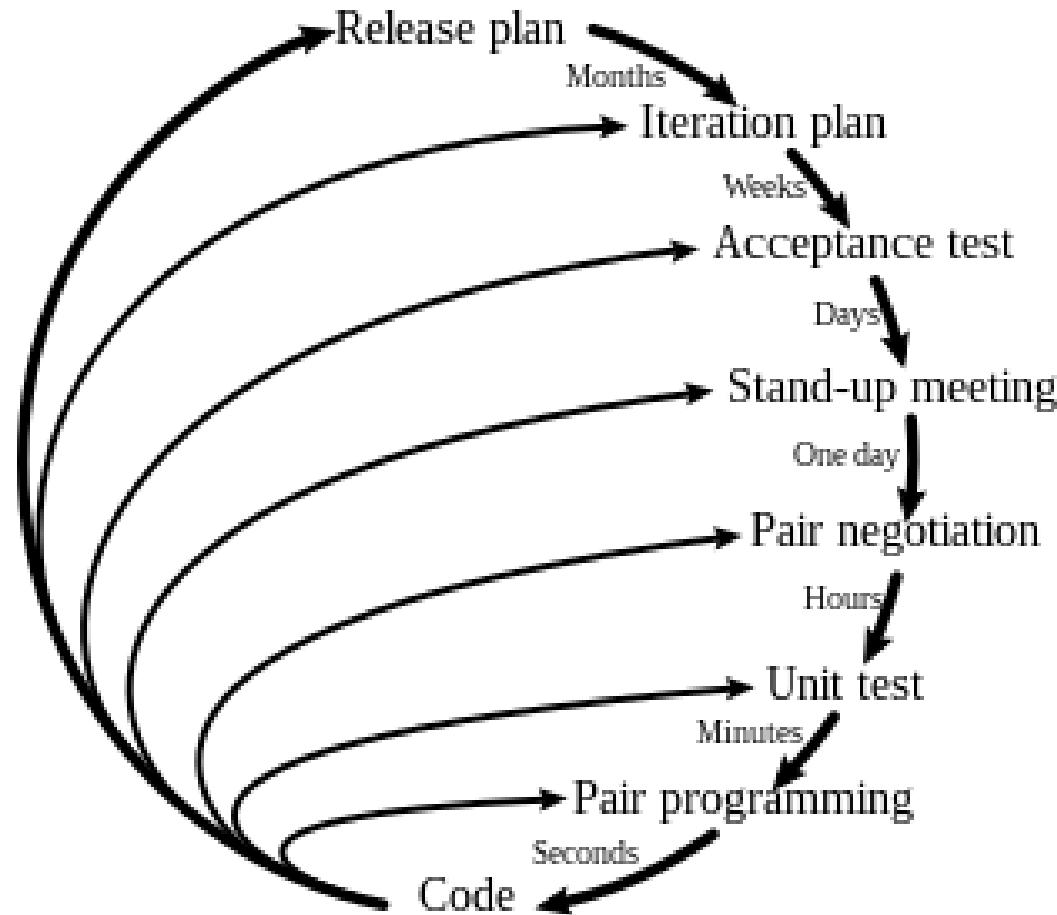


# ScrumBan



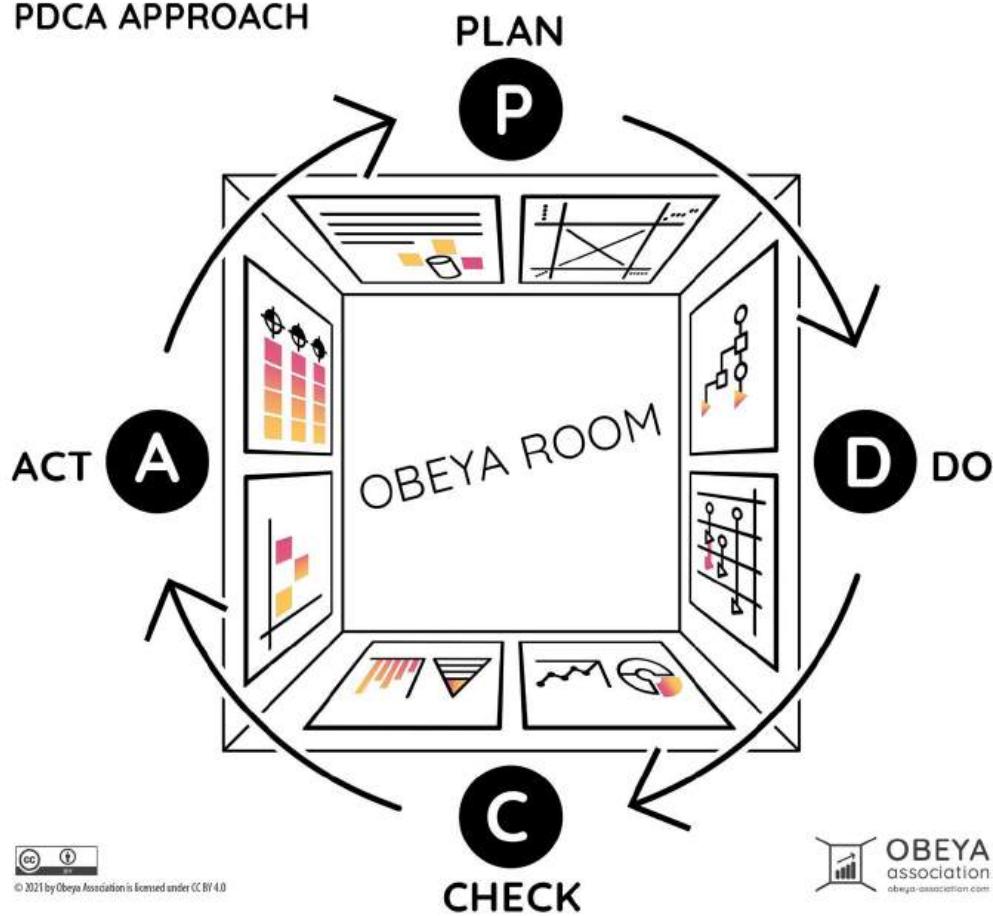
# eXtreme Programming (XP)

## Planning/feedback loops



# Obeya and more.....

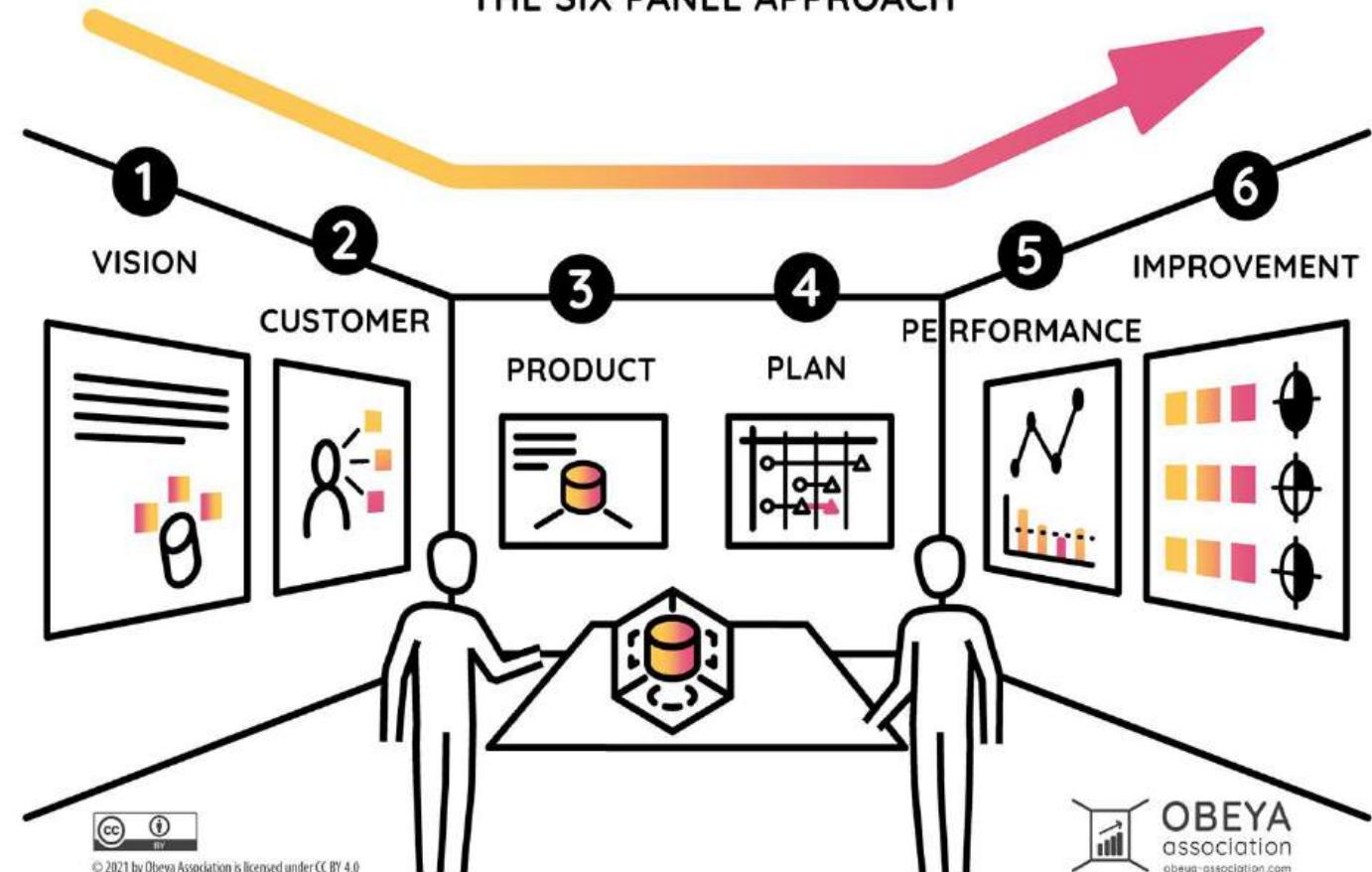
OBEYA:  
PDCA APPROACH



© 2021 by Obeya Association is licensed under CC BY 4.0

OBEYA  
association  
[obeya-association.com](http://obeya-association.com)

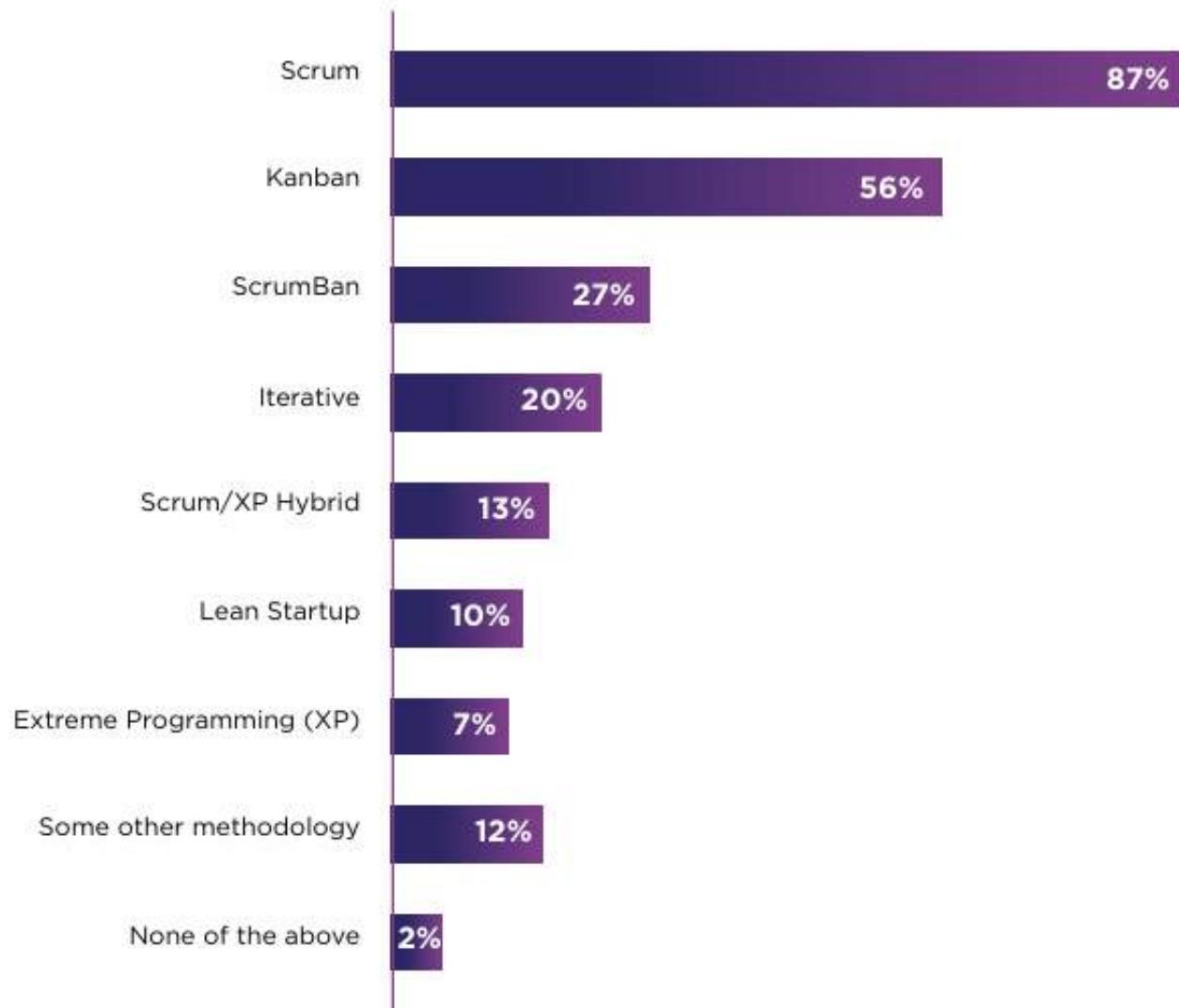
OBEYA:  
THE SIX PANEL APPROACH



© 2021 by Obeya Association is licensed under CC BY 4.0

OBEYA  
association  
[obeya-association.com](http://obeya-association.com)

# Frameworks at team level in practice



# Agenda | Part I (10:45-11:30) and part II (11:45-12:30)

1.1

Why agile?

- VUCA
- Complexity

1.2

How to tackle?

- Iterative delivery
- Inspect & Adapt

1.3

Frameworks

Scrum, KanBan,  
ScrumBan, XP,  
Obeya...

1.4

Scaling Agile  
Discussion

1.5

BREAK  
optional Q&A

2.1

Why scale?

- Speed
- VUCA

2.2

How to scale?

- Principles
- Feedback & learning

2.3

Frameworks

SAFe,  
Scrum@Scale,  
Spotify, Less,  
Nexus...

2.4

Challenges

- dependencies
- no feedback/  
learning loop
- over-processing

2.5

Quiz



# Exercise

What are the challenges to apply these ways of working at large scale?

E.g. in organizations with 1.000-10.000 developers

What may hinder them to do this successfully?

What bottlenecks would you expect?

Discuss for 7 minutes with 3-5 students  
(remain seated)

## Challenges Agile at Scale?

29

40

Combining products

communication on what to focus on between people/teams and combining it

challenging to have customer involvement

Dependencies

The fact that you're doing agile instead of valuing your mental health

Synchronization

Coordinating

Progress overview

Managing a company which is vertically structured becomes very hard. Not everyone can be aligned :(

Not being agile

Dependencies between teams

Chain of command is screwed

SHREK Once upon a time there was a lovely princess. But she had an enchantment upon her of a fearful sort which coul

Conflict between teams

26-9-2024

crossing of contexts

Underestimating how long a feature will take to implement

Redundancy

Party box

Meeting time in different timezones

Contradicting ideas

Losing the overview because of the bigger scale

Skill issue

Ambiguity especially if its international

leadership

Internal communication

Communicatie

dependeciens between teams

Burnout

Conflicting interests

uncertainty in which feature has the most value

Hella expensive

Recursion

26-9-2024

31

# Agenda | Part I (10:45-11:30) and part II (11:45-12:30)

**1.1**

Why agile?

- VUCA
- Complexity

**1.2**

How to tackle?

- Iterative delivery
- Inspect & Adapt

**1.3**

Frameworks

Scrum, KanBan,  
ScrumBan, XP,  
Obeya...

**1.4**

Scaling Agile

Discussion

**1.5**

BREAK

optional Q&A

**2.1**

Why scale?

- Speed
- VUCA

**2.2**

How to scale?

- Principles
- Feedback & learning

**2.3**

Frameworks

SAFe,  
Scrum@Scale,  
Spotify, Less,  
Nexus...

**2.4**

Challenges

- dependencies
- no feedback/  
learning loop
- over-processing

**2.5**

Quiz



2

# Scaling Agile

Organizing for very many teams

# Agenda | Part I (10:45-11:30) and part II (11:45-12:30)

**1.1**

Why agile?

- VUCA
- Complexity

**1.2**

How to tackle?

- Iterative delivery
- Inspect & Adapt

**1.3**

Frameworks

Scrum, KanBan,  
ScrumBan, XP,  
Obeya...

**1.4**

Scaling Agile

Discussion

**1.5**

BREAK

optional Q&A

**2.1**

Why scale?

- Speed
- VUCA

**2.2**

How to scale?

- Principles
- Feedback & learning

**2.3**

Frameworks

SAFe,  
Scrum@Scale,  
Spotify, Less,  
Nexus...

**2.4**

Challenges

- dependencies
- no feedback/  
learning loop
- over-processing

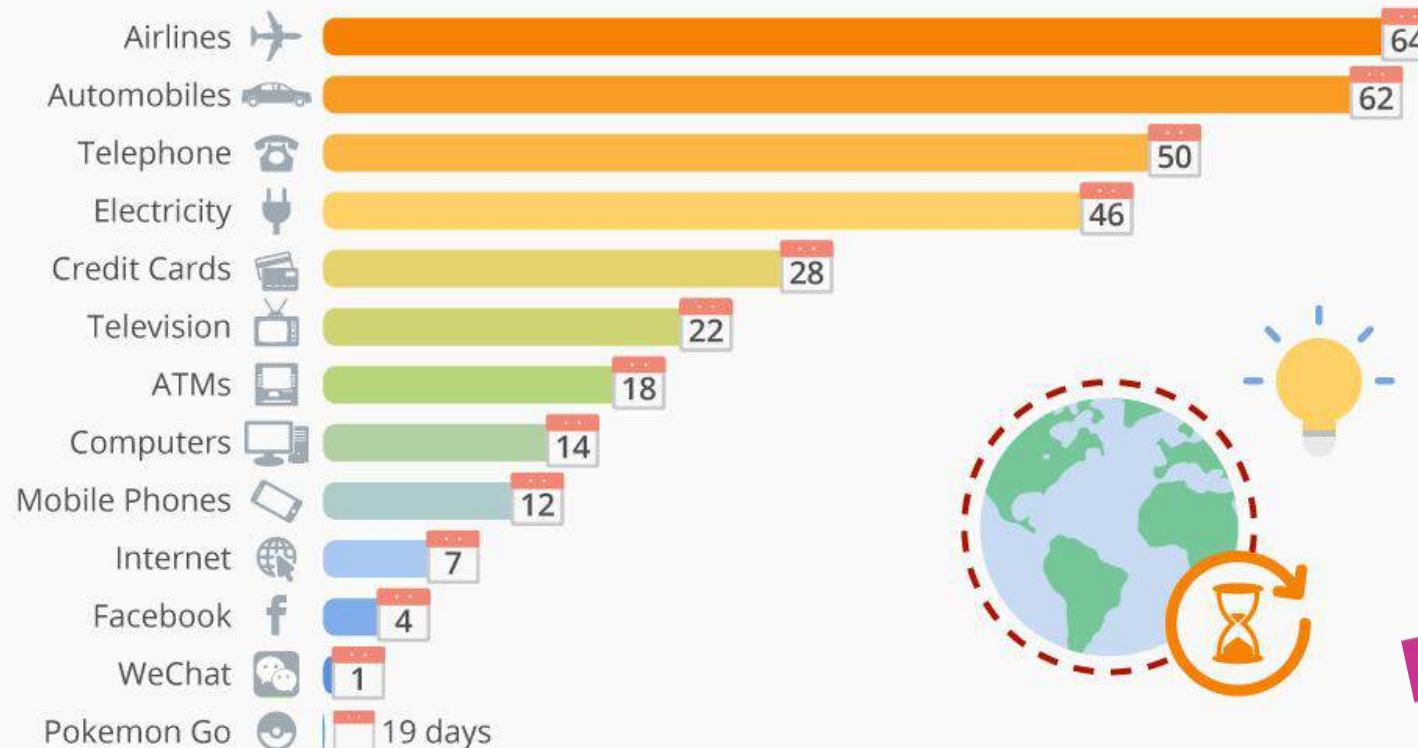
**2.5**

Quiz

# Why scaling agile?

## The Road to Ubiquity is Getting Shorter

Time innovations needed to reach 50 million users (in years)



@StatistaCharts Source: Visual Capitalist



Hindustan Times

Home / India News / Aarogya Setu has 50 million users in 13 days, beats 'Pokémon GO' record

**INDIA NEWS**

**Aarogya Setu has 50 million users in 13 days, beats 'Pokémon GO' record**

India's coronavirus disease (Covid-19) contact-tracing app Aarogya Setu (health bridge) became the world's fastest growing mobile app on Tuesday night with 50 million users in 13 days.

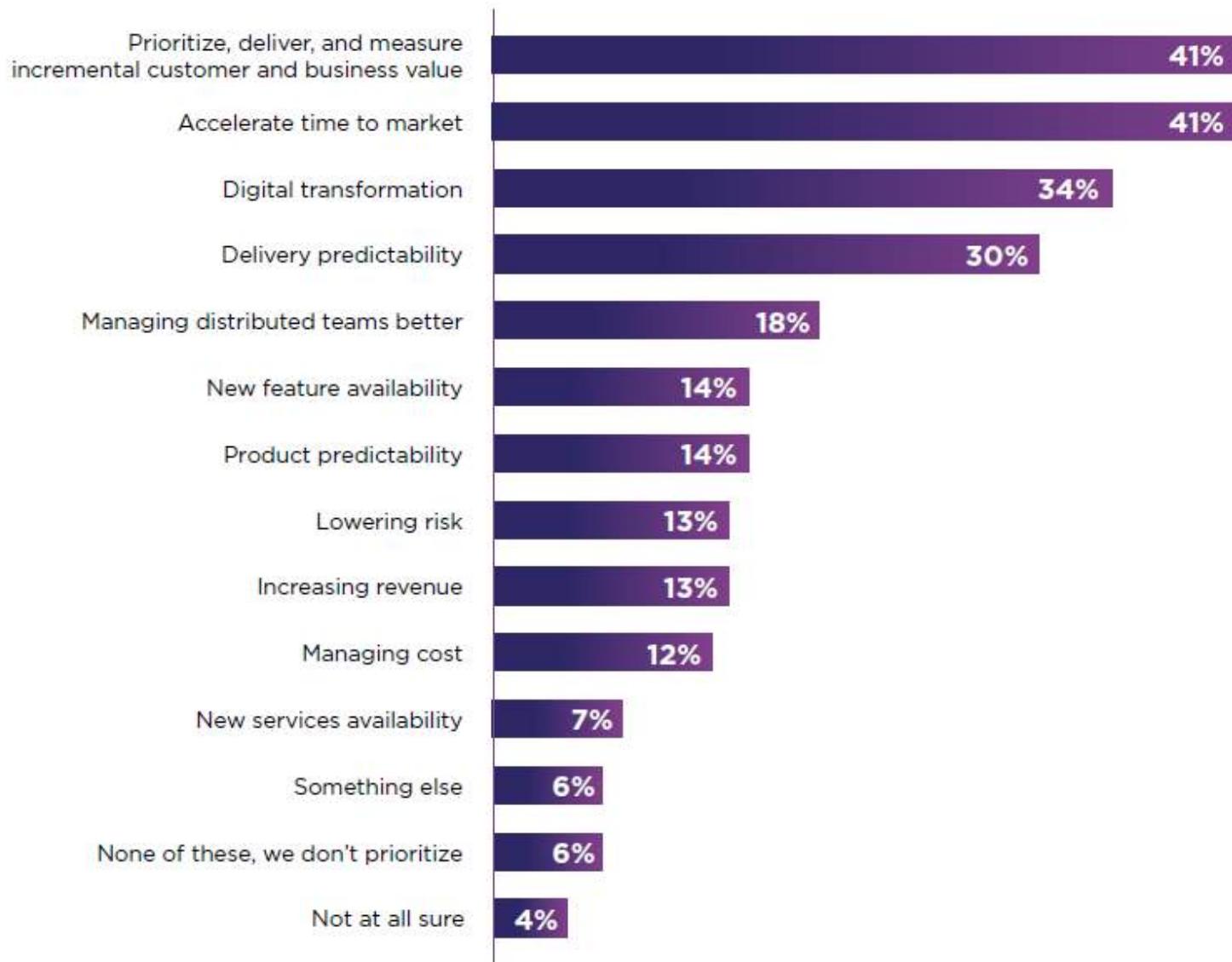
By Sanchita Sharma | Hindustan Times, New Delhi  
UPDATED ON APR 15, 2020 08:43 AM IST

**newsround**

**BTS: New song Butter drops and breaks YouTube records**

The new tune also set a record for the largest YouTube video premiere with 3.9 million people watching it at the same time. Butter was thought to have already beaten Dynamite's previous record by earning 10 million views in less than 15 minutes of its release. After just 6 hours it has been watched more than 50 million times!

# Why scale agile in practice?



# Agenda | Part I (10:45-11:30) and part II (11:45-12:30)

1.1

Why agile?

- VUCA
- Complexity

1.2

How to tackle?

- Iterative delivery
- Inspect & Adapt

1.3

Frameworks

Scrum, KanBan,  
ScrumBan, XP,  
Obeya...

1.4

Scaling Agile

Discussion

1.5

BREAK

optional Q&A

2.1

Why scale?

- Speed
- VUCA

2.2

How to scale?

- Principles
- Feedback & learning

2.3

Frameworks

SAFe,  
Scrum@Scale,  
Spotify, Less,  
Nexus...

2.4

Challenges

- dependencies
- no feedback/  
learning loop
- over-processing

2.5

Quiz

# Key principles on scaling agile:



1. In context: close to the customer/together with customers
2. Very short iterations – finishing and delivering very quickly
3. Continuous validation – what is customer value and what not?
4. Finishing and delivering the most valuable first
5. There is constantly a product that is really DONE (possibility to stop/release)
6. Cross-functional teams with all the necessary competencies present
7. Lots and intense human contact and interaction
8. Safe environment in which experimentation, discovery and learning is allowed
9. Visual management to coordinate the work

# Scaling agile in literature:



# Agenda | Part I (10:45-11:30) and part II (11:45-12:30)

**1.1**

Why agile?

- VUCA
- Complexity

**1.2**

How to tackle?

- Iterative delivery
- Inspect & Adapt

**1.3**

Frameworks

Scrum, KanBan,  
ScrumBan, XP,  
Obeya...

**1.4**

Scaling Agile

Discussion

**1.5**

BREAK

optional Q&A

**2.1**

Why scale?

- Speed
- VUCA

**2.2**

How to scale?

- Principles
- Feedback & learning

**2.3**

Frameworks

SAFe,  
Scrum@Scale,  
Spotify, Less,  
Nexus...

**2.4**

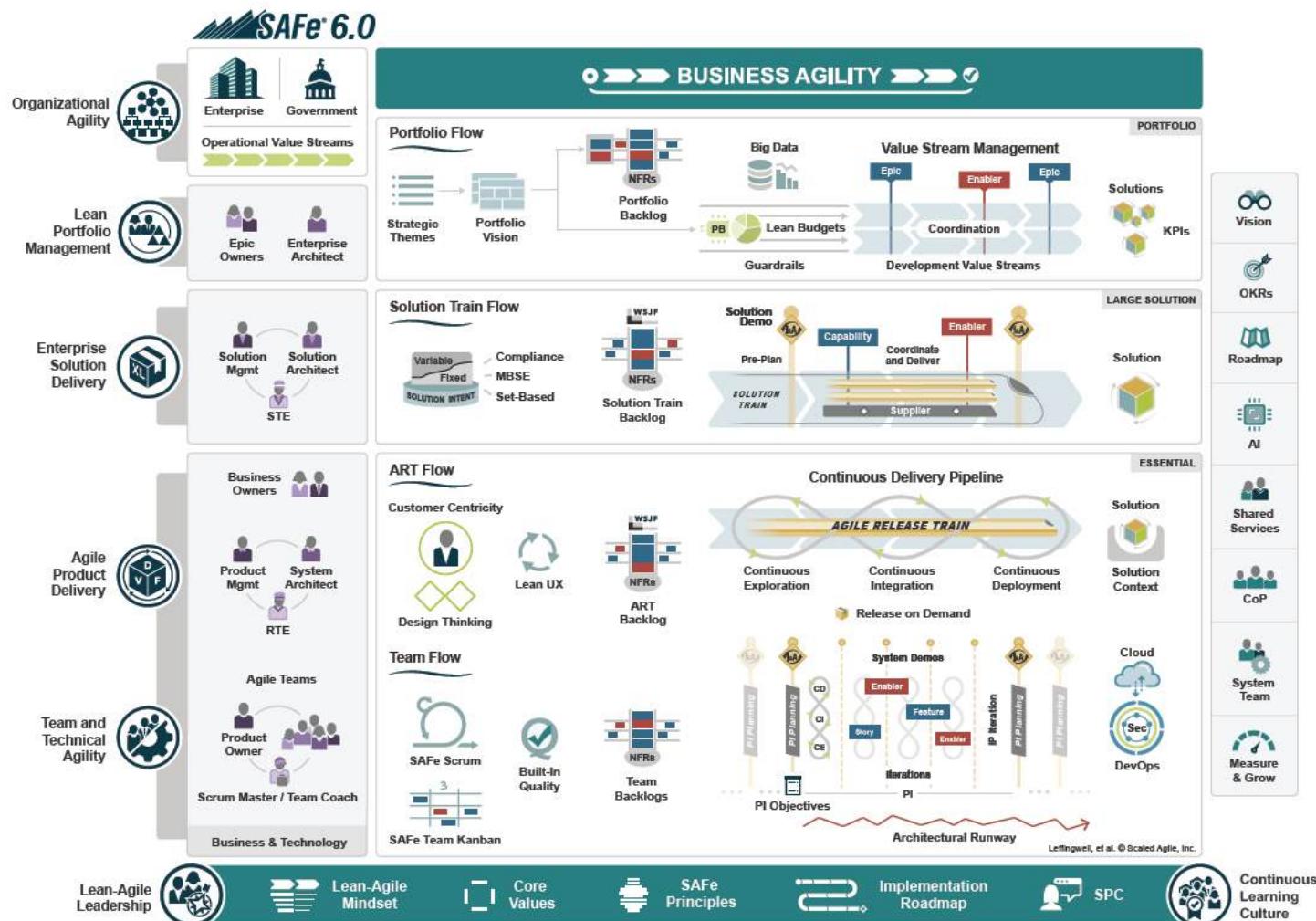
Challenges

- dependencies
- no feedback/  
learning loop
- over-processing

**2.5**

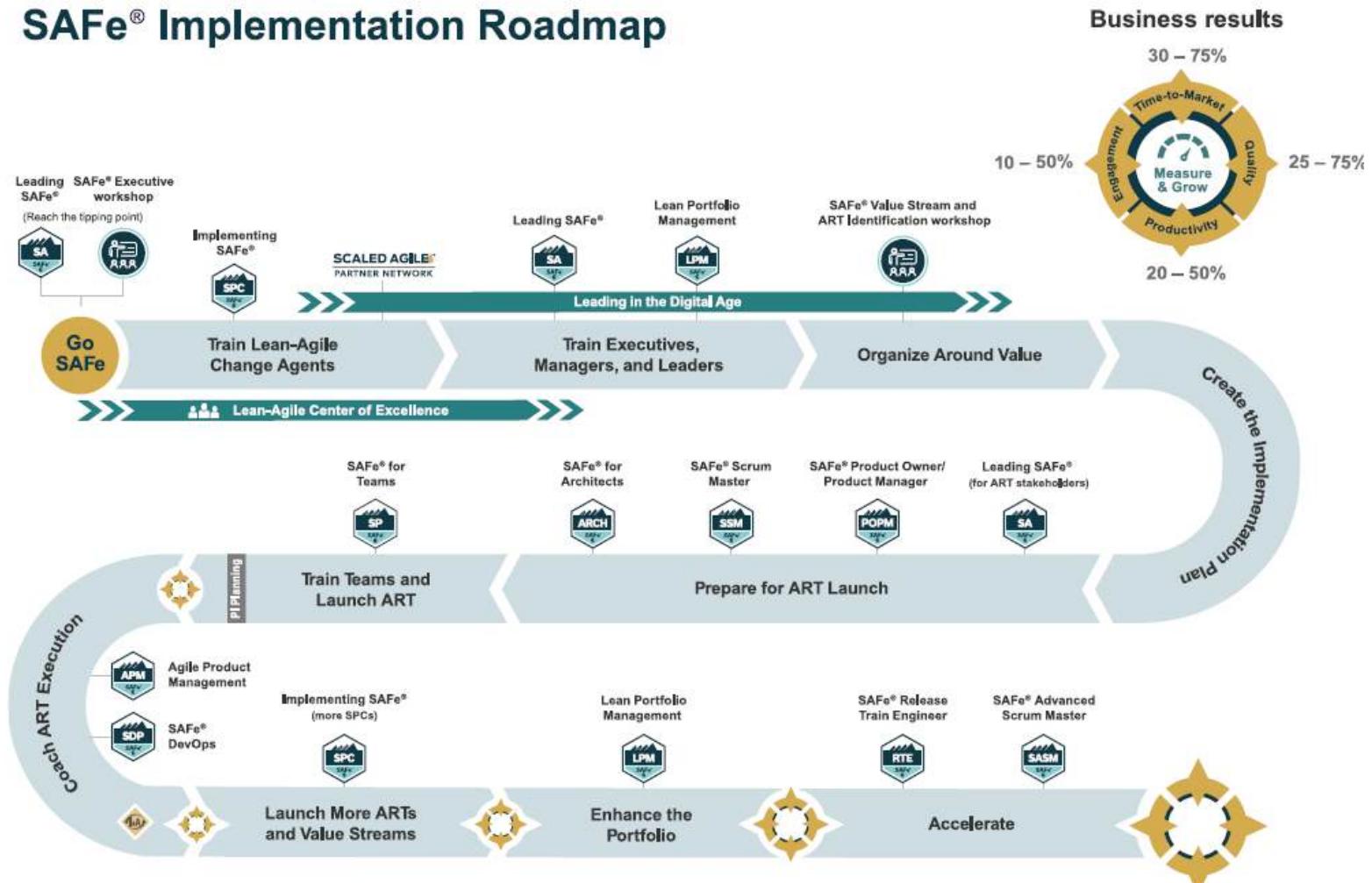
Quiz

# Scaled Agile Framework

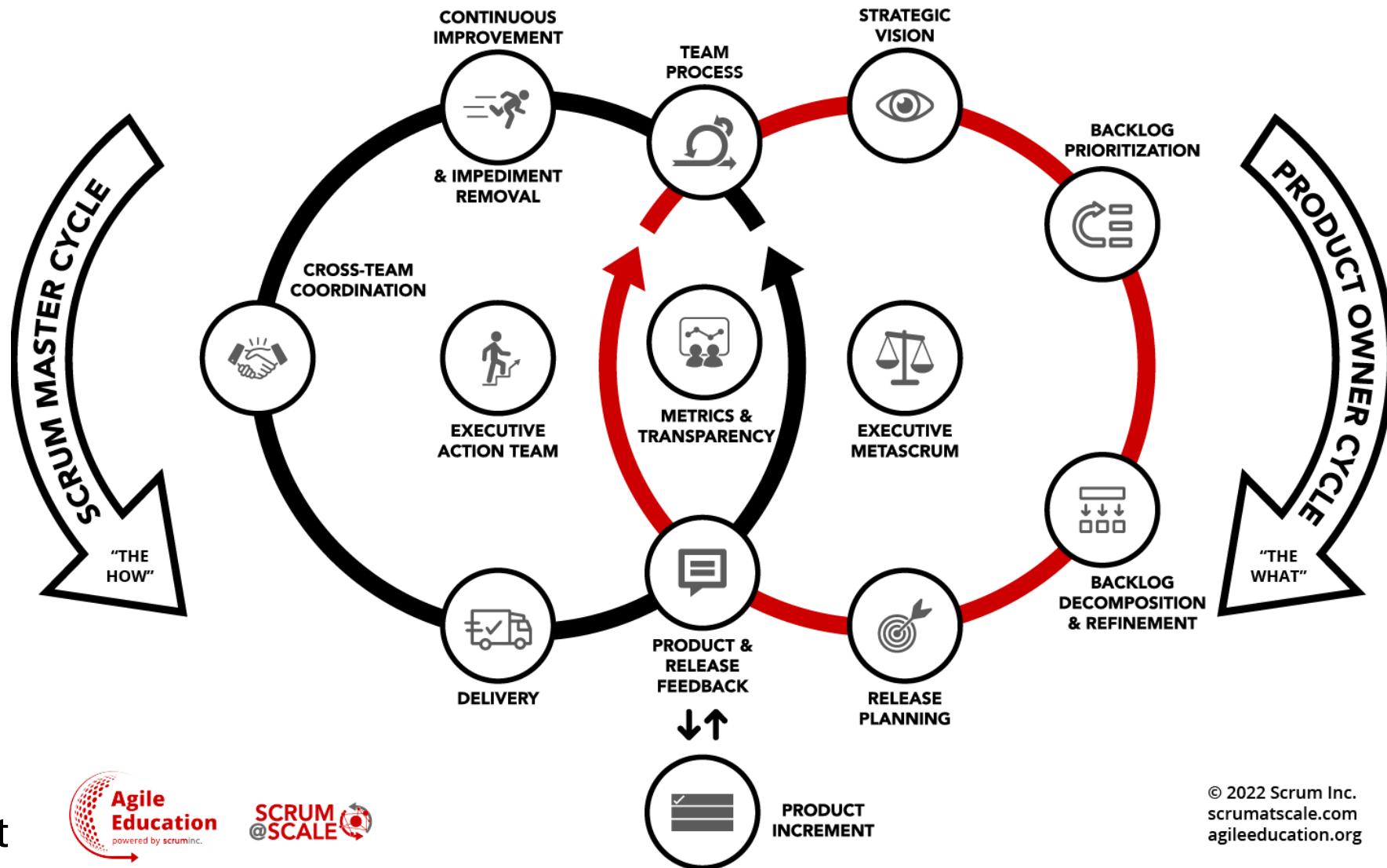


# Scaled Agile Framework

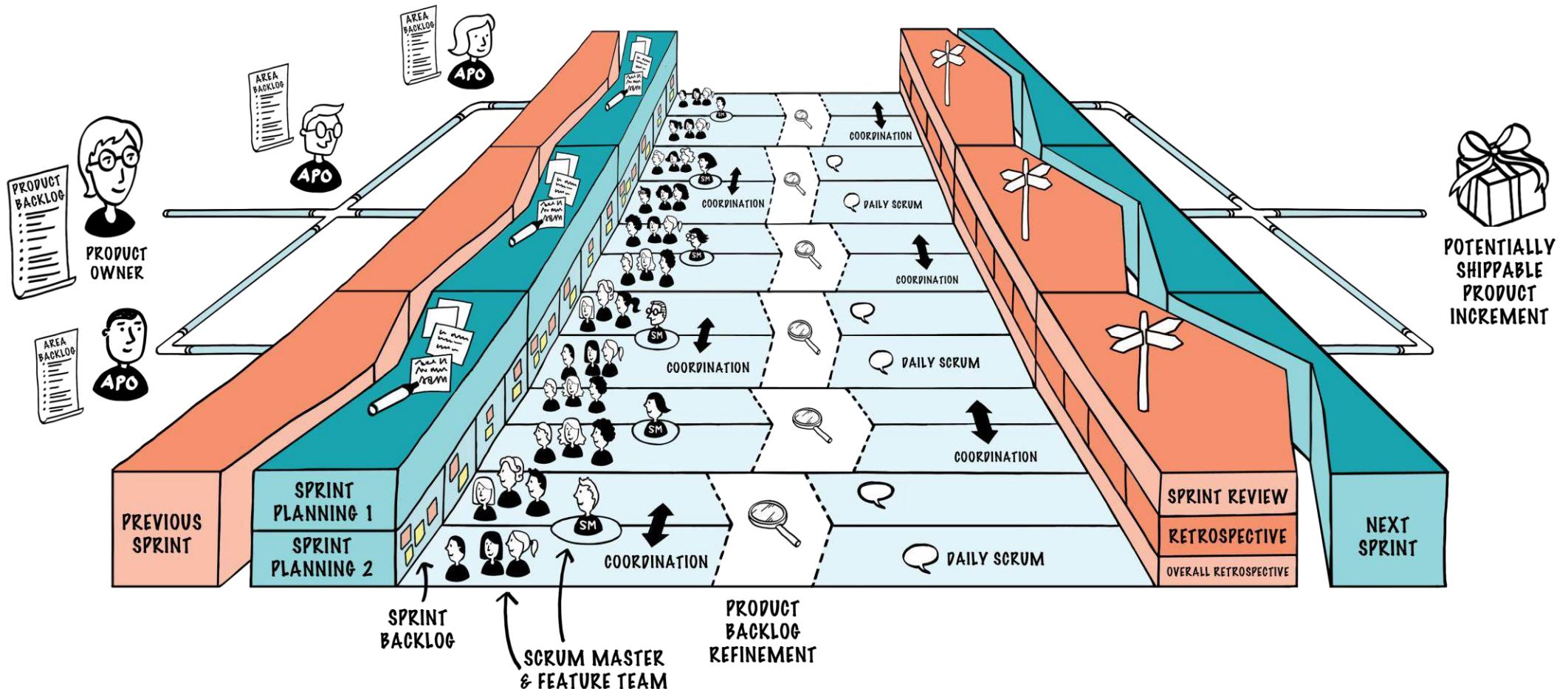
## SAFe® Implementation Roadmap



# Scrum@Scale Framework (Jeff Sutherland)

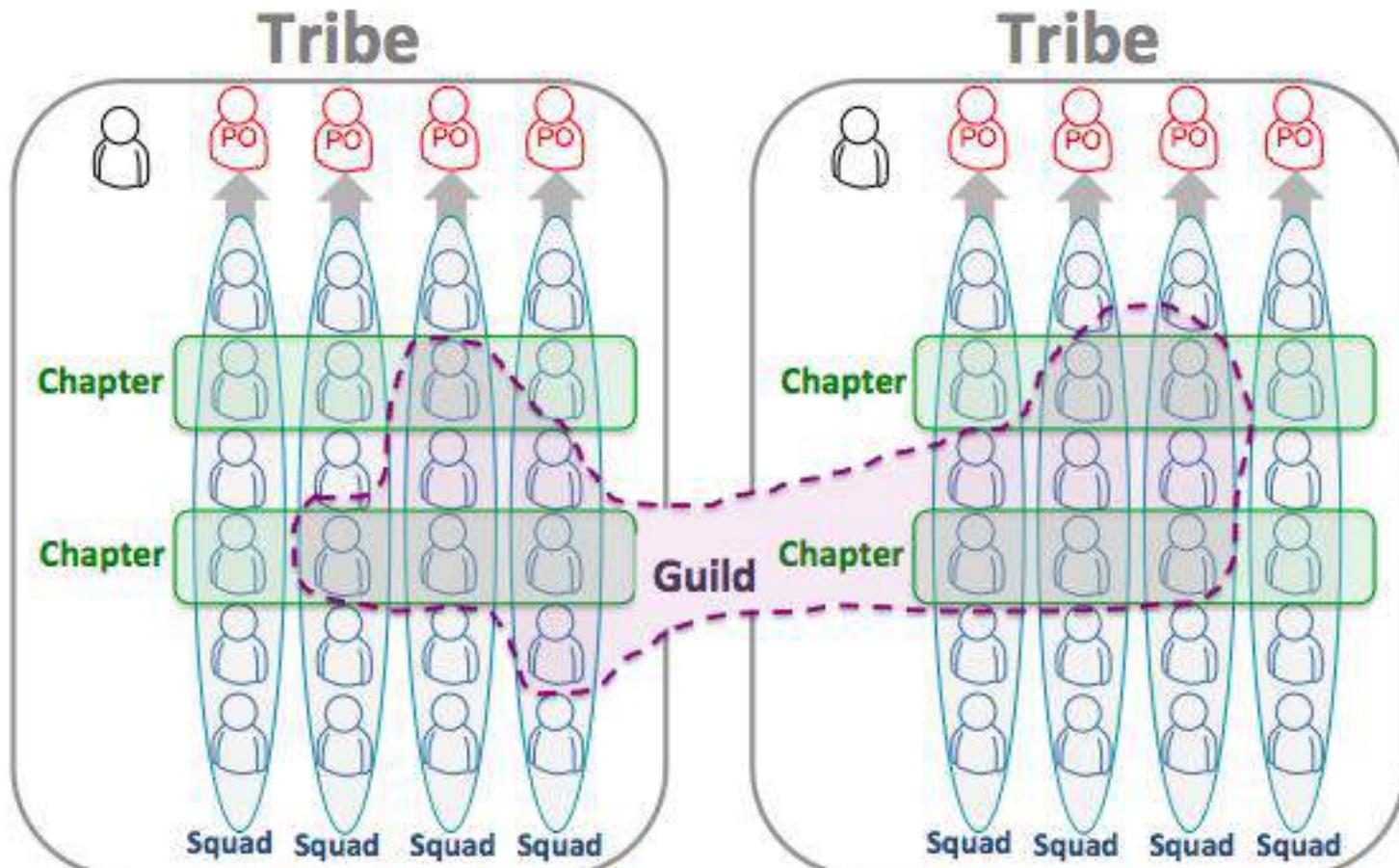


# Large Scale Scrum (LeSS)



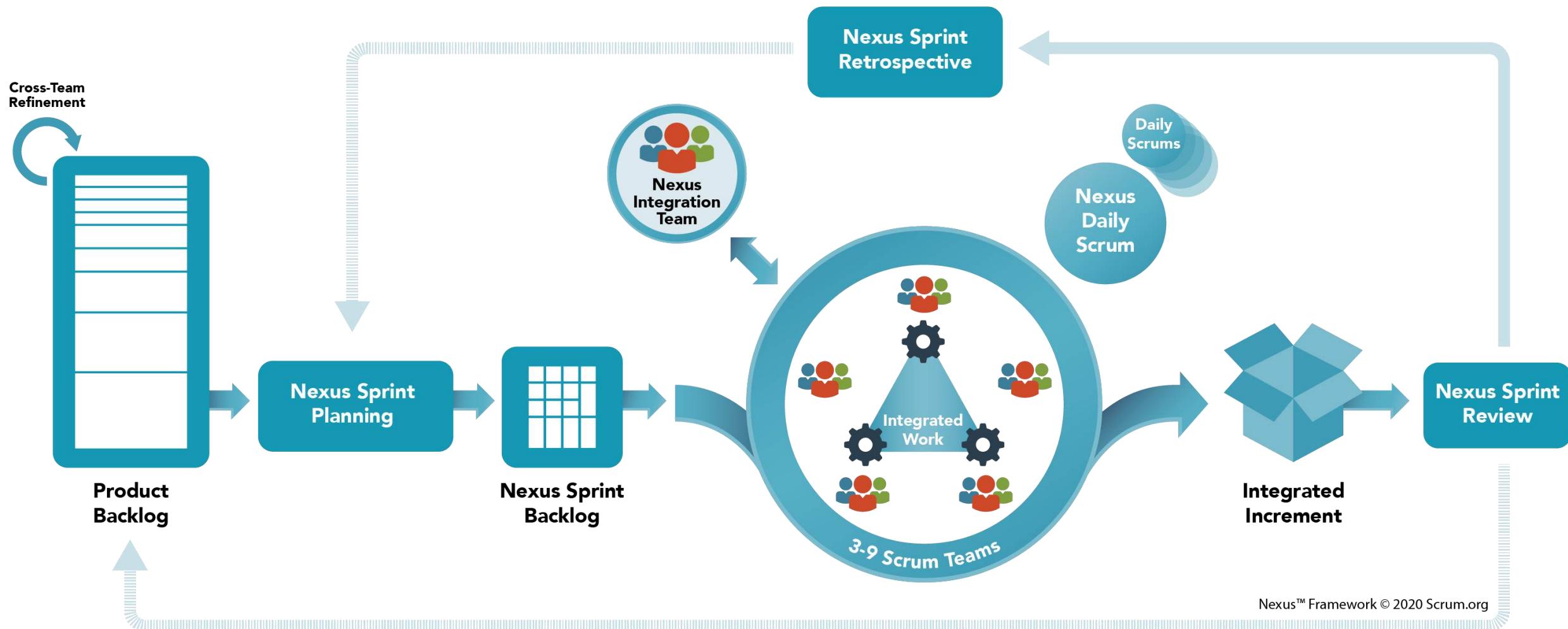
# The Spotify 'model'

- Squad
  - Equivalent to a Scrum team
  - Autonomous as possible
- Tribes
  - Same office < 100 people
  - Common area of the system
  - Organised for minimum interdependency
- Chapter
  - Skills community
  - Chapter Lead is line manager
- Guilds
  - Community of interest
  - Cross Tribe group
  - Guild Unconferences

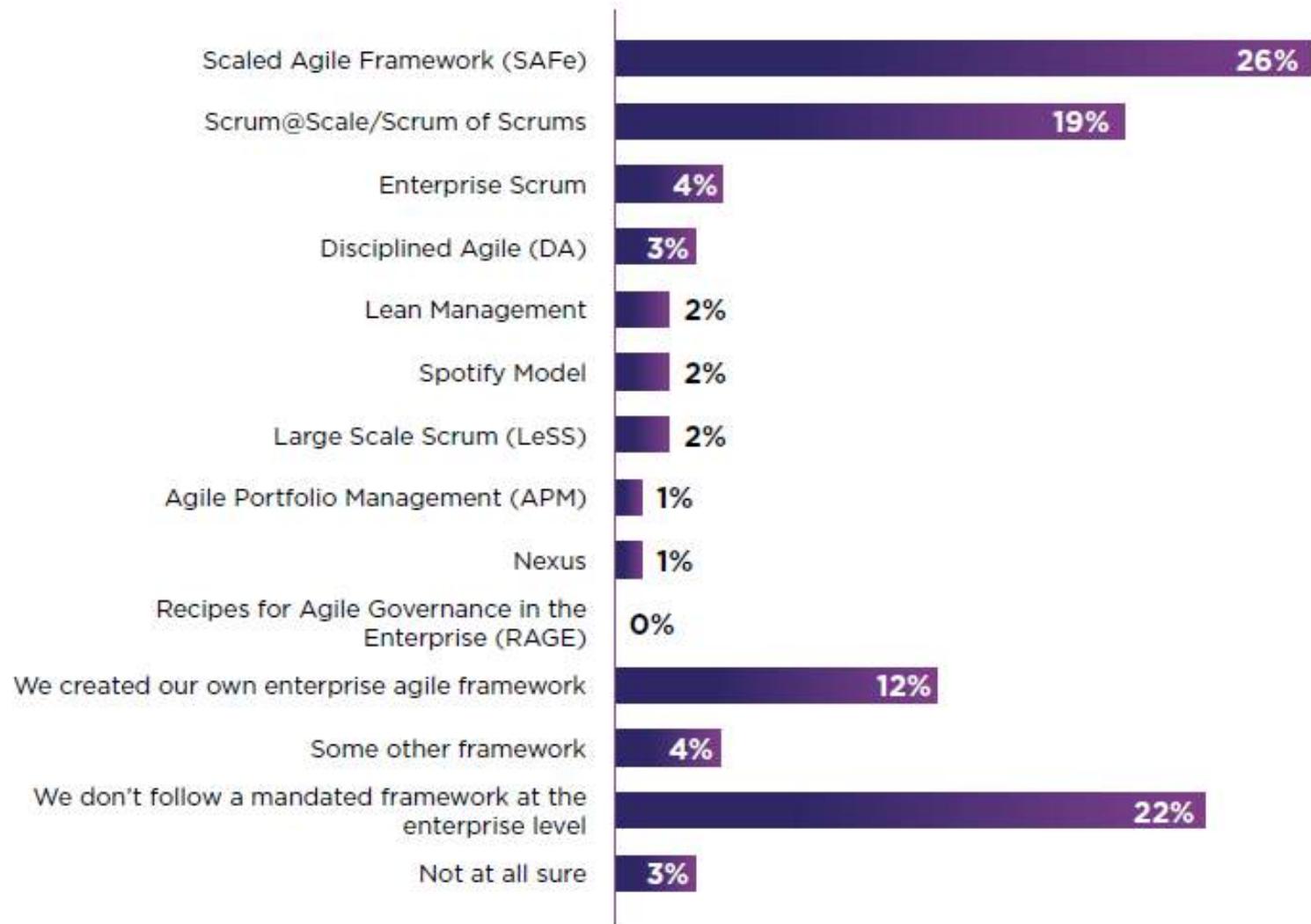


Source: Henrik Kniberg & Anders Ivarsson Oct 2012

# Nexus (Ken Schwaber)



# Scaling Frameworks in practice



# Agenda | Part I (10:45-11:30) and part II (11:45-12:30)

**1.1**

Why agile?

- VUCA
- Complexity

**1.2**

How to tackle?

- Iterative delivery
- Inspect & Adapt

**1.3**

Frameworks

Scrum, KanBan,  
ScrumBan, XP,  
Obeya...

**1.4**

Scaling Agile

Discussion

**1.5**

BREAK

optional Q&A

**2.1**

Why scale?

- Speed
- VUCA

**2.2**

How to scale?

- Principles
- Feedback & learning

**2.3**

Frameworks

SAFe,  
Scrum@Scale,  
Spotify, Less,  
Nexus...

**2.4**

Challenges

- dependencies
- no feedback/  
learning loop
- over-processing

**2.5**

Quiz

# Most common practical challenges

1. Too much interdependencies between teams (and underlying applications)  
Large refactoring needed – while doing new development in parallel
2. Unclear line of decision making (and feedback) from strategy to execution
3. Too long (or absent) feedback loops – very low frequent delivery  
In-team, inter-team, value stream, etc.
4. Overprocessing and management of bottlenecks (not taking them away)  
Framework implementation as a goal in itself
5. Management thinking the work is done after initial training/implementation  
The real implementation is in doing and improving – it is never done
6. Underestimating the required change in cultural and behavioral habits
7. ...and many many more.....

# Scaling agile challenges in literature:

The Journal of Systems and Software 119 (2016) 87–108

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: [www.elsevier.com/locate/jss](http://www.elsevier.com/locate/jss)

 CrossMark

**Challenges and success factors for large-scale agile transformations: A systematic literature review**

Kim Dikert<sup>a</sup>, Maria Paasivaara<sup>a,b</sup>, Casper Lassenius<sup>a,b,\*</sup>

<sup>a</sup> School of Science, Department of Computer Science and Engineering, Aalto University, Finland  
<sup>b</sup> Massachusetts Institute of Technology, Sloan School of Management, United States

**ARTICLE INFO**

**Article history:**  
Received 5 October 2015  
Revised 5 June 2016  
Accepted 6 June 2016  
Available online 7 June 2016

**Keywords:**  
Agile software development  
Organizational transformation  
Large-scale agile  
Adopting agile software development  
Challenges  
Success factors  
Systematic literature review

**ABSTRACT**

Agile methods have become an appealing alternative for companies striving to improve their performance, but the methods were originally designed for small and individual teams. This creates unique challenges when introducing agile at scale, where development teams must synchronize their activities, and there might be a need for interface with other organizational units. In this paper we present a systematic literature review on how agile can be scaled and what challenges exist at scale, focusing on reported challenges and success factors in the transformation. We conducted a systematic literature review of industrial large-scale agile transformations. Our keyword search found 1875 papers. We included 52 publications describing 42 industrial cases presenting the process of taking large-scale agile development into use. Almost 90% of the included papers were experience reports, indicating a lack of sound academic research on the topic. We identified 35 reported challenges grouped into nine categories, and 29 success factors, grouped into eleven categories. The most salient success factor categories were management support, choosing and customizing the agile model, training and coaching, and mindset and alignment.

© 2016 The Authors. Published by Elsevier Inc.  
This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

**1. Introduction**

The State of Agile Survey that Version One has been conducting annually since 2007, has recently asked a few questions related to large scale as well, e.g. on scaling methods used and tips for success with scaling agile. According to the latest survey (VersionOne, Inc., 2016), 62% of the almost 4000 respondents had more than a hundred people in their software organization and 43% of all the respondents worked in development organizations where more than half of the team were agile. Of course, the sample of this study is limited to a selected subset of companies and countries (of the almost 4000 respondents to the latest survey 65% were from North America and 26% from Europe). However, this indicates that there seems to exist a large number of companies that have taken or are taking agile into use in large-scale settings (VersionOne, Inc., 2016).

While the research literature contains several experience reports and some case studies on large-scale agile adoption, a systematic overview and synthesis of this growing body of research is still missing. Freudenberg and Sharp (2010) asked the industrial practitioners at the XP2010 conference to create a backlog of topics they thought should be studied. The practitioners voted "Agile and large projects" as the top burning research question. Moreover, among the top ten items three focused on distributed agile development, which is relevant especially for larger organizations as

\* Corresponding author.  
E-mail addresses: kim-karol.dikert@aalto.fi (K. Dikert), maria.paasivaara@aalto.fi (M. Paasivaara), casper.lassenius@aalto.fi (C. Lassenius).

<http://dx.doi.org/10.1016/j.jss.2016.06.013>  
© 2016 The Authors. Published by Elsevier Inc. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

Received 13 September 2017; | Revised 14 March 2018; | Accepted 15 March 2018  
Accepted version of the article. Final version at <https://doi.org/10.1002/smcr.1954>

**PRACTICE**

**Scaling Agile in Large Organizations: Practices, Challenges and Success Factors**

Martin Kalenda<sup>1</sup> | Petr Hyna<sup>2</sup> | Bruno Rossi\*<sup>1</sup>

<sup>1</sup> Faculty of Informatics, Masaryk University, Brno, Czech Republic  
<sup>2</sup> Kenticco, Brno, Czech Republic

**Correspondence:**  
\*Corresponding author. Email: brossi@mail.muni.cz

**Summary**

Context: Agile software development has nowadays reached wide adoption. However, moving agile to large scale contexts is a complex task with many challenges involved. Objective: In this paper, we review practices, challenges and success factors for scaling agile both from literature and within a large software company, identifying the most critical factors. Method: we conduct a focused literature review to map the importance of scaling practices, challenges and success factors. The outcome of this focused literature review is used to guide action research within a software company with a view to scaling agile processes. Results: company culture, prior agile and lean experience, management support and value unification were found to be key success factors during the action research process. Resistance to change, an overly aggressive roll-out timeframe, quality assurance concerns, and integration into pre-existing non-agile business processes were found to be the critical challenges in the scaling process. Conclusion: the action research process allowed to cross-fertilize ideas from literature to the company's context. Scaling agile within an organization does not need to follow a specific schema, rather the process can be tailored to the needs while keeping the core values and principles of agile methodologies.

**KEYWORDS:**  
Large-scale Agile, Agile Adoption, Large Scale Scrum (LeSS), Scaled Agile Framework (SAFe), Action Research

**1 | INTRODUCTION**

Agile software development is very popular nowadays, as its application in many different contexts shows.<sup>[1]</sup> However, traditional agile methods were designed with a single team in mind and were not meant to face scalability challenges.<sup>[2]</sup> In this sense, "scaling" can be described as "a continuous process of transferring, translating and transforming knowledge across various communities and individuals" (Rollen,<sup>[3]</sup> citing Carlile<sup>[4]</sup>) – placing large emphasis on the communication needs during a scaling process.

Scaling agile software development in large organizations is complex and poses several challenges.<sup>[5][6]</sup> Large projects require appropriate coordination and communication between teams,<sup>[7]</sup> dependencies

between teams need to be managed, other non-agile units need to be involved<sup>[8]</sup> and the right people need to be part of the process.<sup>[9][10]</sup> Recent research reports that the majority of goals and practices for scaling agile are domain independent, listing key factors/challenges in coordinating multiple teams, difficulties with managing requirements, problems in adaptation with the organizational structure and issues in understanding agile concepts along the value chain.<sup>[11]</sup> Additionally, customer involvement, software architectural concerns and inter-team coordination were also reported.<sup>[12]</sup> Together with challenges in coordinating the work of several teams.<sup>[13]</sup>

In general, training personnel, informing and engaging people in the process, and involving actors that can push the process further were general success factors found in case studies related to agile scalability.<sup>[14]</sup> Adopting new scaling practices is also challenging, as there are many cross-cutting factors that can impact and have been found in technology adoption studies.<sup>[15][16]</sup> In particular, resistance to change and weak management support can play a relevant role.<sup>[17][18]</sup> As it has been observed,

**Abbreviations:** LeSS, Large Scale Scrum; SAFe, Scaled Agile Framework; CoP, Communities of Practice; SoS, Scrum of Scrums; CTO, Chief Technology Officer; CEO, Chief Executive Officer; UX, User Experience; QA, Quality Assurance; CMS, Content Management System

# Agenda | Part I (10:45-11:30) and part II (11:45-12:30)

**1.1**

Why agile?

- VUCA
- Complexity

**1.2**

How to tackle?

- Iterative delivery
- Inspect & Adapt

**1.3**

Frameworks

Scrum, KanBan,  
ScrumBan, XP,  
Obeya...

**1.4**

Scaling Agile

Discussion

**1.5**

BREAK

optional Q&A

**2.1**

Why scale?

- Speed
- VUCA

**2.2**

How to scale?

- Principles
- Feedback & learning

**2.3**

Frameworks

SAFe,  
Scrum@Scale,  
Spotify, Less,  
Nexus...

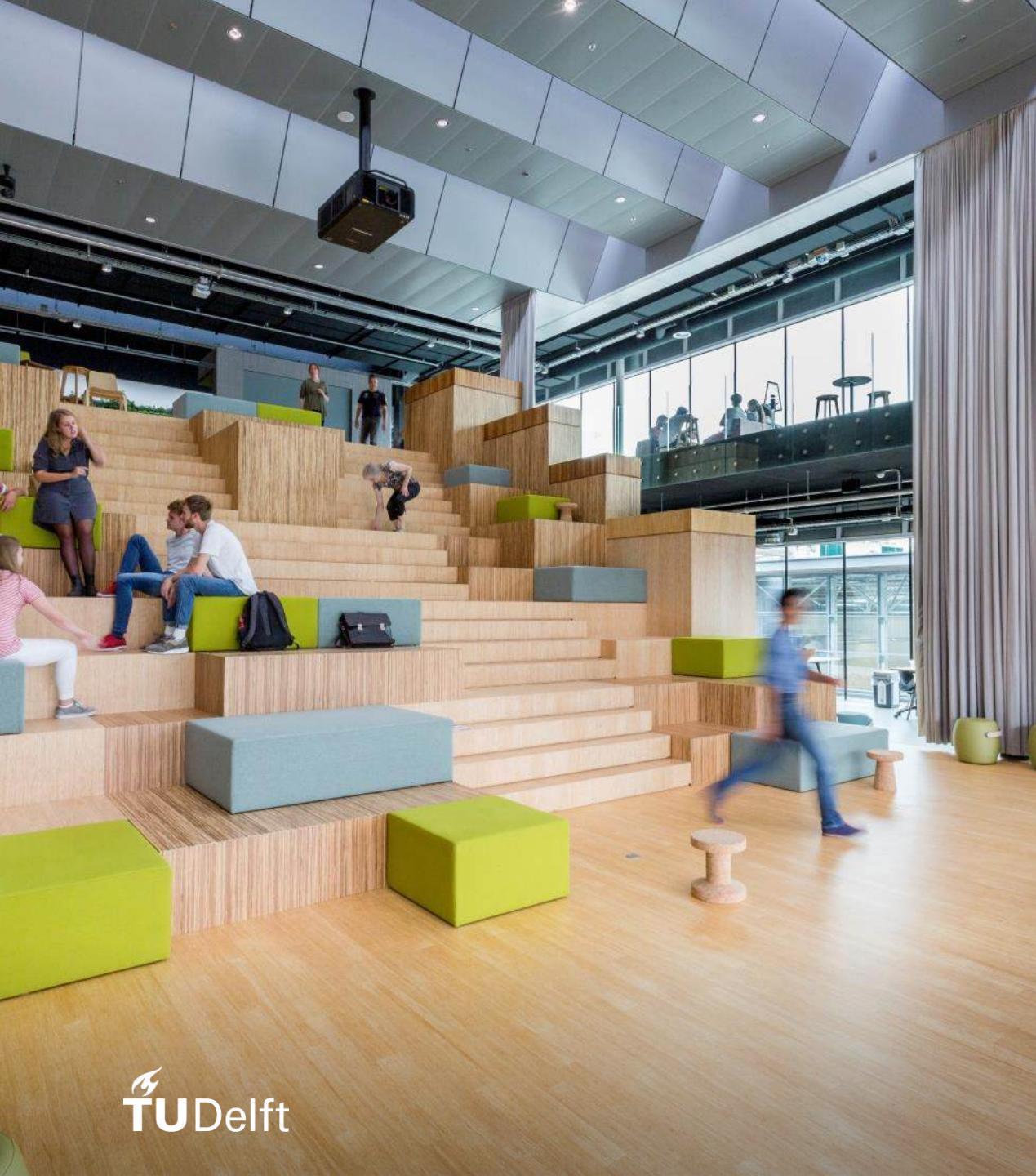
**2.4**

Challenges

- dependencies
- no feedback/  
learning loop
- over-processing

**2.5**

Quiz



# Exercise: Scrum Quiz

## Table of Contents

- Purpose of the Scrum Guide
- Scrum Definition
- Scrum Theory
  - Transparency
  - Inspection
  - Adaptation
- Scrum Values
- Scrum Team
  - Developers
  - Product Owner
  - Scrum Master
- Scrum Events
  - The Sprint
  - Sprint Planning
  - Daily Scrum
  - Sprint Review
  - Sprint Retrospective
- Scrum Artifacts
  - Product Backlog
  - Sprint Backlog
  - Increment

# The 2020 Scrum Guide™

This HTML version of the Scrum Guide is a direct port of the November 2020 version available [as a PDF here](#).

## Purpose of the Scrum Guide

We developed Scrum in the early 1990s. We wrote the first version of the Scrum Guide in 2010 to help people worldwide understand Scrum. We have evolved the Guide since then through small, functional updates. Together, we stand behind it.

The Scrum Guide contains the definition of Scrum. Each element of the framework serves a specific purpose that is essential to the overall value and results realized with Scrum. Changing the core design or ideas of Scrum, leaving out elements, or not following the rules of Scrum, covers up problems and limits the benefits of Scrum, potentially even rendering it useless.

We follow the growing use of Scrum within an ever-growing complex world. We are humbled to see Scrum being adopted in many domains holding essentially complex work, beyond software product development where Scrum has its roots. As Scrum's use spreads, developers, researchers, analysts, scientists, and other specialists do the work. We use the word "developers" in Scrum not to exclude, but to simplify. If you get value from Scrum, consider yourself included.

As Scrum is being used, patterns, processes, and insights that fit the Scrum framework as described in this document, may be found, applied and devised. Their description is beyond the purpose of the Scrum Guide because they are context sensitive and differ widely between Scrum uses. Such tactics for using within the Scrum framework vary widely and are described elsewhere.

## Scrum Definition

Scrum is a lightweight framework that helps people, teams and organizations generate value through adaptive solutions for complex



# Closure

# Wrapping up: learning objectives

1. Understand why iterative and incremental development is a solution for VUCA settings
2. Understand that iterative and incremental development is about high-frequency delivery, continuous (and thus automated) verification and validation, and continuous improvement
3. Know the most important agile approaches for teams (Scrum, KanBan, ScrumBan, eXtreme Programming and Obeya)
4. Knowing what the most important prerequisites are for large-scale agile and the main barriers to large-scale agile
5. Know key agile scaling frameworks (SAFe, Spotify, LESS, Scrum@Scale)
6. Be able to reason about strengths and weaknesses of Scrum and SAFe

# Thank you very much!

Professional agility at scale | prof.dr.ir. Rini van Solingen

26-9-2024

# Guest Lecture

Bol.com

**bol.**



# Martijn Steenbergen

Senior Software Engineer @ bol

## About me

Frontend Engineer in the webshop

- Live in Utrecht
- Automation Nerd
- Love the color red
- Needs to update his profile picture

# About Bol

## Some facts

Started in **1999**

Active in **Netherlands and Belgium**

**38 million products** on our website

**50.000 sellers** on our platform

**13.5 million** active customers

**200+** teams

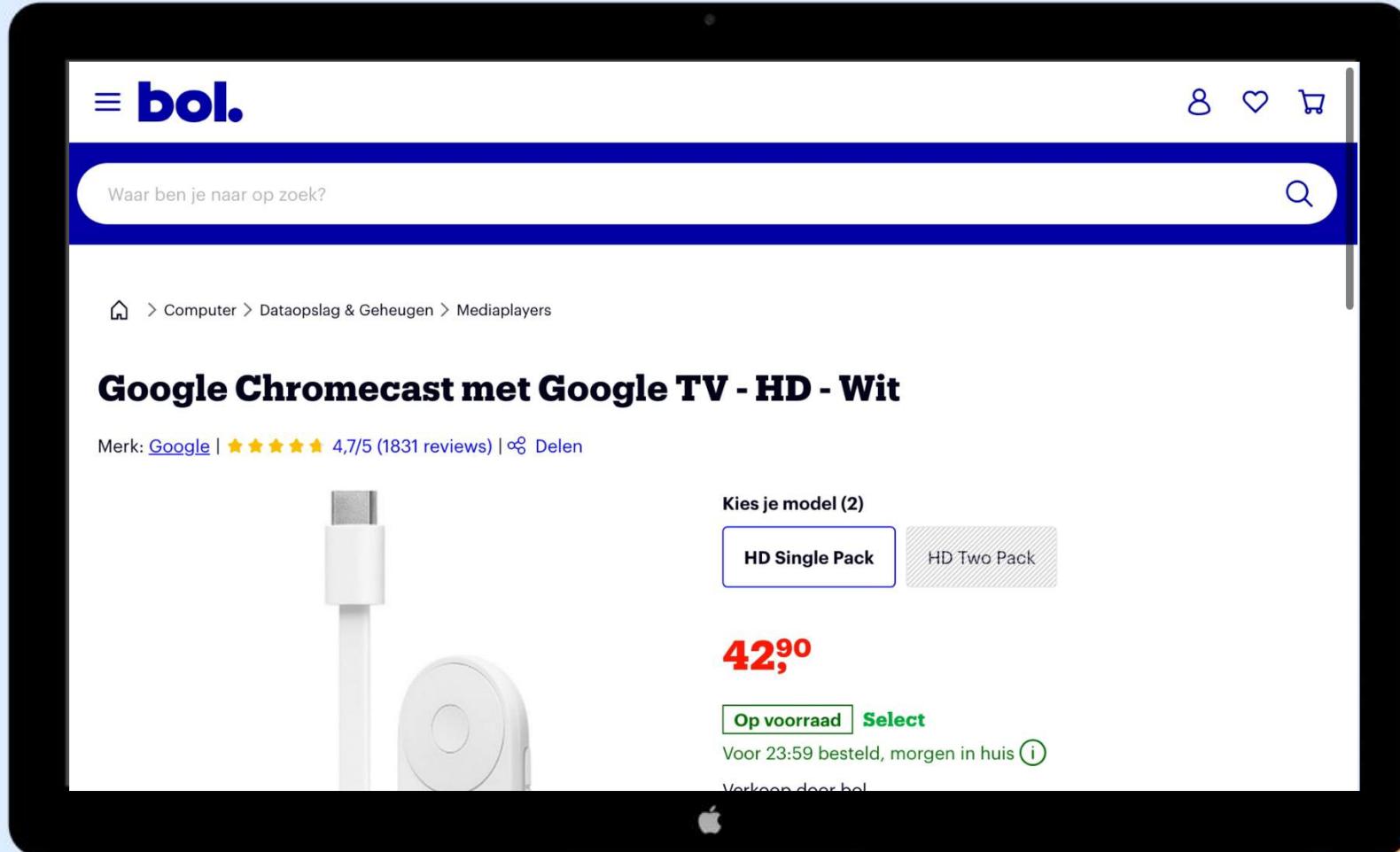
**1500+** microservices



# The Webshop

- 2023
  - 11596 commits
  - 129 people added code
- Release every morning
- Oldest line of code by someone who hasn't worked for Bol for 8 years
- Number of files:
  - Java: 6000+
  - Typescript: 4000+
  - JavaScript: 875
  - Sass: 700+

# The Product Page



# A project from beginning to end

Chapter One



# Step 1: The request



The screenshot shows a news article from the European Council's website. The header includes the European Council logo and navigation links for Home, Press, and Press releases. The main title is "EU introduces new rules on transparency and targeting of political advertising". The text below the title states: "The Council today adopted a new regulation on the transparency and targeting of political advertising, aimed at countering information manipulation and foreign interference in elections." A second paragraph explains: "The regulation will make it easy for citizens to recognise political advertisements, understand who is behind them and know whether they have received a targeted advertisement, so that they are better placed to make informed choices. It will also ensure that political advertising takes place in full respect of the right to privacy and that the freedom of opinion and freedom of speech are protected." A section titled "Main elements of the new regulation" is present, followed by a detailed description of the regulation's scope and a note about editorial responsibility.

Home > Press > Press releases

Council of the EU | Press release | 11 March 2024 14:55

## EU introduces new rules on transparency and targeting of political advertising

The Council today adopted a new regulation on the transparency and targeting of political advertising, aimed at countering information manipulation and foreign interference in elections.

The regulation will make it easy for citizens to recognise political advertisements, understand who is behind them and know whether they have received a targeted advertisement, so that they are better placed to make informed choices. It will also ensure that political advertising takes place in full respect of the right to privacy and that the freedom of opinion and freedom of speech are protected.

### Main elements of the new regulation

The new rules cover the transparency and targeting of political advertising in relation to an election, referendum, or a legislative process at EU level or in a member state. They do not affect the content of political advertisements nor other aspects of political advertising, such as the conduct of political campaigns, which remain subject to the specific national rules of member states.

Content under editorial responsibility, as well as views expressed in a personal capacity, are not covered.

According to the rules:

# Step 1: The request



VOTE WOOF  
For a better future!

\*PAID FOR BY *company that wants to influence this election*

# Step 1: What that means for us

Bekijk ook eens



Yaber T2 Beamer -  
Projector - Sound by JB...

369,95

Gesponsord



Nokia - Streaming Box -  
8000 - 4K Ultra HD - ...

54,90

Gesponsord



Ibley Afstandsbediening  
geschikt voor Google...

18,95

Gesponsord



Nokia - Streaming Box -  
8010 - 4K Ultra HD - ...

84,99

Gesponsord



123Beamer® A1 -  
Projector - Beamer - ( 8...

250,00

Gesponsord



HDMI Switch 4K 60Hz - 3  
ingangen 1 uitgang - ...

7,95

Gesponsord

# Step 1: What that means for us

What do we need to show:

1. Who paid for the advertisement
2. Who is advertising

Bekijk ook eens



Yaber T2 Beamer -  
Projector - Sound by JB...

369,95

Gesponsord



Nokia - Streaming Box -  
8000 - 4K Ultra HD - ...

54,90

Gesponsord



Ibley Afstandsbediening  
geschikt voor Google...

18,95

Gesponsord



Nokia - Streaming Box -  
8010 - 4K Ultra HD - ...

84,99

Gesponsord



123Beamer® A1 -  
Projector - Beamer - ( 8...

250,00

Gesponsord

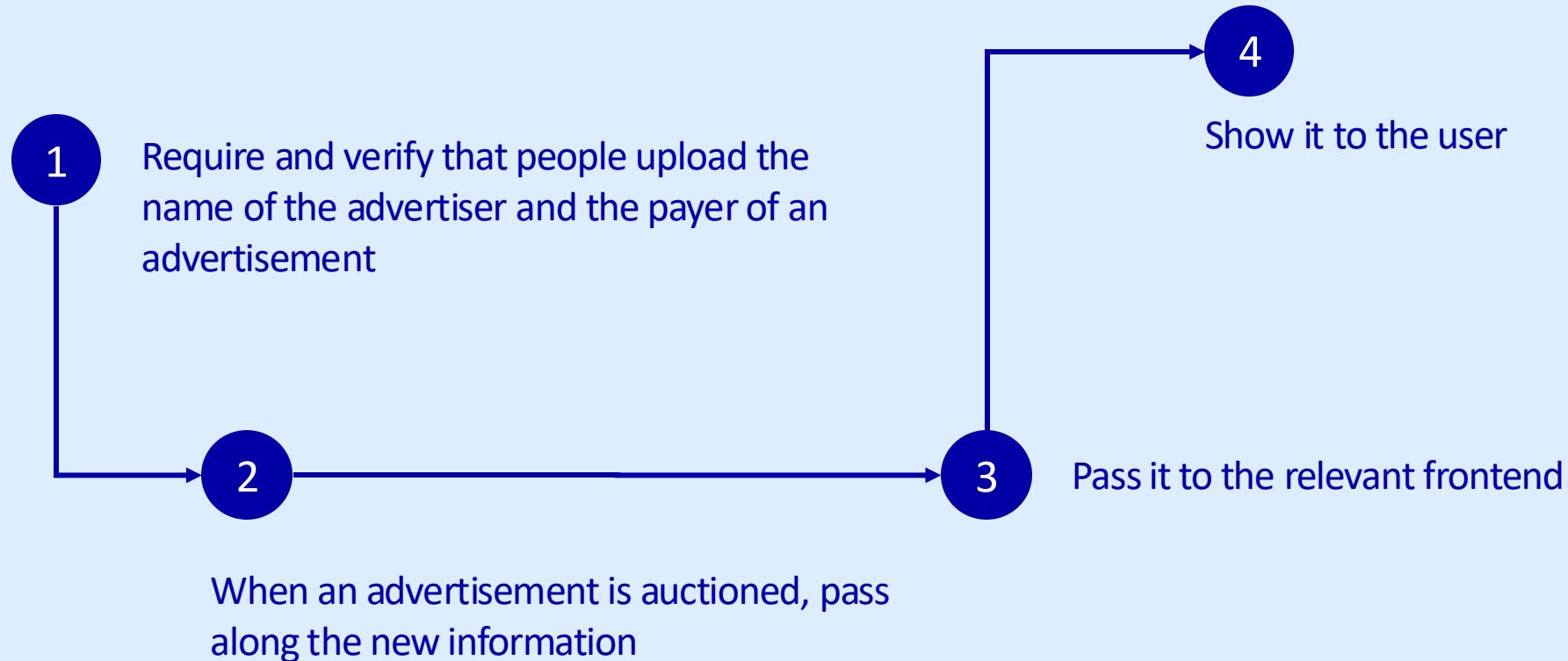


HDMI Switch 4K 60Hz - 3  
ingangen 1 uitgang - ...

7,95

Gesponsord

## Step 2: The plan



# Step 3: Preparation

## 1: Interaction measurement

Are there things we need to look out for?

Do we have any success metrics?

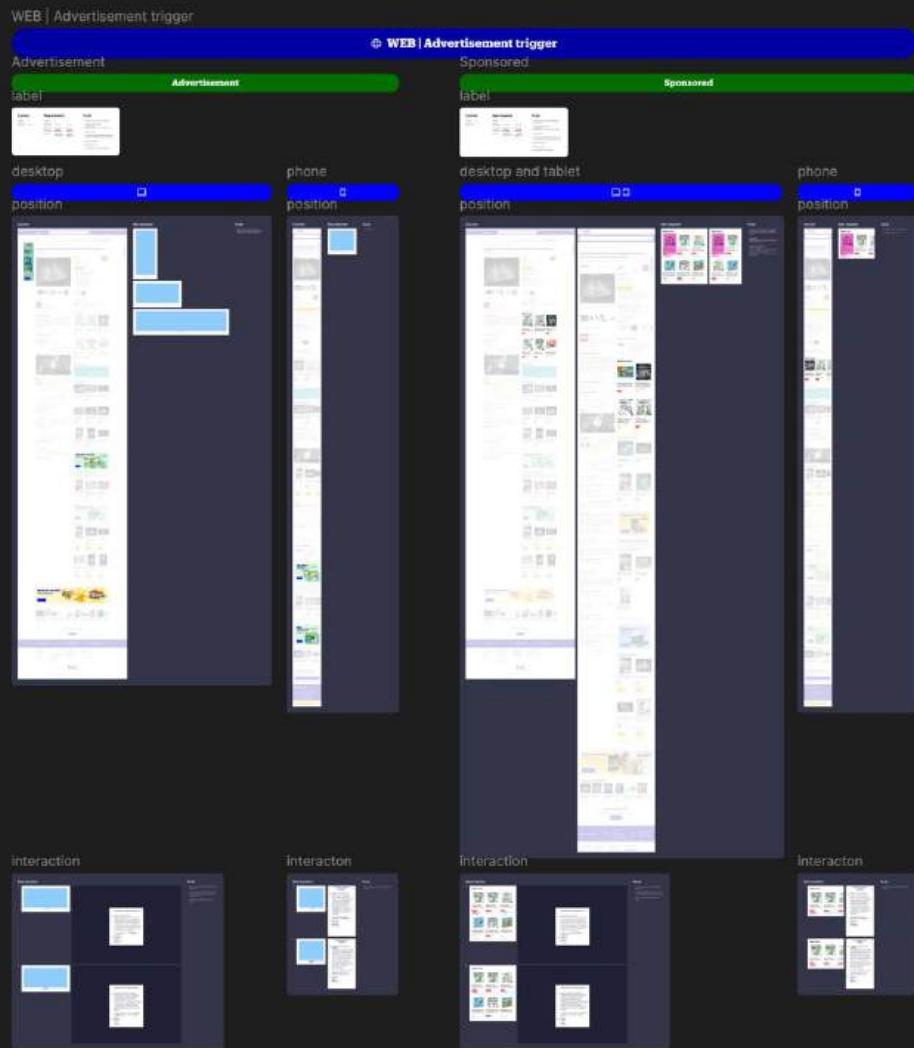
What are we testing?

What do we want to gain by implementing?

## 2: Design

How do we make sure it works best as possible for the user?

# The Design



# The Design

**New situation**

**To do**

- If user clicks on sponsored label open small sliding modal.
- Text and icon should be tag either as one button. Inactive the clickable area visible by making it gray.
- Sponsored modal should have both descriptive images.

**Informatie over deze advertentie**

Wat zijn de belangrijkste punten van deze advertentie?

We hebben de belangrijkste punten van deze advertentie voor u samengevat. De belangrijkste punten zijn: **Gravitrax Power Elements**. Als u ons hierover meer informatie wilt, kunt u de advertentie bekijken op de pagina waarop deze advertentie staat. We hebben ook een aantal tips voor u om deze advertentie te kunnen gebruiken. Wilt u meer informatie over hoe u deze advertentie kunt gebruiken? Dan kunt u de advertentie bekijken op de pagina waarop deze advertentie staat.

De voor meer informatie over deze advertentie kunt u contact opnemen met de auteur.

**Advertentie**

Gravitrax Power Elements

**Bekijk deze advertentie**

**Informations sur cette publicité**

Pourriez-vous nous dire ce que vous aimez dans cette publicité ?

Ces publicités peuvent être pour plusieurs raisons de manière expérimentale ou pour essayer une approche que possible. Celles-ci sont affichées sur la base du comportement précédent de nos clients et de leur intérêt pour les produits ou services correspondants. Nous voulons promouvoir des opportunités qui permettent et adaptées à nos clients. Tous ces types de publicités sont générées par nos algorithmes pour nous renseigner et le connaître ce que vous avez aimé et ce que vous n'avez pas aimé.

Pour plus d'informations, consultez notre politique de confidentialité et notre politique en matière de cookies.

**Advertentie**

Gravitrax Power Elements

**Bekijk deze advertentie**

# The Design

Informatie over deze advertentie X

**Waarom zie ik deze advertentie?**  
We laten je advertenties zien om je winkelervaring zo prettig mogelijk te maken. Ze worden getoond op basis van algemeen klantgedrag en statistieken. Als jij ons hier voor toestemming hebt gegeven, laten we je relevantere advertenties zien die zijn afgestemd op jouw interesses. Hierbij kijken we, afhankelijk van waar je toestemming voor hebt gegeven, bijvoorbeeld naar je zoekgedrag en de content die je bekeek, het land waar je woont en/of jouw bestellingen.

Zie voor meer informatie ons [privacybeleid](#) en [cookiestatement](#).

**Adverteerder**  
GraviTrax

**Betaald door**  
Ravensburger

- If user clicks on sponsored label open dialog modal.
- Text and icon should be together as one to make the clickable area visible by making the icon larger.
- Screen reader should skip both decorative images.

# Step 4: Lets build

# The (old) webshop in 60 seconds

## HTML TEMPLATES

```

1 <div class="reviews-summary-container {{#if isMobileVariant}}reviews-summary__mobile {{/if}}js-reviews-slot" !#bltgi#!>
2   <h2 class="reviews-summary__header">{{#if showTitle}}{{#if isMobileVariant}}{{rosettext.mobileHeader}}{{else}}{{rosettext.header}}{{/if}}
3   <div class="reviews-summary__flex">
4     <div class="reviews-summary__total-reviews" data-test="total-reviews">{{totalNumberOfReviewsLabel}}</div>
5   </div>
6   <div class="reviews-summary__avg-score" data-test="review-rating-average" aria-hidden="true">{{overallAverageRating}}</div>
7   <div class="reviews-summary__main-stars" role="img" aria-label="{{rosetta rosettext.stars "stars" overallAverageRating}}">
8     <div class="star-rating star-rating--large">
9       <span style="width: {{starWidthPercentage}}%"></span>
10    </div>
11  </div>
12  <div class="reviews-summary__divider"></div>
13  <div class="reviews-summary__cta">
14    <div class="u-hide@screen-medium-up">{{reviewsExplainer}}</div>
15    {{#if writeReviewAllowed}}
16      {{reviewsSummaryCta}}
17    {{/if}}
18  </div>
19  {{#each ratingDetails}}
20    <div class=reviews-summary__table-rating>
21      <div class="star-rating star-rating--clean" data-test="{{star}}-stars-rating"
22        style="max-width: {{star}}rem">
23          <span></span>
24        </div>
25    </div>
26    <div class="reviews-summary__table-bar{{#unless ../showTotalNumberLabels}} extra-column{{/unless}}">
27      <div class="reviews-summary__bar" data-test="rating-bar-{{star}}">
28        <div class="reviews-summary__bar-inner" style="width: {{barWidthPercentage}}%;"></div>
29      </div>
30    </div>
31    {{#if ../showTotalNumberLabels}}
32      <span class="reviews-summary__table-info" data-test="horizontal-bar-info-{{star}}" role="img" aria-label="{{rosetta rosettext.star "stars|people" star totalNumber}}">{{totalNumber}}</span>
33    {{/if}}
34  {{/each}}
35  <div class="reviews-summary__explainer u-hide@screen-small-only">{{reviewsExplainer}}</div>
36</div>

```



```

views-summary-container {{#if isMobileVariant}}reviews-summary__mobile {{/if}}js-reviews-s
"reviews-summary__header">>{{#if showTitle}}{{#if isMobileVariant}}{{rosettext.mobileHeader}}
="reviews-summary-flex">
lass="reviews-summary__total-reviews" data-test="total-reviews">>{{totalNumberOfReviewsLabel}}
```

```

="reviews-summary__avg-score" data-test="review-rating-average" aria-hidden="true">>{{overallAverageRating}}</div>
="reviews-summary__main-stars" role="img" aria-label="{{rosetta rosettext.stars "stars" overallAverageRating}}">
lass="star-rating star-rating--large">
span style="width: {{starWidthPercentage}}%"></span>
```

```

="reviews-summary__divider"></div>
="reviews-summary__cta">
ss="u-hide@screen-medium-up">>{{reviewsExplainer}}</div>
iteReviewAllowed}}
```

```

iewsSummaryCta}}
```

```

details}}
```

" data-test="{{star}}-stars-rating"


```
1 <div class="reviews-summary-container {{#if isMobileVariant}} reviews-summary__mobile {{/if}} js-reviews-stat :#dtlg>
2   <h2 class="reviews-summary__header">{{#if showTitle}}{{#if isMobileVariant}}
3   <div class="reviews-summary-flex">
4     <div class="reviews-summary__total-reviews" data-test="total-reviews">>
5       </div>
6     <div class="reviews-summary__avg-score" data-test="review-rating-average">
7       <div class="reviews-summary__main-stars" role="img" aria-label="{{rosettaScore}} stars">
8         <div class="star-rating star-rating--large">
9           <span style="width: {{starWidthPercentage}}%"></span>
10        </div>
11      </div>
12      <div class="reviews-summary__divider"></div>
13      <div class="reviews-summary__cta">
14        <div class="u-hide@screen:medium-up">{{reviewsExplainer}}</div>
15        {{#if writeReviewAllowed}}
16          {{reviewsSummaryCta}}
17        {{/if}}
18      </div>
19      {{#each ratingDetails}}
20        <div class="reviews-summary__table-rating">
21          <div class="star-rating star-rating--clean" data-test="{{star}}-stars-rating"
22            style="max-width: {{star}}rem">
23            <span></span>
24          </div>
25        </div>
26        <div class="reviews-summary__table-bar{{#unless ../showTotalNumberLabels}} extra-column{{/unless}}">
27          <div class="reviews-summary__bar" data-test="rating-bar-{{star}}">
28            <div class="reviews-summary__bar-inner" style="width: {{barWidthPc}}%">
```



```
12 <div class="reviews-summary__divider"></div>
13 <div class="reviews-summary__cta">
14   <div class="u-hide@screen-medium-up">{{reviewsExplainer}}</div>
15   {{#if writeReviewAllowed}}
16     {{reviewsSummaryCta}}
17   {{/if}}
18 </div>
19 {{#each ratingDetails}}
20   <div class=reviews-summary__table-rating>
21     <div class="star-rating star-rating--clean" data-test="{{star}}-stars"
22       style="max-width: {{star}}rem">
23       <span></span>
24     </div>
25   </div>
26   <div class="reviews-summary__table-bar{{#unless ../showTotalNumberLabels}} extra-column{{/unless}}">
27     <div class="reviews-summary__bar" data-test="rating-bar-{{star}}">
28       <div class="reviews-summary__bar-inner" style="width: {{barWidthPercentage}}%;"></div>
29     </div>
30   </div>
31   {{#if ../showTotalNumberLabels}}
32     <span class="reviews-summary__table-info" data-test="horizontal-bar-info-{{star}}" role="img" aria-label="{{rose}}"/>
33   {{/if}}
34 {{/each}}
35   <div class="reviews-summary__explainer u-hide@screen-small-only">{{reviewsExplainer}}</div>
36 </div>
```

**Reviews**

4,7 ★★★★★ Gemiddelde van 1836 reviews

★★★★★ 1539  
★★★★ 186  
★★★ 36  
★★ 20  
★ 55

▼ Hoe controleren en plaatsen wij reviews?

Sorteer op: Meest behulpzame ▾



# The (old) webshop in 60 seconds

ISLANDS

# The (old) project

Gratis verzending vanaf 25,- Bezorging dezelfde dag, 's avonds in het weekend\* Gratis retourneren

Select Ontdek nu de 4 voordeel Select-Deals >

Categorieën ▾ Cadeaus & Inspiratie ▾ Aanbiedingen ▾ Zakelijk Cadeaukaart Bestelstatus Klantenservice NL ▾

Soepele start De nieuwste mode Oktoberfest Voorraaddeals Select-Deals >

< Terug Elektronica > Audio & Hifi > Audio voor jou > Oordopjes

**evanell® E24EB Pro - Bluetooth Oordopjes Draadloos, Draadloze In-Ear Oortjes V5.3, 42u Batterij, Microfoon, Touch, iOS/Android, 2024**

Merk: Evanell | ★★★★ 4,5/5 (46 reviews) | ⚡ Delen

**14,99** korting

Meest getoonde prijs 21,99 Je bespaart 32%

Op voorraad Select Voor 23:59 besteld, morgen in huis ⓘ Verkoopt door Evanell 7,8

In winkelwagen Winkelwagen

- ✓ Bestellen en betalen via bol
- ✓ Gratis verzending door bol vanaf 25 euro
- ✓ 30 dagen bedenktijd en gratis retourneren
- ✓ Wettelijke garantie via Evanell
- ✓ Dag en nacht klantenservice

Bekijk de extra [voorwaarden](#) van deze partner.

**Bezorgopties**

- ✓ **Vandaag** nog in huis (bestel ma-vr voor 12:00, bezorging tussen 17:00 en 22:00)
- ✓ **Dooreweeks ook 's avonds** in huis
- ✓ **Ook zondag** in huis (bestel voor za 23:59)

[Bekijk alle bezorgopties](#)

**Vergelijk met andere artikelen**

**In-ear | Draadloos | Met microfoon** | Geschikt voor: Bellen/Werken, Dagelijks gebruik, Gaming, Reizen, Sporten | Tot 42 uur

[Alle productspecificaties](#)

**Productbeschrijving**

**"Fijne bluetooth oordopjes prima geluid"**

Nieuwste Bluetooth versie 5.3 ✓ Blijven altijd zitten! ✓ Zweet- en regenbestendig ✓ EXTRA Bass ✓ Bel Handsfree ✓ Nederlands merk ✓ Tot 8 uur op een laading, 42 uur totaal met case

Stel je voor: je begint je dag met een glimlach, klaar om elke uitdaging aan te gaan. Je pakt je Evanell oordopjes en met een simpele aanraking zijn ze direct verbonden met je telefoon. De diepe bas en heldere tonen van je favoriete muziek vullen je oren terwijl je de vooravond opmaakt. Ze zijn niet alleen

Toon meer

**Productspecificaties**

Zoeken naar kenmerken?

**Algemene kenmerken**

Merk	Evanell
Model	EB24 Pro
EAN	770618409607

Toon meer

**Video's**

**evanell® E24EB PRO Draadloze oordopjes**

**Anderen bekeken ook**

Draadloze Oordopjes - Roze Goud - Bluetooth ...  
★★★★ (219)  
19,99  
Gesponsord

Draadloze Oordopjes - Blauw - Bluetooth - Geschikt vo...  
★★★★ (219)  
19,99  
Gesponsord

PrimeSound Sky Pro Draadloze Oordopjes - ...  
★★★★ (299)  
39,99  
Gesponsord

A Project | Step 4

Dialog Model

For ally reasons a dialog always contains a description.

Random button

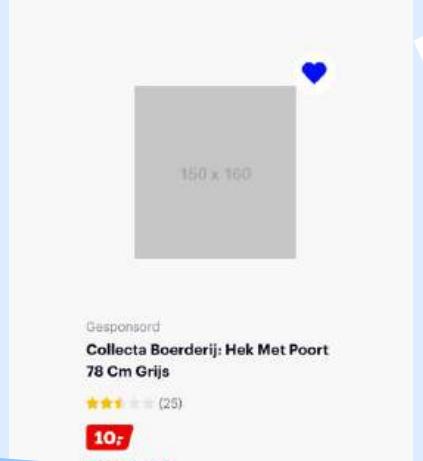
Close this dialog

# The (old) webshop in 60 seconds

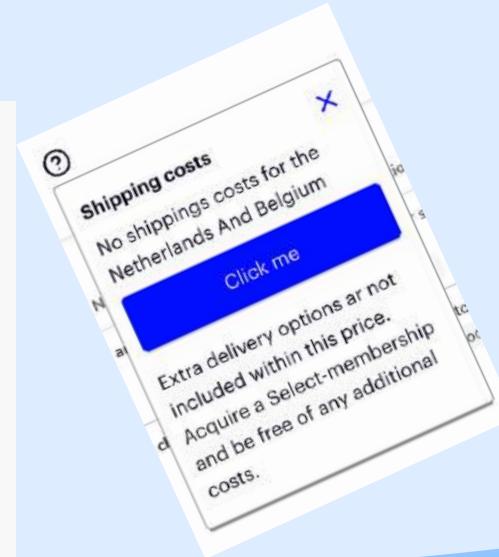
Home > Link1 > Link2 > Link3 > CurrentPage



Click me! (Variant A)



Gesponsord  
Collecta Boorderij: Hek Met Poort  
78 Cm Grijs  
★★★ (25)  
10,-



## DESIGN SYSTEM

Option without body

Option with body

Option with icon

I confirm that I read and accept the terms and conditions  
And present a nice sublabel

I am disabled  
You should not be able to select me!

Select second item

### Icons


# Storybook

The screenshot shows the Storybook interface. On the left is a sidebar containing a tree view of components:

- ChipGroup
- Dialog
- Link
- Popover
- Price
- ToggleSwitch
- Tooltip
- FOUNDATION
  - Colors
  - Icons
- INPUT
  - Checkbox Group
    - Default (highlighted)
    - Controlled
    - Form
    - Disabled
    - Group Size Small
    - Invalid
  - Checkbox
  - Radio Group
  - Select
  - Text
  - Textarea
- LAYOUT
  - Box
  - Flex
  - Grid
- BETA
  - Collapsible
  - CountrySelectFlagIcon
  - CountrySelectV2
  - CountrySelect
  - EnergyLabel
  - Image
  - Select Custom
  - MediaCard
  - Offcanvas

The main area displays a component named "Fruit" which is a Checkbox Group. It contains four checkboxes: Banana, Strawberry, Pear (which is checked), and Apple. Below the component preview is a controls panel with the following fields:

Name	Control
label	"Fruit"
children	<code>children : [</code> ↳ 0 : {...} 5 keys ↳ 1 : {...} 5 keys ↳ 2 : {...} 5 keys ↳ 3 : {...} 5 keys ]
value	<code>value : [</code> 0 : "pear" ]
name	"fruit"
className	Set object
isSmall	Set boolean
isTouched	Set object
message	Set object
validationState	Choose option...
isDisabled	Set boolean
onChange	-
aria-labelledby	Set object

# Testing

```
import { composeStories } from '@storybook/react';
import { render, screen } from '@testing-library/react';
import { renderWithUserEvent } from '@utils/jest-resources';
import { axe, toHaveNoViolations } from 'jest-axe';
import * as stories from '../_stories_/checkbox-group.stories';

const { Default, Controlled, Disabled, GroupSizeSmall } = composeStories(stories);

describe('<CheckboxGroup />', () => {
  it('should handle click events', async () => {
    const { user } = renderWithUserEvent(<Controlled />);

    const checkboxGroup = screen.getByRole('group');
    const checkboxes = screen.getAllByRole('checkbox');

    expect(checkboxGroup).toBeDefined();
    expect(checkboxes.length).toBe(4);

    expect(screen.getByLabelText('Banana')).not.toBeChecked();
    expect(screen.getByLabelText('Strawberry')).not.toBeChecked();
    expect(screen.getByLabelText('Pear')).toBeChecked();
    expect(screen.getByLabelText('Apple')).toBeChecked();

    // Apple are checked by default
    await user.click(screen.getByLabelText('Apple'));
    expect(screen.getByLabelText('Apple')).not.toBeChecked();

    // Banana is unchecked by default
    await user dblClick(screen.getByLabelText('Banana'));
    expect(screen.getByLabelText('Banana')).not.toBeChecked();

    // Pear is checked and disabled
    await user.click(screen.getByLabelText('Pear'));
    expect(screen.getByLabelText('Pear')).toBeChecked();
  });
});
```

# Testing

```
import { composeStories } from '@storybook/react';
import { render, screen } from '@testing-library/react';
import { renderWithUserEvent } from '@utils/jest-resources';
import { axe, toHaveNoViolations } from 'jest-axe';
import * as stories from '../__stories__/checkbox-group.stories';

const { Default, Controlled, Disabled, GroupSizeSmall } = composeStories(stories);

describe('<CheckboxGroup />', () => {
  it('should handle click events', async () => {
    const { user } = renderWithUserEvent(<Controlled />);

    const checkboxGroup = screen.getByRole('group');
    const checkboxes = screen.getAllByRole('checkbox');

    expect(checkboxGroup).toBeDefined();
    expect(checkboxes.length).toBe(4);

    expect(screen.getByLabelText('Banana')).not.toBeChecked();
    expect(screen.getByLabelText('Strawberry')).not.toBeChecked();
    expect(screen.getByLabelText('Pear')).toBeChecked();
    expect(screen.getByLabelText('Apple')).toBeChecked();

    // Apple are checked by default
    await user.click(screen.getByLabelText('Apple'));
    expect(screen.getByLabelText('Apple')).not.toBeChecked();

    // Banana is unchecked by default
    await user dblClick(screen.getByLabelText('Banana'));
    expect(screen.getByLabelText('Banana')).not.toBeChecked();

    // Pear is checked and disabled
    await user.click(screen.getByLabelText('Pear'));
    expect(screen.getByLabelText('Pear')).toBeChecked();
  });
});
```

# Getting your code merged

## WSP1-13136 | Add complete DSA Modal

Merged Steven den Hartog requested to merge [WSP1-13136](#) into [master](#) 9 months ago

Overview 16 Commits 37 Pipelines 12 Changes 54

Link to experiment: [https://web.ra-7593.stg.bol.io/nl/nl/p/Sapiens/9200000116370070?experiments=WSP1-12634\\_a](https://web.ra-7593.stg.bol.io/nl/nl/p/Sapiens/9200000116370070?experiments=WSP1-12634_a)

Closes [WSP1-13136](#)

Edited 9 months ago by Martijn Steenbergen



### Merged result pipeline #3212358 passed

Merged result pipeline passed for [9bdee4ce](#) 8 months ago  
Test coverage 24.60% from 6 jobs [?](#)

Deployed to [review/ra-7593](#) 8 months ago



### Approved by

Merged by  [Martijn Steenbergen](#) 8 months ago

[Revert](#) [Cherry-pick](#)

#### Merge details

- Changes merged into master with [ef030b3a](#).
- Deleted the source branch.
- Closed [WSP1-13136](#)
- Auto-merge enabled

### Pipeline #3212762 passed

Pipeline passed for [ef030b3a](#) on [master](#) 8 months ago



Deployed to [pro](#) 8 months ago

[View app](#) 

Deployed to [stg](#) 8 months ago

[View app](#) 

## Activity

- Steven den Hartog added 1 commit 9 months ago

[→ 1 commit → WSP1-13136 fix tests + use forGraphQL command I found, also add extra debug logging to inspector](#)

All activity 

# Getting your code merged

**WSP1-13136 | Add complete DSA Modal**

3 Merge Checks 100% 8 months ago

**Overview** 16 Commits 37 Pipelines 12 Changes 54

Link to experiment: [https://web.ra-7593.stg.bol.io/nl/nl/p/Sapiens/9200000116370070?experiments=WSP1-12634\\_a](https://web.ra-7593.stg.bol.io/nl/nl/p/Sapiens/9200000116370070?experiments=WSP1-12634_a)

Closes [WSP1-13136](#)

Edited 9 months ago by Martijn Steenbergen

0 0 0

**Merged result pipeline #3212358 passed** → ✓  
Merged result pipeline passed for 9bdee4ce 8 months ago  
Test coverage 24.60% from 6 jobs ⓘ

Deployed to [review/ra-7593](#) 8 months ago

8✓ Approved by ✓

**Merged by Martijn Steenbergen** 8 months ago

**Revert** **Cherry-pick**

**Merge details**

- Changes merged into master with [ef030b3a](#).
- Deleted the source branch.
- Closed [WSP1-13136](#)
- Auto-merge enabled

**Pipeline #3212762 passed** ✓  
Pipeline passed for ef030b3a on master 8 months ago

Deployed to pro 8 months ago

Deployed to stg 8 months ago

**View app** **View app**

**Activity**

Steven den Hartog added 1 commit 9 months ago

→ 2-157/2a - WSP1-13136 fix tests + use forGraphQL command I found, also add extra debug logging to inspector

All activity ↑

# Getting your code merged

## WSP1-13136 | Add complete DSA Modal

Merged Steven den Hartog requested to merge WSP1-13136 into master 9 months ago

Overview 16 Commits 37 Pipelines 12 Changes 54

Link to experiment: [https://web.ra-7593.stg.bol.io/nl/nl/p/Sapiens/9200000116370070?experiments=WSP1-12634\\_a](https://web.ra-7593.stg.bol.io/nl/nl/p/Sapiens/9200000116370070?experiments=WSP1-12634_a)

Closes WSP1-13136

Edited 9 months ago by Martijn Steenbergen



Merged result pipeline #3212358 passed

Merged result pipeline passed for 9bdee4ce 8 months ago  
Test coverage 24.60% from 6 jobs

Deployed to review/ra-7593 8 months ago

Approved by

Merged by [Martijn Steenbergen](#) 8 months ago

Revert Cherry-pick

Merge details

- Changes merged into master with [ef030b3a](#).
- Deleted the source branch.
- Closed [WSP1-13136](#)
- Auto-merge enabled

Pipeline #3212762 passed

Pipeline passed for ef030b3a on master 8 months ago

Deployed to pro 8 months ago

Deployed to stg 8 months ago



[View app](#)

[View app](#)

### Activity

- Steven den Hartog added 1 commit 9 months ago

[See 157/200](#) · WSP1-13136 fix tests + use forGraphQL command I found, also add extra debug logging to inspector

All activity

# Getting your code merged

## WSP1-13136 | Add complete DSA Modal

Merged Steven den Hartog requested to merge WSP1-13136 into master 9 months ago

Overview 16 Commits 37 Pipelines 12 Changes 54

Link to experiment: [https://web.ra-7593.stg.bol.io/nl/nl/p/Sapiens/9200000116370070?experiments=WSP1-12634\\_a](https://web.ra-7593.stg.bol.io/nl/nl/p/Sapiens/9200000116370070?experiments=WSP1-12634_a)

Closes WSP1-13136

Edited 9 months ago by Martijn Steenbergen



Merged result pipeline #3212358 passed

Merged result pipeline passed for 9bdee4ce 8 months ago  
Test coverage 24.60% from 6 jobs

Deployed to review/ra-7593 8 months ago



8✓ Approved by

Merged by [Martijn Steenbergen](#) 8 months ago

Revert Cherry-pick

Merge details

- Changes merged into master with [ef030b3a](#).
- Deleted the source branch.
- Closed [WSP1-13136](#)
- Auto-merge enabled

✓ Pipeline #3212762 passed

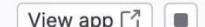
Pipeline passed for ef030b3a on master 8 months ago



Deployed to pro 8 months ago



Deployed to stg 8 months ago



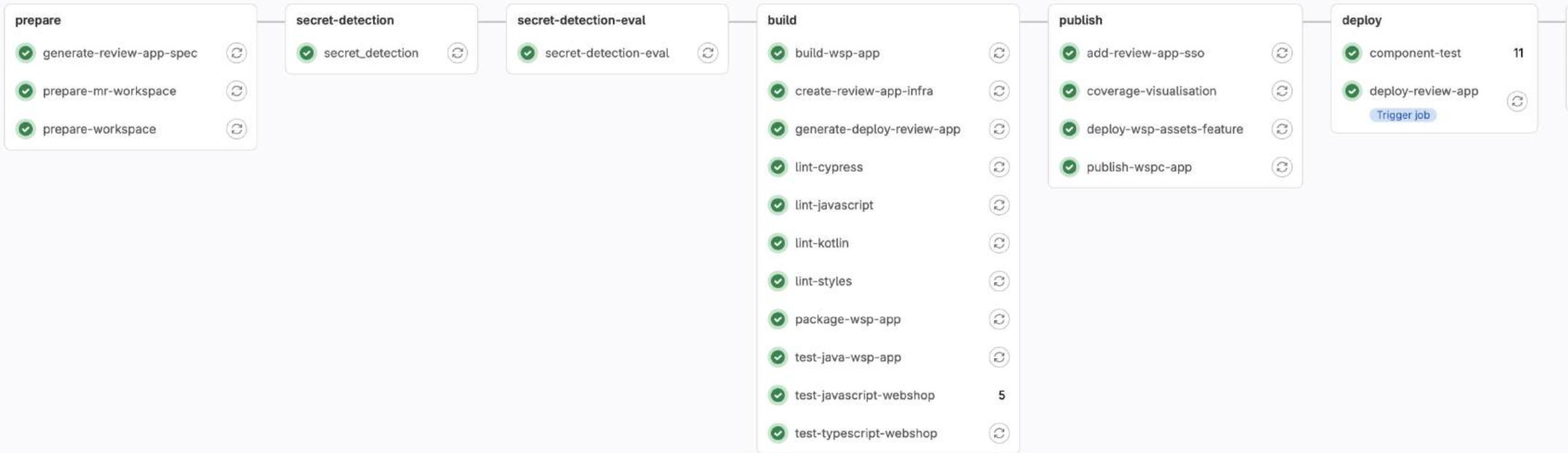
### Activity

- Steven den Hartog added 1 commit 9 months ago

[→ 1 commit · WSP1-13136 fix tests + use forGraphQL command I found, also add extra debug logging to inspector](#)



# Getting your code merged



## WSP1-13136 | Add complete DSA Modal

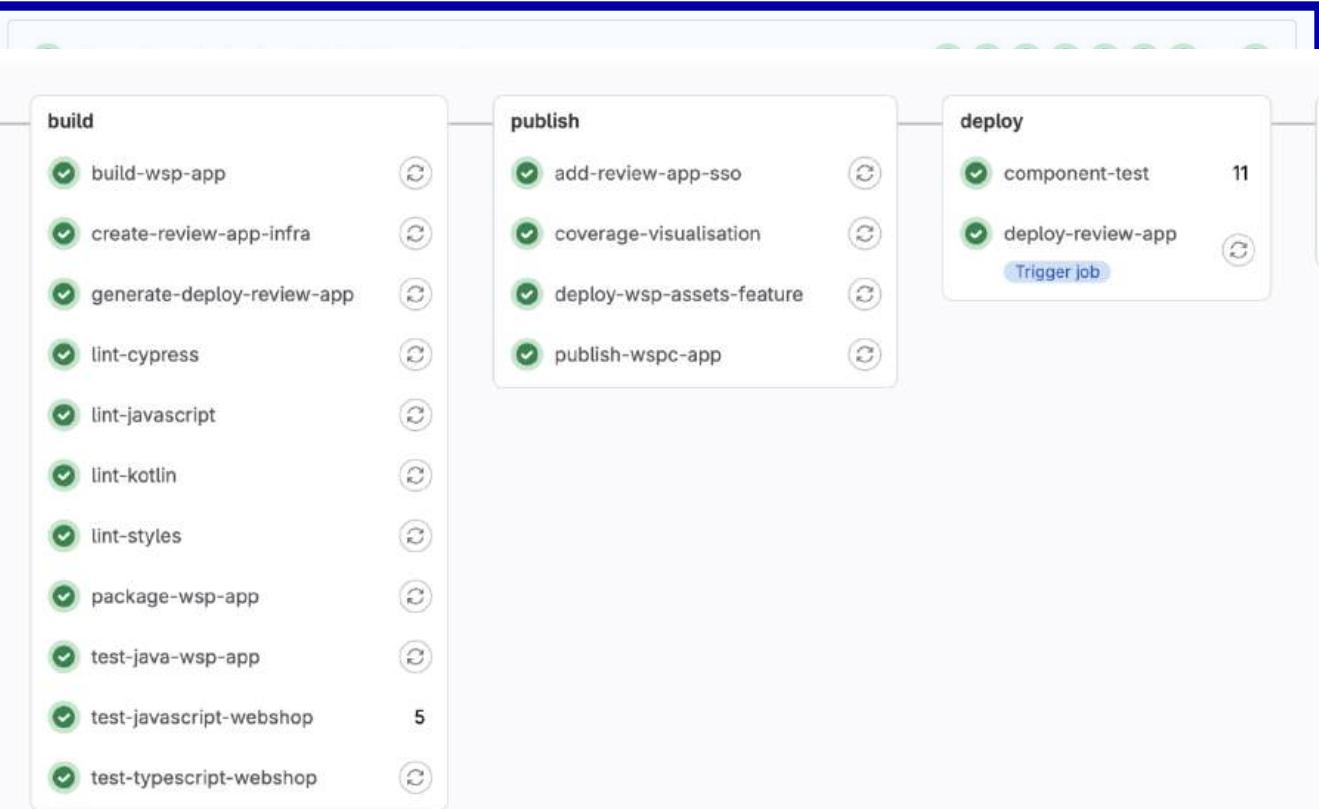
Merged Steven den Hartog requested to merge [WSP1-13136](#) into [master](#) 9 months ago

Overview 16 Commits 37 Pipelines 12 Changes 54

Link to experiment: [https://web.ra-7593.stg.bol.io/nl/nl/p/Sapiens/9200000116370070?experiments=WSP1-12634\\_a](https://web.ra-7593.stg.bol.io/nl/nl/p/Sapiens/9200000116370070?experiments=WSP1-12634_a)

Closes [WSP1-13136](#)

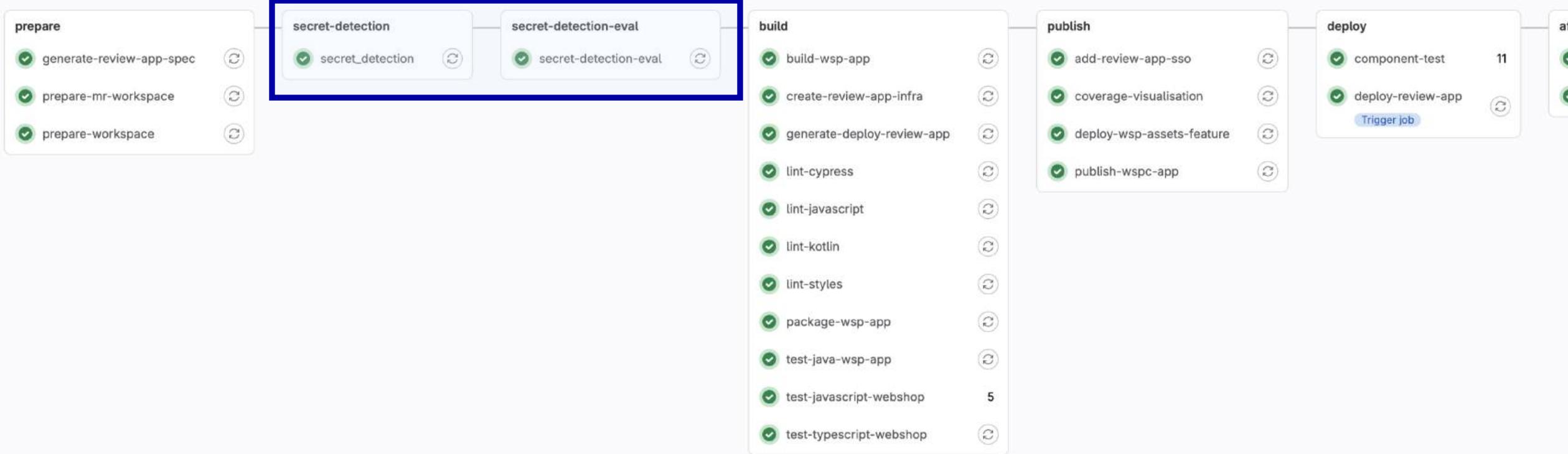
Edited 9 months ago by Martijn Steenbergen



• Steven den Hartog added 1 commit 9 months ago

→ 2-157/2 - WSP1-13136 fix tests + use forGraphQI command I found, also add extra debug logging to inspector

# Getting your code merged

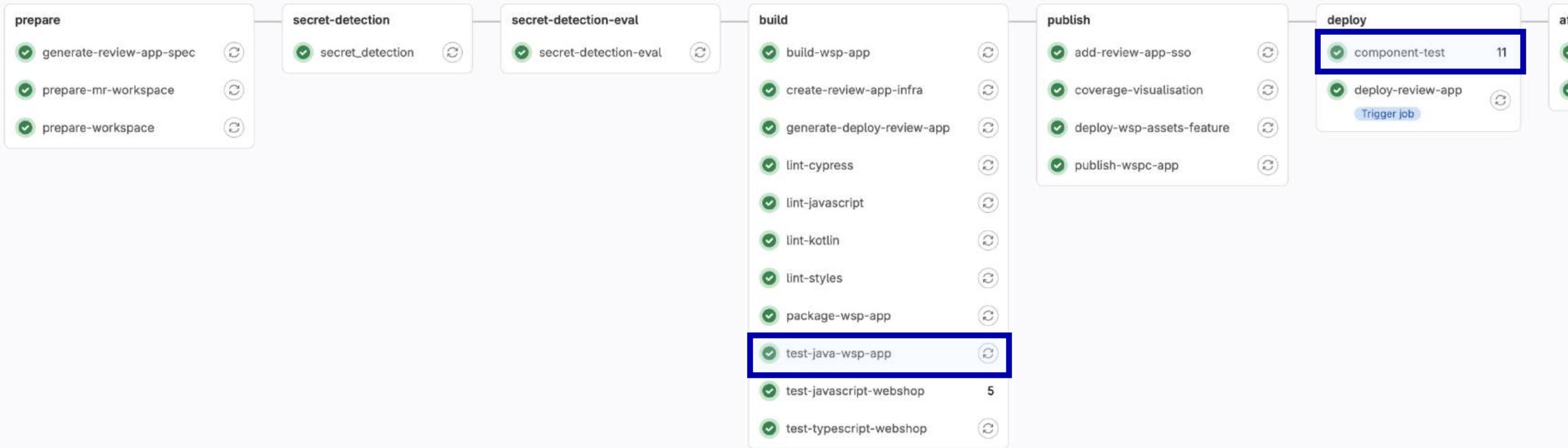


- Steven den Hartog added 1 commit 9 months ago

→ 2-157/2 - WSP1-13136 fix tests + use forGraphQl command I found, also add extra debug logging to inspector

# Getting your code merged

7649+ tests



- Steven den Hartog added 1 commit 9 months ago

→ 2-157/2 - WSP1-13136 fix tests + use forGraphQI command I found, also add extra debug logging to inspector

# Getting your code merged

7649+ tests

- **Martijn Steenbergen** added 6 commits 9 months ago
    - [f5ea36e7 - WSP1-13136](#) | Change colors of icons
    - [1f38ab64 - WSP1-13136](#) | Stories for DSA belong in DSAModal
    - [4a5c0ba9 - WSP1-13136](#) | remove useless tag

[▼ Toggle commit list](#)
  - **Martijn Steenbergen** added 1 commit 9 months ago
    - [89097dd1 - WSP1-13136](#) | More lint fixes

[Compare with previous version](#)
  - **Martijn Steenbergen** added 1 commit 9 months ago
    - [877bbb32 - WSP1-13136](#) | Fix productlistCC

[Compare with previous version](#)
  - **Martijn Steenbergen** changed the description 9 months ago [▼ Compare with previous version](#)
-  **Salman Peirovi** @speirovi started a thread on the diff 9 months ago  
Resolved 8 months ago by **Martijn Steenbergen** [▼ Show thread](#)

 **Salman Peirovi** @speirovi started a thread on the diff 9 months ago  
Resolved 8 months ago by **Martijn Steenbergen** [▼ Show thread](#)

 **Salman Peirovi** approved this merge request 9 months ago

 **Leon Shen** @lshen started a thread on an old version of the diff 9 months ago  
Automatically resolved 8 months ago by **Martijn Steenbergen** [▼ Show thread](#)

 **Leon Shen** @lshen started a thread on an old version of the diff 9 months ago  
Resolved 8 months ago by **Martijn Steenbergen** [▼ Show thread](#)

 **Leon Shen** @lshen started a thread on an old version of the diff 9 months ago  
Automatically resolved 8 months ago by **Martijn Steenbergen** [▼ Show thread](#)

 **Leon Shen** approved this merge request 9 months ago

  - **Martijn Steenbergen** added 1 commit 8 months ago
    - [2d251f68 - WSP1-13136](#) | Apply suggestions

[Compare with previous version](#)

**Martijn Steenbergen** added 1 commit 8 months ago

  - [2c51e146 - WSP1-13136](#) | Change Adprovider to String (Look ma! I'm a backender now!)
  - [603d8139 - WSP1-13136](#) | Actual urls!

# Getting your code merged

7649+ tests

- **Martijn Steenbergen** added 4 commits 8 months ago
  - [2c51e146 - WSP1-13136](#) | Change Adprovider to String (Look ma! I'm a backender now!)
  - [603d8139 - WSP1-13136](#) | Actual urls!
  - [a0045ec4 - WSP1-13136](#) | Slight hack to make sure button is not in focus
- **Martijn Steenbergen** added 1 commit 8 months ago
  - [1dac8d58 - WSP1-13136](#) | The code is pretty ✨ now[Compare with previous version](#)
- **Martijn Steenbergen** resolved all threads 8 months ago



**Leon Shen** @lshen started a thread on commit a0045ec4 8 months ago  
Resolved 8 months ago by Leon Shen

▼ Show thread

- **Martijn Steenbergen** added 1 commit 8 months ago
  - [2cd0b6bc - WSP1-13136](#) | Fix test[Compare with previous version](#)
- **Martijn Steenbergen** added 1 commit 8 months ago
  - [c4770745 - WSP1-13136](#) | Add comment why we are using the hack[Compare with previous version](#)
- **Leon Shen** resolved all threads 8 months ago
- **Martijn Steenbergen** added 1 commit 8 months ago
  - [ccff4033 - WSP1-13136](#) | undo changes to listpage product[Compare with previous version](#)
- **Martijn Steenbergen** enabled an automatic merge when the pipeline for [9bdee4ce](#) succeeds 8 months ago
- **Martijn Steenbergen** merged 8 months ago
- **Martijn Steenbergen** mentioned in commit [ef030b3a](#) 8 months ago

Preview | **B** *I* ~~S~~ | `|` `/>` `</` `&` `≡` `≡` `≡` `≡` |

Write a comment or drag your files here...

# So? Done?

 Checkin go live DSA WSP sponsored products / topbanner

 Tue 23/01/2024 09:00 - 09:15

 Microsoft Teams Meeting

# Releasing

## A/B Testing



## Feature Toggle

Turn on a new feature

- When you want
- To a small percentage of your users
- And roll back when it breaks



# Rituals

Chapter Two



# We don't do sprints

# We don't do sprints (not really)

- At any given time, we have multiple people working on multiple features in multiple stages of development
- We produce a shippable product every day/all the time
- We use a kanban style way of development
- Weekly check

# We do everything else (and more)

- Planning every Monday
- Marketplace every Thursday
- Retrospective every other Wednesday

“  
Agile does not  
work for us



”

# “Agile does not work for us”

- What is your definition of Agile?
- No one is forcing you to use everything from scrum
- No process is going to save you from a toxic environment

# Being Senior

Chapter Three



# Being a good senior engineer

Being able to make difficult decisions,  
with people liking you afterwards

# Being a good senior engineer

Being able to make difficult decisions,  
with people liking you afterwards

# Being a good senior engineer

Being able to make difficult decisions,  
with people liking you afterwards

# Being a good senior engineer

Being able to make difficult decisions,  
with people liking you afterwards

# Young Professional

Chapter Four



[Sign in with single sign-on](#)

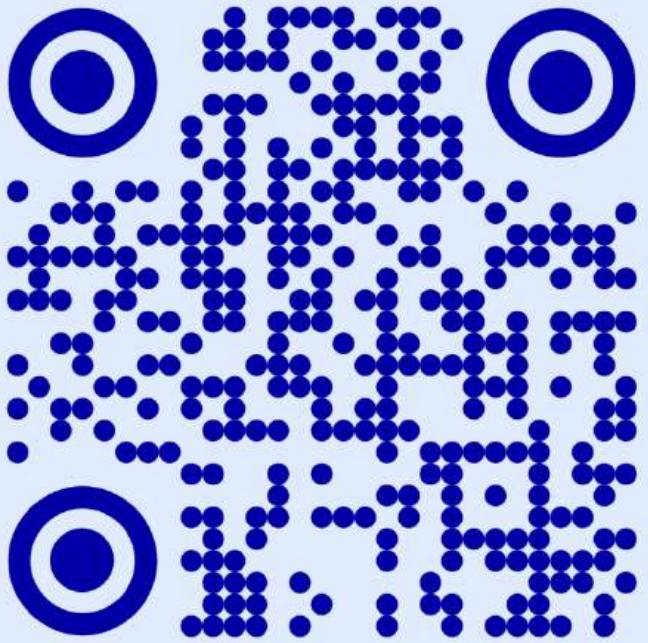
▶ [FAQ](#)

▶ [Other login options](#)

# Young Professional Program

- Spend one day a week on whatever you want
  - Learn a new language (Dutch/Go)
  - Do fun side projects
  - Develop yourself!
- Be a Young Professional
- Work at bol.com
  - Warm
  - Helpful



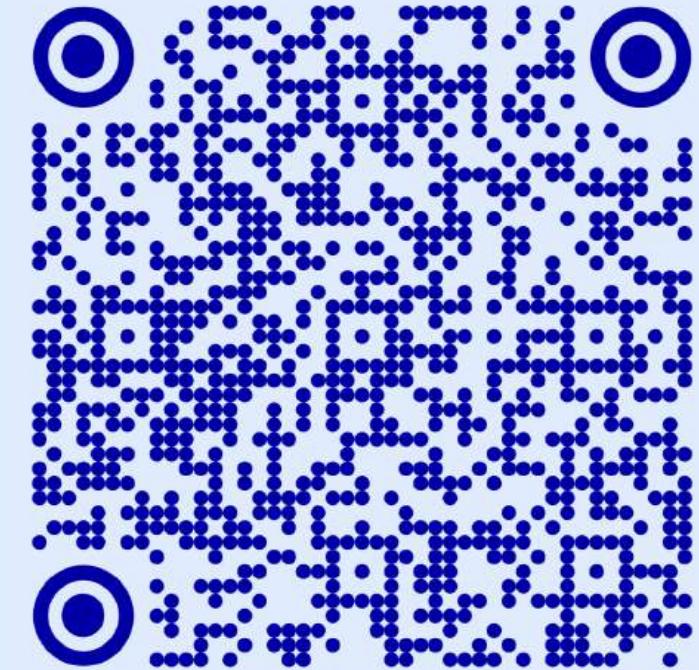


Propose questions we can  
ask during the panel



## We are taking a break

Let's resume at 09:45 pm



Learn about working at bol

# Round Table

Chapter Six



# Round Table



**Tim Beerens**  
Team Pathfinder  
Webshop



**Naval Bhagat**  
Team Helm  
Logistics



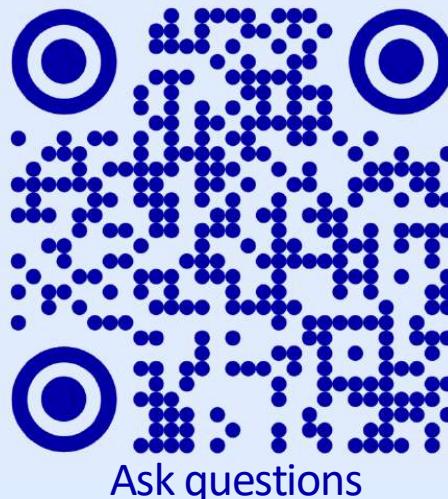
**Giorgia Corrado**  
Team Botsquad  
ML & Data



**Chris Langhout**  
Team Northstar  
Developer Productivity



**Martijn Steenbergen**  
Team Matter  
Webshop



## Contact

Martijn Steenbergen

[martijnsteenbergen@bol.com](mailto:martijnsteenbergen@bol.com)

Tim Beerens

[tbeerens@bol.com](mailto:tbeerens@bol.com)

Giorgia Corrado

[gcorrado@bol.com](mailto:gcorrado@bol.com)

Naval Bhagat

[nbhagat@bol.com](mailto:nbhagat@bol.com)

Chris Langhout

[clanghout@bol.com](mailto:clanghout@bol.com)

## Contents

A project from beginning to end

It takes a lot to releasing good software

Rituals

# Thank you

Software can be built in lots of ways,  
we use mostly scrum

Being Senior

In the end, it is all about people

Join Bol





# Technical Debt

CSE2115 Software  
Engineering Methods

Andy Zaidman  
(based on Annibale Panichella's slides from 2023-2024)

# More on technical debt and metrics

- Book: "Refactoring: Improving the Design of Existing Code." by M. Fowler Chapters 2, 3, 5, 6, 7, 8
- Additional reading: "Technical Debt in Practice: How to Find It and Fit It" by Ernst, Kazman and Delange

# What is technical debt?

Technical debt is a metaphor

- **Financial debt:** we borrow money to achieve a goal more quickly (e.g., buying a car or house)
- **Technical debt:** we try to get a product to market as quickly as possible, even though this product may be flawed, even though we may not fully understand our market and hence the features that we should be providing, and even though, in our hurry to get the product out the door, we inevitable take shortcuts and make some ill-considered decisions





# Why “debt”?

- While the “quick and dirty” solution got your product to the market, you incur a penalty later. For example, it becomes more difficult to make adjustments to the software system.

# Drowning in technical debt

- Once worked with a company X
- They had a method with 64 nested if statements in it  
(remember cyclomatic complexity from OOP in year 1?)
- No software engineer of the company dared touch that method any more, afraid of breaking something

# Broken Windows Theory

- Theory from criminology
- “visible signs of crime, antisocial behavior and civil disorder create an urban environment that encourages further crime and disorder, including serious crimes”
- Research has shown similar effects in terms of (internal) software quality



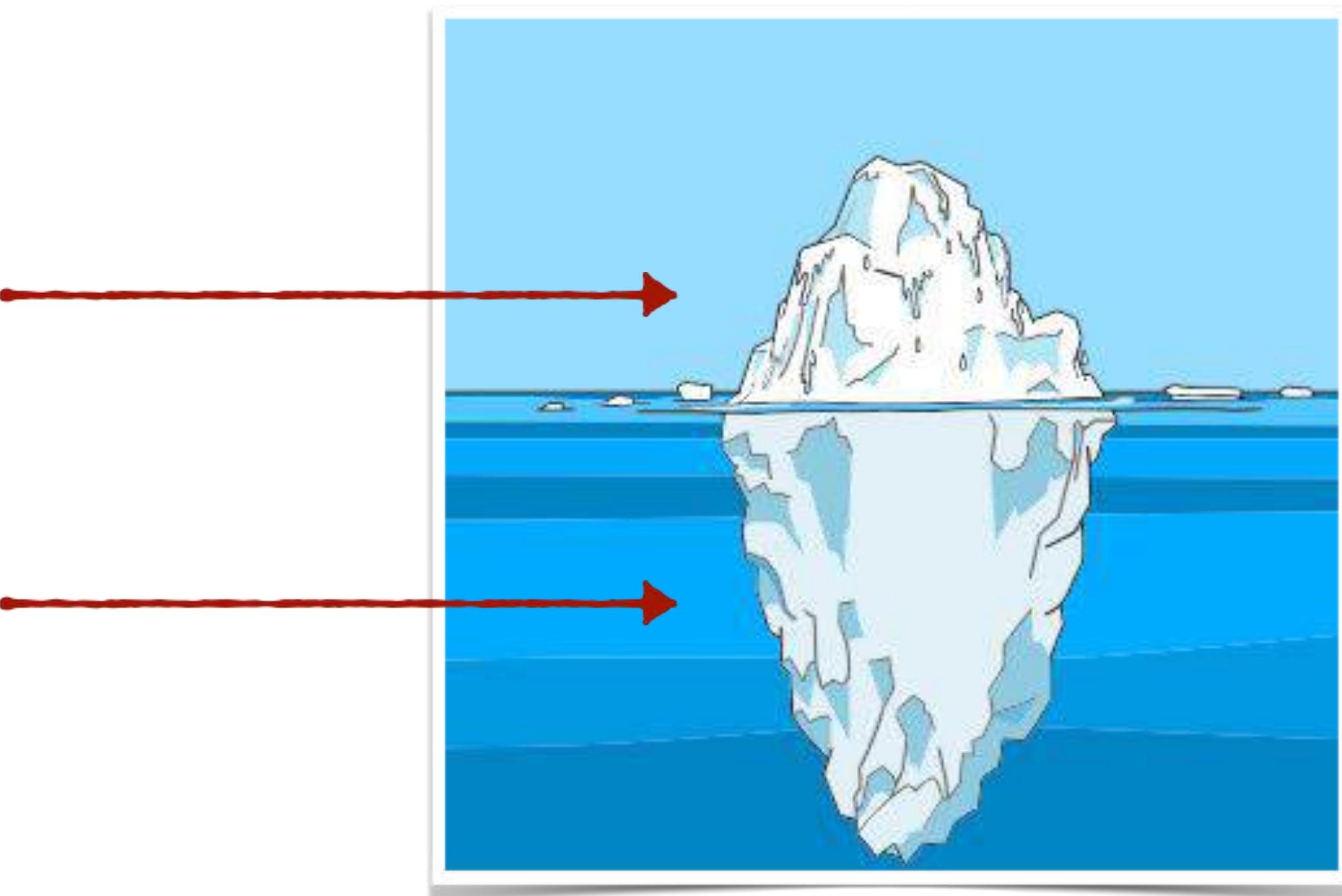
# Technical Debt

Any software (for real-world problems) has to types of complexity:

- **Intrinsic complexity:** it is the essential complexity that required to solve the problem (e.g., complexity of sort algorithms)
- **Additional complexity or cruft** added to complete the implementation quickly



This “cruft” makes it harder to understand and maintain the software

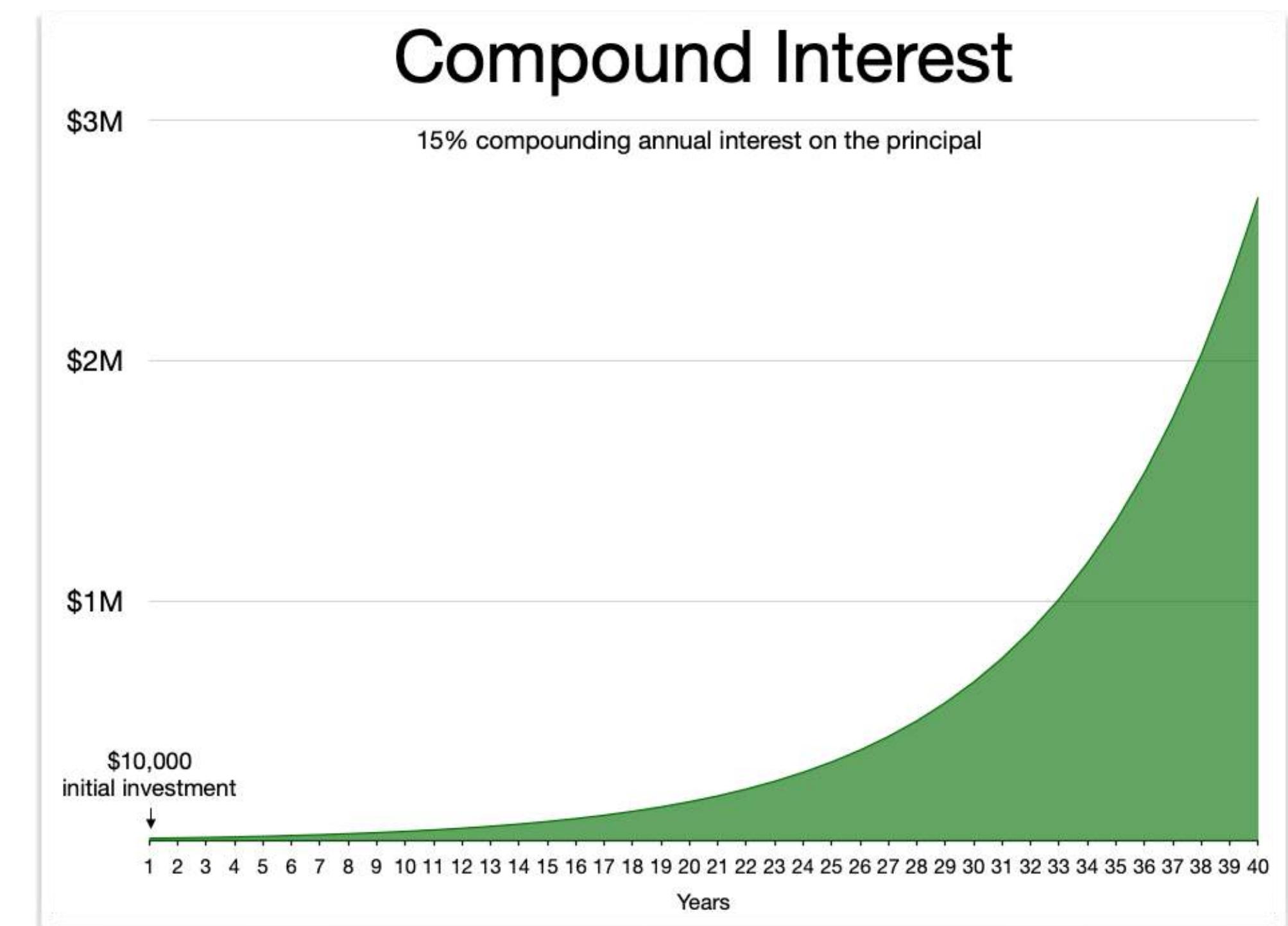


# Technical Debt

Technical Debt is a metaphor that associates the ``cruft'' with financial debt:

- Someone in the future will pay some extra effort to add new features or to change the code
- Technical debt accumulates (like negative interests). The more debt we have, the more new debt it will accumulate if we don't fix the code

Technical Debt was coined by Ward Cunningham in 1992 at the OOPSLA conference



# This Lecture

## Software Development Life Cycle

SDLC is the process followed in SE for planning, designing, developing, testing, and deploying software projects

Different SDLC models can be used.

Most of the existing models use forms of the same five steps, but differ on emphasis, frequency, order.

Here are a few SDLC models:

- Waterfall (Winston W. Royce, 1970)
- Spiral (Barry Boehm, 1986)
- V-model (1991)
- Agile (2001)
- Scrum (2001)

```
graph TD; subgraph SDLC [Software Development Life Cycle]; A([Analysis]) --> D([Design]); D --> I([Implementation]); I --> M([Maintenance]); M --> P([Planning]); P --> A; end; RM[Refactoring] --> A; RM --> D; RM --> I; RM --> M; SM[Software Metrics] --> A; SM --> D; SM --> I; SM --> M;
```

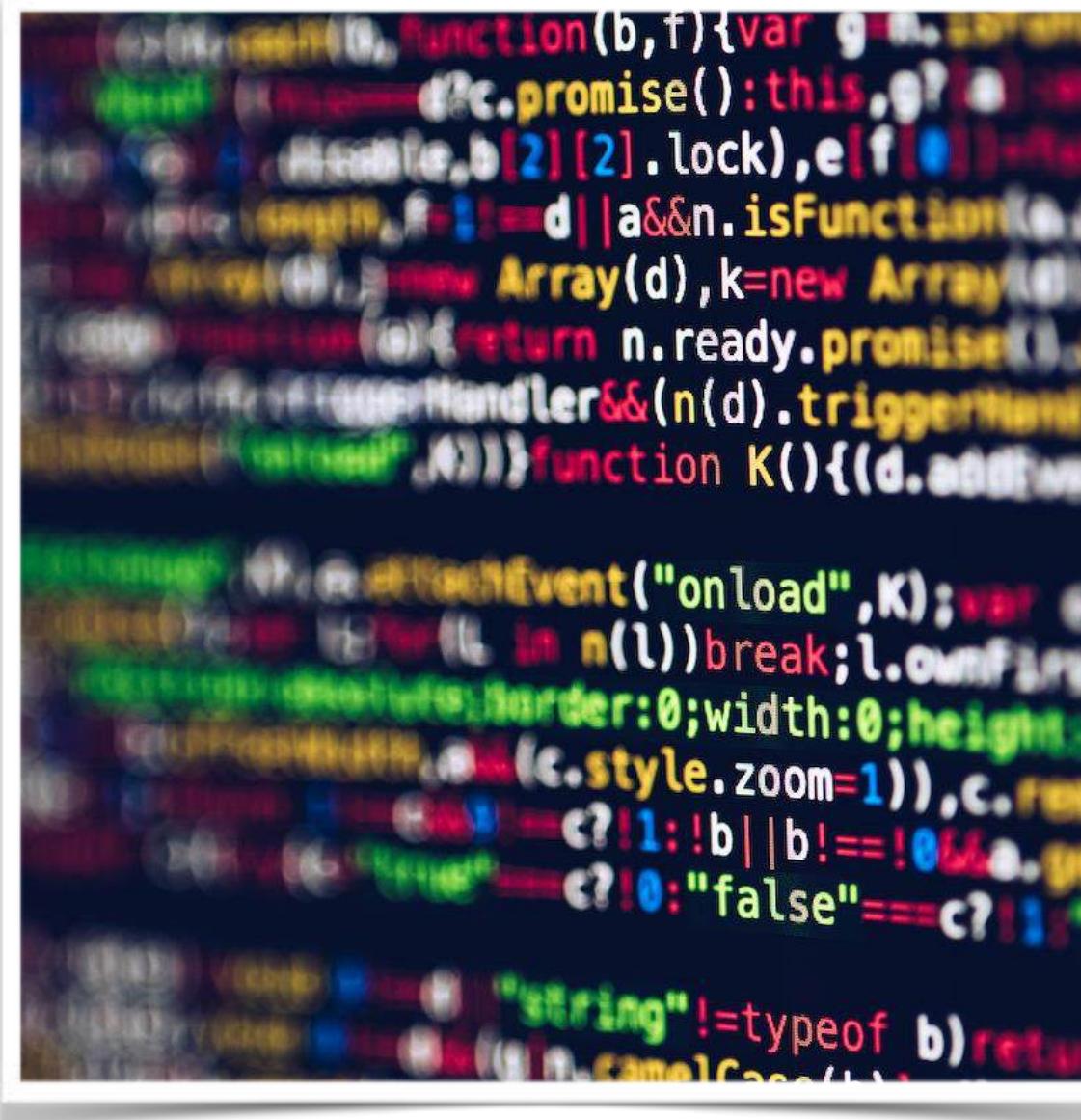
# Roadmap

- Bad Code Smells
- Software Metrics
- Refactoring
- Tools



# Types of Technical Debt

Code-related TD



Testing-related TD



Organizational TD



# Types of Technical Debt

Today we focus on this

Code-related TD



Testing-related TD



Organizational TD



# Self-admitted technical debt

Software engineers leave behind “TODOs”  
where they see that they can improve, but  
currently don’t have time to fix it



A screenshot of a Java code editor showing the file `src/main/java/org/apache/commons/lang3/reflect/MemberUtils.java`. The code contains the following lines:

```
31 * @since 2.5
32 */
33 final class MemberUtils {
34     // TODO extract an interface to implement compareParameterSets(...)?
```

# Example of Technical Debt

Project = Apache commons BCEL

Class = Pass2Verifier.java

```
1  /*
2   * Licensed to the Apache Software Foundation (ASF) under one or more
3   * contributor license agreements. See the NOTICE file distributed with
4   * this work for additional information regarding copyright ownership.
5   * The ASF licenses this file to You under the Apache License, Version 2.0
6   * (the "License"); you may not use this file except in compliance with
7   * the License. You may obtain a copy of the License at
8   *
9   *      http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in writing, software
12  * distributed under the License is distributed on an "AS IS" BASIS,
13  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14  * See the License for the specific language governing permissions and
15  * limitations under the License.
16  *
17  */
18 package org.apache.bcel.verifier.statics;
19
20
21 import java.util.HashMap;
22 import java.util.HashSet;
23 import java.util.Locale;
24 import java.util.Map;
25 import java.util.Set;
26
27 import org.apache.bcel.Const;
28 import org.apache.bcel.Constants;
29 import org.apache.bcel.Repository;
30 import org.apache.bcel.classfile.Attribute;
31 import org.apache.bcel.classfile.ClassFormatException;
32 import org.apache.bcel.classfile.Code;
33 import org.apache.bcel.classfile.CodeException;
34 import org.apache.bcel.classfile.Constant;
35 import org.apache.bcel.classfile.ConstantClass;
36 import org.apache.bcel.classfile.ConstantDouble;
37 import org.apache.bcel.classfile.ConstantFieldref;
38 import org.apache.bcel.classfile.ConstantFile;
39 import org.apache.bcel.classfile.ConstantInterfaceMethodref;
40 import org.apache.bcel.classfile.ConstantMethodref;
41 import org.apache.bcel.classfile.ConstantPool;
42 import org.apache.bcel.classfile.ConstantString;
43 import org.apache.bcel.classfile.ConstantUTF16;
44 import org.apache.bcel.classfile.ConstantUTF32;
45 import org.apache.bcel.classfile.Field;
46 import org.apache.bcel.classfile.FieldRef;
47 import org.apache.bcel.classfile.Instruction;
48 import org.apache.bcel.classfile.Method;
49 import org.apache.bcel.classfile.Type;
50 import org.apache.bcel.classfile.TypeRef;
51 import org.apache.bcel.classfile.Visitor;
52 import org.apache.bcel.classfile.constantpool.ConstantPoolGen;
53 import org.apache.bcel.classfile.constantpool.ConstantPoolTable;
54 import org.apache.bcel.classfile.constantpool.ConstantTable;
55 import org.apache.bcel.classfile.constantpool.ConstantTableGen;
56 import org.apache.bcel.classfile.constantpool.ConstantTableTable;
57 import org.apache.bcel.classfile.constantpool.ConstantTableTableGen;
58 import org.apache.bcel.classfile.constantpool.ConstantTableTableTable;
59 import org.apache.bcel.classfile.constantpool.ConstantTableTableTableGen;
60 import org.apache.bcel.classfile.constantpool.ConstantTableTableTableTable;
61 import org.apache.bcel.classfile.constantpool.ConstantTableTableTableTableGen;
62 import org.apache.bcel.classfile.constantpool.ConstantTableTableTableTableTable;
63 import org.apache.bcel.classfile.constantpool.ConstantTableTableTableTableTableGen;
64 import org.apache.bcel.classfile.constantpool.ConstantTableTableTableTableTableTable;
65 import org.apache.bcel.classfile.constantpool.ConstantTableTableTableTableTableTableGen;
66 import org.apache.bcel.classfile.constantpool.ConstantTableTableTableTableTableTableTable;
67 import org.apache.bcel.classfile.constantpool.ConstantTableTableTableTableTableTableTableGen;
68 import org.apache.bcel.classfile.constantpool.ConstantTableTableTableTableTableTableTableTable;
69 import org.apache.bcel.classfile.constantpool.ConstantTableTableTableTableTableTableTableTableGen;
69
```

# Example

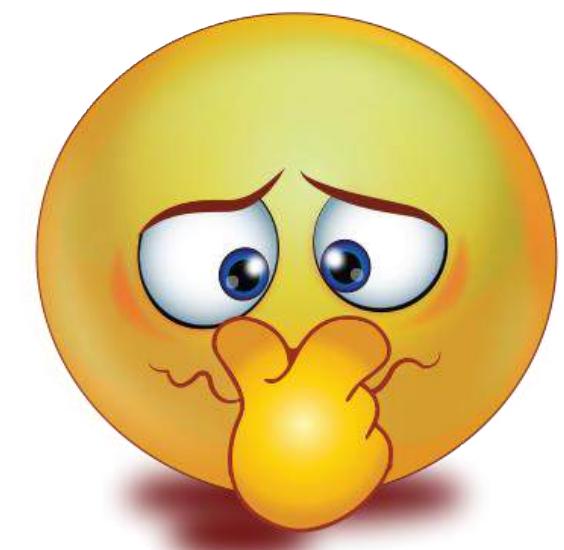
Project = Apache commons BCEL

Class = Pass2Verifier.java

> 1400 Lines of Code

```
1  /*
2   * Licensed to the Apache Software Foundation (ASF) under one or more
3   * contributor license agreements. See the NOTICE file distributed with
4   * this work for additional information regarding copyright ownership.
5   * The ASF licenses this file to You under the Apache License, Version 2.0
6   * (the "License"); you may not use this file except in compliance with
7   * the License. You may obtain a copy of the License at
8   *
9   *      http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in writing, software
12  * distributed under the License is distributed on an "AS IS" BASIS,
13  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14  * See the License for the specific language governing permissions and
15  * limitations under the License.
16  *
17  */
18 package org.apache.bcel.verifier.statics;
19
20
21 import java.util.HashMap;
22 import java.util.HashSet;
23 import java.util.Locale;
24 import java.util.Map;
25 import java.util.Set;
26
27 import org.apache.bcel.Const;
28 import org.apache.bcel.Constants;
29 import org.apache.bcel.Repository;
30 import org.apache.bcel.classfile.Attribute;
31 import org.apache.bcel.classfile.ClassFormatException;
32 import org.apache.bcel.classfile.Code;
33 import org.apache.bcel.classfile.CodeException;
34 import org.apache.bcel.classfile.Constant;
35 import org.apache.bcel.classfile.ConstantClass;
36 import org.apache.bcel.classfile.ConstantDouble;
37 import org.apache.bcel.classfile.ConstantFieldref;
38 import org.apache.bcel.classfile.ConstantFile;
39 import org.apache.bcel.classfile.ConstantInterfaceMethodref;
40 import org.apache.bcel.classfile.ConstantMethodref;
41 import org.apache.bcel.classfile.ConstantPool;
42 import org.apache.bcel.classfile.ConstantString;
43 import org.apache.bcel.classfile.ConstantUTF16;
44 import org.apache.bcel.classfile.ConstantUTF32;
45 import org.apache.bcel.classfile.Field;
46 import org.apache.bcel.classfile.FieldVisitor;
47 import org.apache.bcel.classfile.Instruction;
48 import org.apache.bcel.classfile.InstructionVisitor;
49 import org.apache.bcel.classfile.Method;
50 import org.apache.bcel.classfile.MethodVisitor;
51 import org.apache.bcel.classfile.Type;
52 import org.apache.bcel.classfile.TypeVisitor;
53 import org.apache.bcel.classfile.Visitor;
54 import org.apache.bcel.classfile.VisitorVisitor;
55 import org.apache.bcel.classfile.attribute.Annotation;
56 import org.apache.bcel.classfile.attribute.Attribute;
57 import org.apache.bcel.classfile.attribute.CodeAttribute;
58 import org.apache.bcel.classfile.attribute.ConstantPoolAttribute;
59 import org.apache.bcel.classfile.attribute.InnerClassesAttribute;
60 import org.apache.bcel.classfile.attribute.MethodAttribute;
61 import org.apache.bcel.classfile.attribute.SignatureAttribute;
62 import org.apache.bcel.classfile.attribute.TypeAttribute;
63 import org.apache.bcel.classfile.constantpool.Constant;
64 import org.apache.bcel.classfile.constantpool.ConstantPool;
65 import org.apache.bcel.classfile.constantpool.ConstantPoolTable;
66 import org.apache.bcel.classfile.constantpool.ConstantUtf8;
67 import org.apache.bcel.classfile.constantpool.ConstantUtf16;
68 import org.apache.bcel.classfile.constantpool.ConstantUtf32;
69 import org.apache.bcel.classfile.constantpool.ConstantString;
70 import org.apache.bcel.classfile.constantpool.ConstantDouble;
71 import org.apache.bcel.classfile.constantpool.ConstantFieldref;
72 import org.apache.bcel.classfile.constantpool.ConstantInterfaceMethodref;
73 import org.apache.bcel.classfile.constantpool.ConstantMethodref;
74 import org.apache.bcel.classfile.constantpool.ConstantPoolTable;
75 import org.apache.bcel.classfile.constantpool.ConstantString;
76 import org.apache.bcel.classfile.constantpool.ConstantUTF16;
77 import org.apache.bcel.classfile.constantpool.ConstantUTF32;
78 import org.apache.bcel.classfile.constantpool.ConstantUTF8;
79 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
80 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
81 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
82 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
83 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
84 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
85 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
86 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
87 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
88 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
89 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
90 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
91 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
92 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
93 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
94 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
95 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
96 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
97 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
98 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
99 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
100 import org.apache.bcel.classfile.constantpool.ConstantVisitor;
```

This doesn't  
smell that good

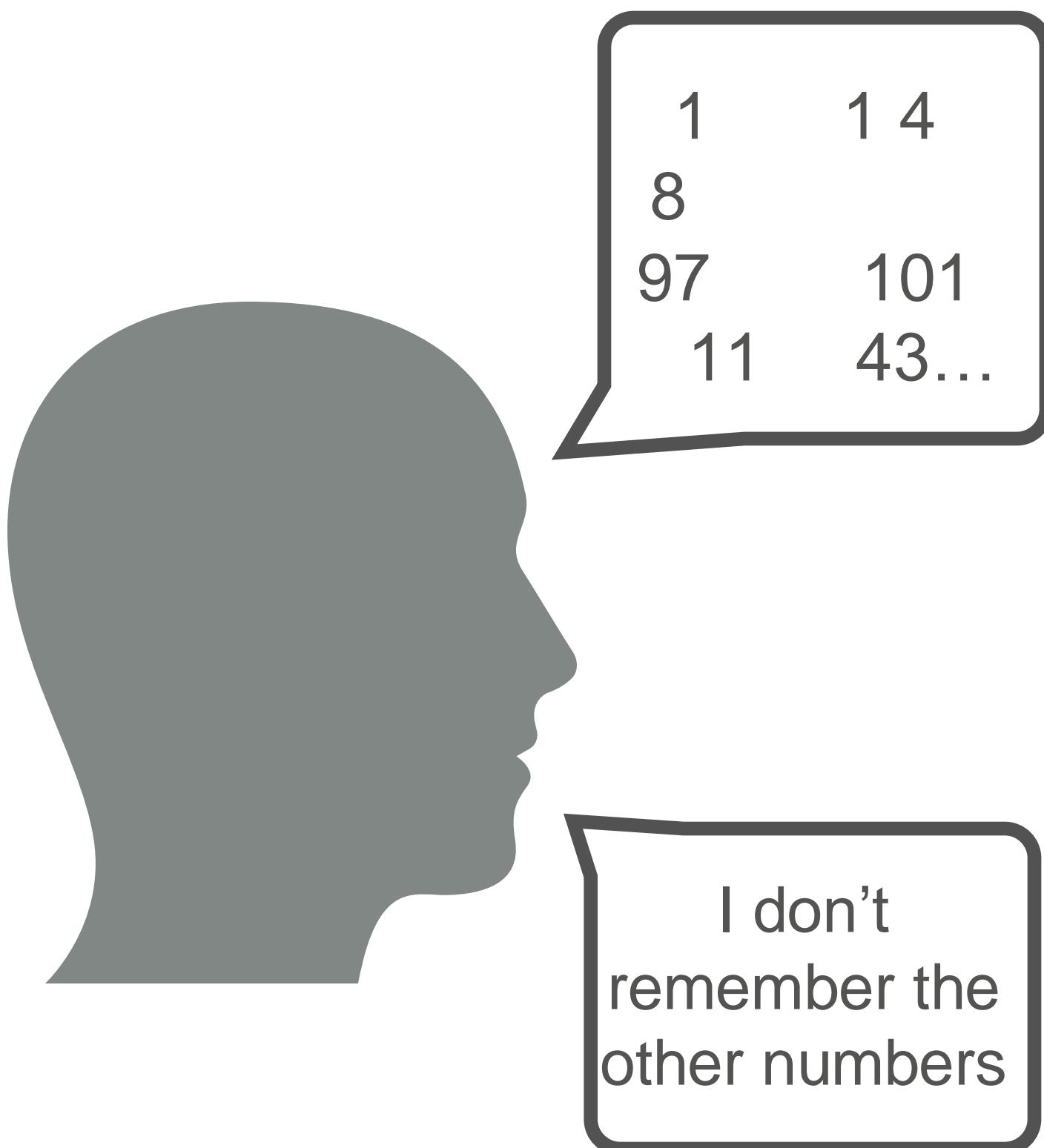


# Cohesion vs. Coupling



- High cohesion facilitates reuse (well-defined modules)
- Low coupling simplifies modification (all relevant code in one place)
- Higher cohesion can lead to higher coupling and vice versa
- Action: split or combine modules

# Does Design complexity Matter?



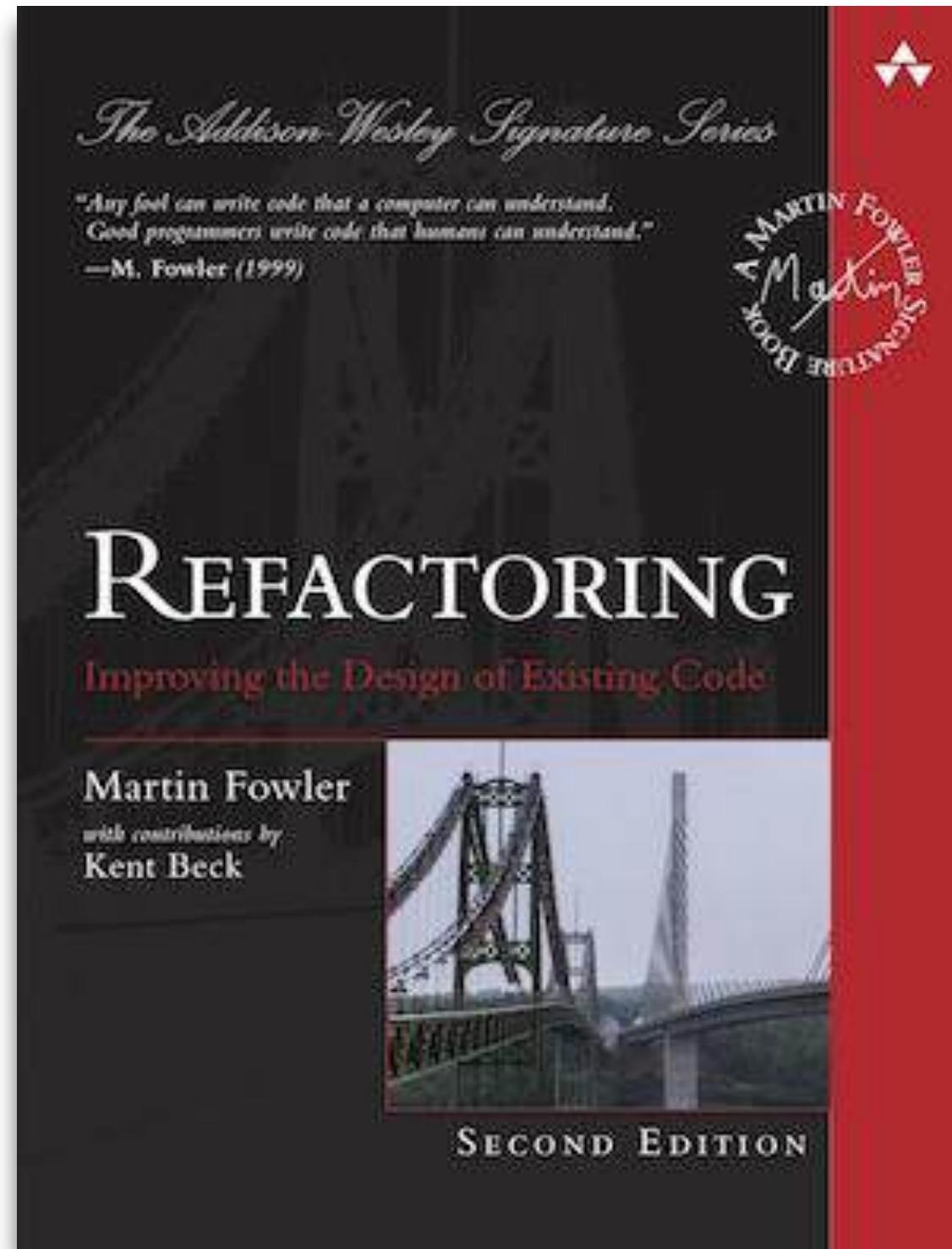
The average person (developer) can hold 5-7 concepts in the short-term memory.

*"The Magical Number Seven, Plus or Minus Two"*,  
George Miller, 1956

When we program, we should target simple design with low complexity (better cohesion, lower coupling)

When the design complexity exceeds what the developers can handle, bugs can occur more often

# Code Smell



“A code smell is a surface indication that usually corresponds to a deeper problem in the system...”

“Firstly a smell is by definition something that's quick to spot (**sniffable**)”

“... smells don't always indicate a problem...”

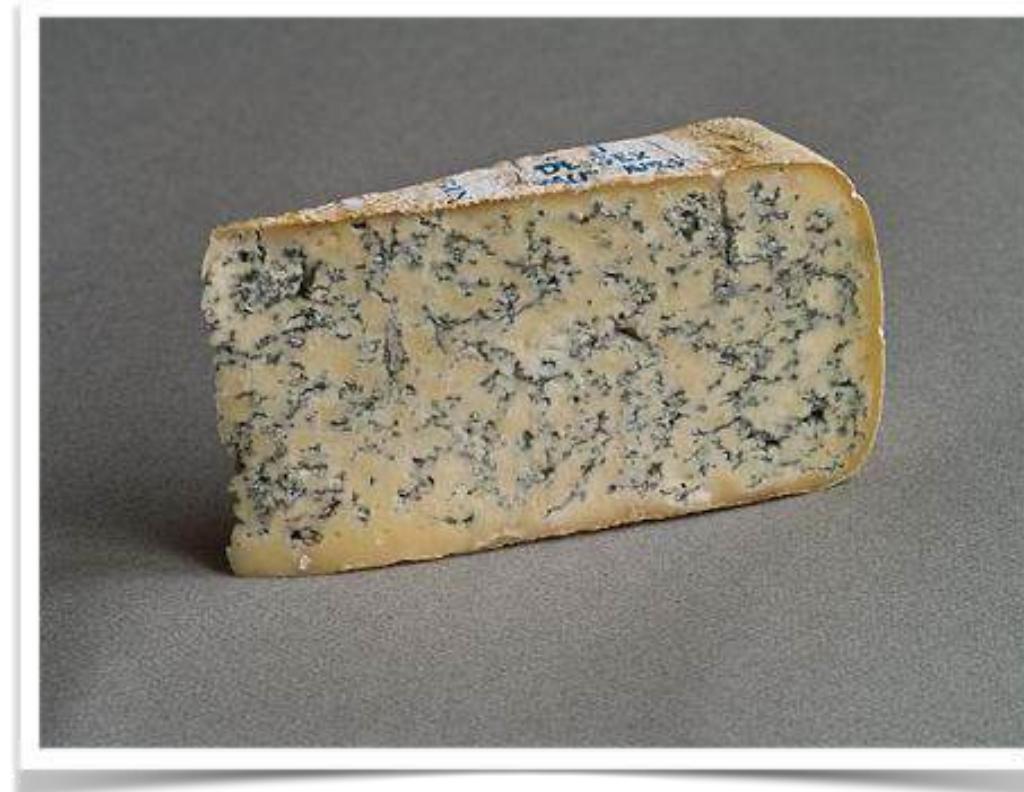
By Martin Fowler  
with Kent Beck

Martin Fowler

# Code Smells

**Code smells are not bugs**, they are not technically incorrect and do not prevent the program from functioning ("maintainability defect" versus "functional defect").

However, **they indicate weaknesses in design** that may slow down development or increase the risk of bugs or failures in the future.



All that smells is  
not bad

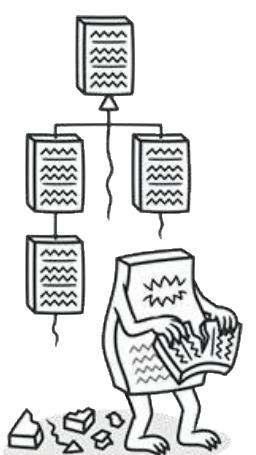


# Categories of Code Smells



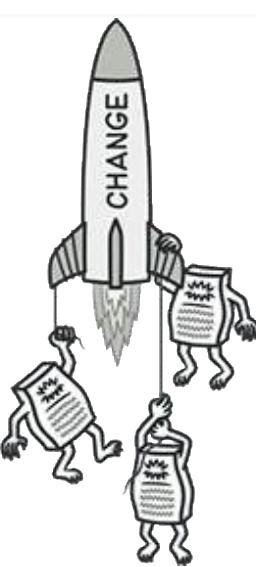
## Blob code:

Method and classes that are too large (blobs) and that are not easy to work with



## Object-oriented abusers:

Incomplete or incorrect applications of object-oriented principles



## Change preventers:

Changes to some parts of the code require to make many changes in other parts too.

# Bad Code Smells



## **Disposable:**

Classes and methods that are pointless and not necessary. Removing dispensables would make the code cleaner.



## **Couplers:**

Classes and methods with very high coupling

# How to detect Code Smells?

# Software Metrics



# Software Metrics

**Measure:** A measure provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of a product or process

**Metric:** A quantitative measure of the degree to which a system, component, or process has a certain attribute

Software metrics are used to:

- Determine the quality of the current product or process
  - Predict qualities of a product or process
  - Improve quality of a product or process

# Software Metrics



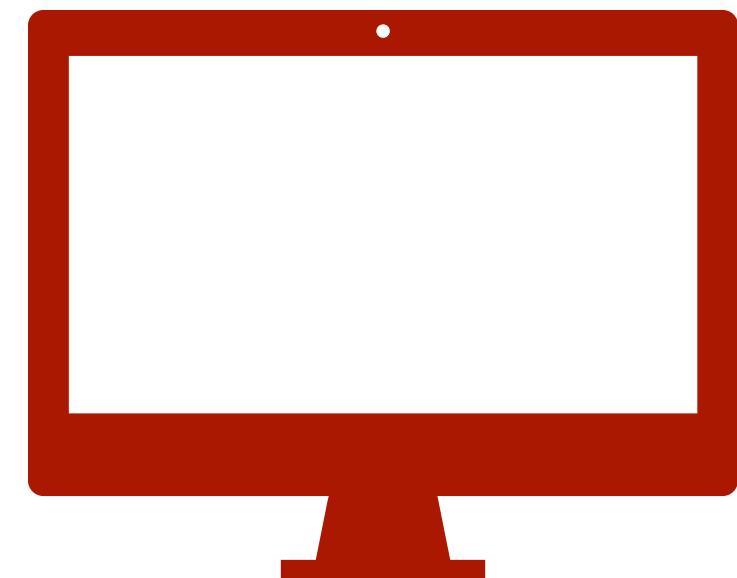
## Process Metrics

Measures for analyzing  
the software  
development process



## Project Metrics

Measure for analyzing the  
quality of the project



## Product Metrics

Measures for analyzing the  
quality of the software  
product

# CK Metrics

476 IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 20, NO. 6, JUNE 1994

## A Metrics Suite for Object Oriented Design

Shyam R. Chidamber and Chris F. Kemerer

**Abstract**— Given the central role that software development plays in the delivery and application of information technology, managers are increasingly focusing on process improvement in the software development area. This demand has spurred the provision of a number of new and/or improved approaches to software development, with perhaps the most prominent being object-orientation (OO). In addition, the focus on process improvement has increased the demand for software measures, or metrics with which to manage the process. The need for such metrics is particularly acute when an organization is adopting a new technology for which established practices have yet to be developed.

This research addresses these needs through the development and implementation of a new suite of metrics for OO design. Previous research on software metrics, while contributing to the field's understanding of software development processes, have generally been subject to one or more types of criticisms. These include: lacking a theoretical basis [41], lacking in desirable measurement properties [47], being insufficiently generalized or too implementation technology dependent [45], and being too labor-intensive to collect [22].

Following Wand and Weber, the theoretical base chosen for the OO design metrics was the ontology of Bunge [5], [6], [43]. Six design metrics were developed, and analytically evaluated against a previously proposed set of measurement principles. An automated data collection tool was then developed and implemented to collect an empirical sample of these metrics at two field sites in order to demonstrate their feasibility and to suggest ways in which managers may use these metrics for process improvement.

**Index Terms**— CR categories and subject descriptors: D.2.8 [software engineering]: metrics; D.2.9 [software engineering]: management; F.2.3 [analysis of algorithms and problem complexity]: tradeoffs among complexity measures; K.6.3 [management of computing and information systems]: software management. General terms: Class, complexity, design, management, measurement, metrics, object orientation, performance.

### I. INTRODUCTION

IT has been widely recognized that an important component of process improvement is the ability to measure the process. Given the central role that software development plays in the delivery and application of information technology, managers are increasingly focusing on process improvement in the software development area. This emphasis has had two effects. The first is that this demand has spurred the provision of a number of new and/or improved approaches to software development, with perhaps the most prominent being object-orientation (OO). Second, the focus on process improvement has increased the demand for software measures, or metrics with which to manage the process. The need for such metrics

**II. RESEARCH PROBLEM**

There are two general types of criticisms that can be applied to current software metrics. The first category are those theoretical criticisms that are leveled at conventional software metrics as they are applied to traditional, non-OO software design and development. Kearney, et al. criticized software complexity metrics as being without solid theoretical bases and lacking appropriate properties [21]. Vessey and Weber also commented on the general lack of theoretical rigor in the structured programming literature [41]. Both Prather and Weyuker proposed that traditional software complexity metrics do not possess appropriate mathematical properties, and consequently fail to display what might be termed normal predictable behavior [34], [47]. This suggests that software

Manuscript received February 17, 1993; revised January, 1994. recommended by S. Zweber. This research was supported in part by the M.I.T. Center for Information Systems Research (CISR), and the cooperation of two industrial organizations who supplied the data.

The authors are with the Massachusetts Institute of Technology, E53-315, 30 Wadsworth Street, Cambridge, MA 02139 USA; e-mail: shyam@athena.mit.edu or ckemerer@sloan.mit.edu.

IEEE Log Number 9401707.

0098-5589/94\$04.00 © 1994 IEEE

S. Chidamber and C. Kemerer

## “A Metric Suite for Object Oriented Design”

IEEE Transactions on Software Engineering (TSE), 1994

A suite of six software metrics for Object-oriented programs:

- **WMC** = Weighted Methods (Complexity) for Class
- **CBO** = Coupling between objects
- **RFC** = Response For Class
- **LCOM** = Lack of Cohesion of Methods
- **DIT** = Depth of Inheritance Tree
- **NOC** = Number of Children

**We will see these metrics in details**

# Does Complexity Matters?

A. Panichella et al.  
SANER 2014

G. Canfora et al.  
ICST 2013

## Cross-Project Defect Prediction Models: L'Union Fait la Force

Annibale Panichella<sup>1</sup>, Rocco Oliveto<sup>2</sup>, Andrea De Lucia<sup>1</sup>  
<sup>1</sup>Department of Management & Information Technology, University of Salerno, Fisciano (SA), Italy  
<sup>2</sup>Department of Bioscience and Territory, University of Molise, Pesche (IS), Italy  
apanichella@unisa.it, rocco.oliveto@unimol.it, adelucia@unisa.it

**Abstract**—Existing defect prediction models use product or process metrics and machine learning methods to identify defect-prone source code entities. Different classifiers (e.g., linear regression, logistic regression, or classification trees) have been investigated in the last decade. The results achieved so far are sometimes contrasting and do not show a clear winner. In this paper we present an empirical study aiming at statistically analyzing the equivalence of different defect predictors. We also propose a combined approach, coined as CODEP (COMbined DEFect Predictor), that employs the classification provided by different machine learning techniques to improve the detection of defect-prone entities. The study was conducted on 10 open source software systems and in the context of cross-project defect prediction, that represents one of the main challenges in the defect prediction field. The statistical analysis of the results indicates that the investigated classifiers are not equivalent and they can complement each other. This is also confirmed by the superior prediction accuracy achieved by CODEP when compared to stand-alone defect predictors.

### I. INTRODUCTION

During software development and maintenance, software engineers need to manage time and resources. Improper management of these factors could lead to project failure. Given its importance, in recent years a lot of effort has been devoted to provide software engineers with tools supporting management activities. Defect prediction is one of these tools. Knowing the parts of a software system that are more defect-prone may aid in scheduling testing or refactoring activities. In fact, it is reasonable to allocate more resources on more critical and defect-prone source code entities.

Existing prediction models use different information (i.e., source code metrics, process metrics, or past defects) and machine learning methods to identify defect-prone source code entities. A plethora of classifiers (e.g., linear regression, logistic regression, or classification trees) have been proposed and empirically investigated in the last decade [1], [2], [3]. Generally, the evaluation of these classifiers is based only on metrics—such as precision and recall—able to capture the prediction accuracy of the experimented approaches.

The achieved results confirmed our conjecture. *The investigated classifiers are not equivalent despite the similar overall prediction accuracy. This means that they are able to identify different sets of defect-prone entities.* This is confirmed by our statistical analysis but especially by the superior performances of a novel combined approach, coined as CODEP (COMbined DEFect Predictor), that uses machine learning techniques to combine different and complementary classifiers.

In summary, the contributions of this paper are:

## Multi-Objective Cross-Project Defect Prediction

Gerardo Canfora<sup>1</sup>, Andrea De Lucia<sup>2</sup>, Massimiliano Di Penta<sup>1</sup>,  
Rocco Oliveto<sup>3</sup>, Annibale Panichella<sup>2</sup>, Sebastiano Panichella<sup>4</sup>  
<sup>1</sup>University of Samo, Via Traiano, 82100 Benevento, Italy  
<sup>2</sup>University of Salerno, Via Ponte don Melillo, 84084 Fisciano (SA), Italy  
<sup>3</sup>University of Molise, Contrada Fonte Lappone, 86090 Pesche (IS), Italy  
<sup>4</sup>{canfora, dipenta, spanichella}@unisamnito.it, {adelucia, apanichella}@unisa.it, rocco.oliveto@unimol.it

**Abstract**—Cross-project defect prediction is very appealing because (i) it allows predicting defects in projects for which the availability of data is limited, and (ii) it allows producing generalizable prediction models. However, existing research suggests that cross-project prediction is particularly challenging and, due to heterogeneity of projects, prediction accuracy is not always very good.

This paper proposes a novel, multi-objective approach for cross-project defect prediction based on a multi-objective logistic regression model built using a genetic algorithm. Instead of providing the software engineer with a single predictive model, the multi-objective approach allows software engineers to choose predictors achieving a compromise between number of likely defect-prone artifacts (effectiveness) and LOC to be analyzed/tested (which can be considered as a proxy of the cost of code inspection).

Results of an empirical evaluation on 10 datasets from the Promise repository indicate the superiority and the usefulness of the multi-objective approach with respect to single-objective predictors. Also, the proposed approach outperforms an alternative approach for cross-project prediction based on local prediction upon clusters of similar classes.

**Keywords**—Cross-project defect prediction, multi-objective optimization, search-based software engineering.

### I. INTRODUCTION

Defect prediction consists of identifying likely defect-prone software components, in order to prioritize quality assurance activities. Indeed, effective defect prediction models can help developers to focus activities such as code inspection or testing on likely defect-prone components, thus optimizing the usage of resources for quality assurance. Existing defect prediction models try to identify defect-prone artifacts based on process or product metrics. For example, Basili *et al.* [1] and Gyimothy *et al.* [2] use Chidamber and Kemerer (CK) metrics [3] to build defect prediction models based on logistic regression. Moser *et al.* [4] use process metrics, e.g., related to the number and kinds of changes occurred on software artifacts. Ostrand *et al.* [5] and Kim *et al.* [6] perform prediction based on knowledge about previously occurred faults.

Building an *effective* (able to identify the largest proportion of defect-prone artifacts) and *efficient* (limiting the number of false positives, thus not wasting the developers' effort in the quality assurance task) bug prediction model

<sup>1</sup>Although local prediction is particularly useful for cross-project prediction, it can also be used to improve performances of within-project prediction.

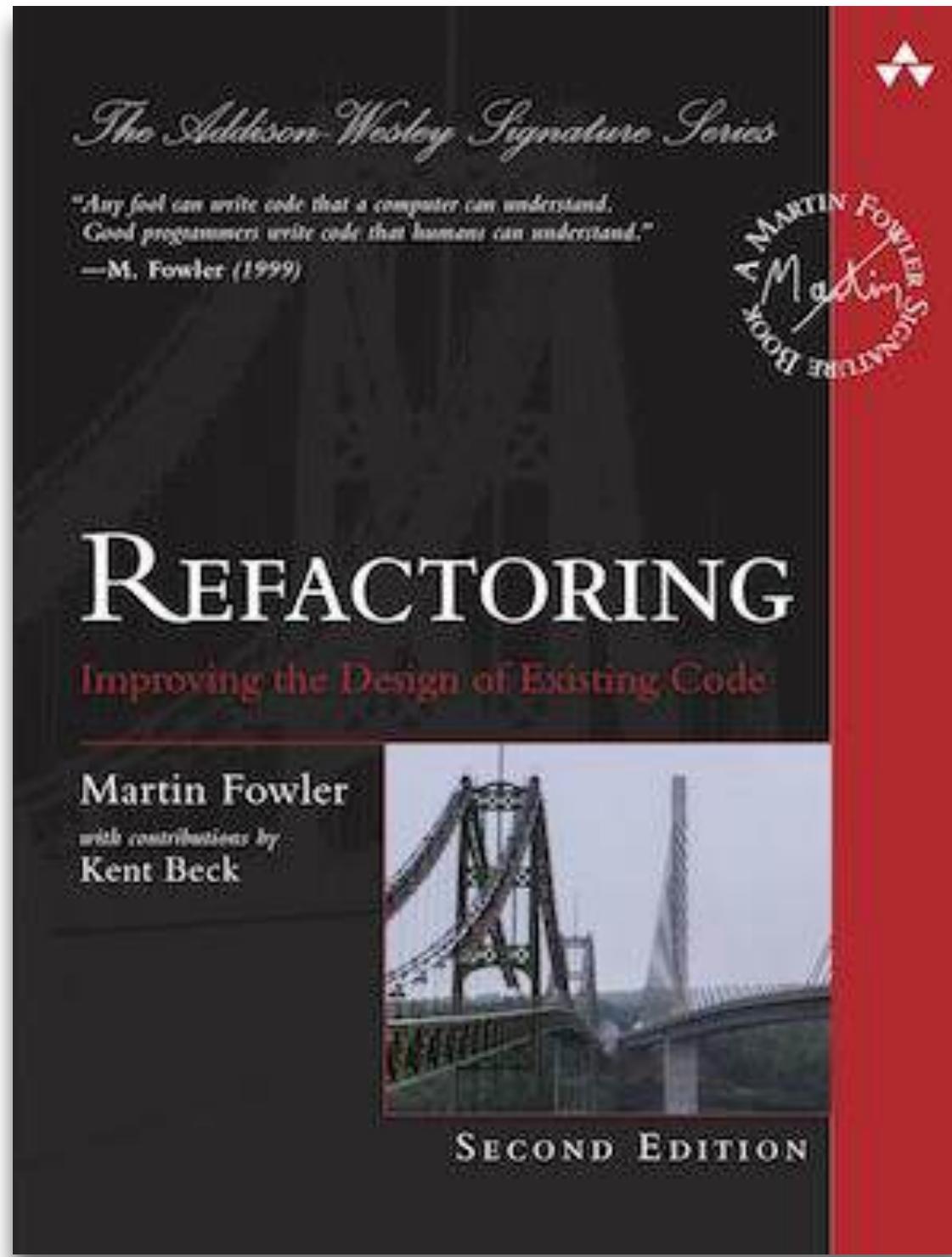
Complexity metrics can be used to **predict** whether a class contains a bug or not.

Simple Machine Learning (ML) models can achieve high accuracy when classifying defective classes.

You can use ML to prioritize which class to test (extra) first

# How Do we fix Code Smells?

# Refactoring



By Martin Fowler  
with Kent Beck

“Refactoring is a controlled technique for improving the design of an existing code base.”

“Its essence is applying a series of small behavior-preserving transformations, each of which **too small to be worth doing**.”

“By doing them in small steps you reduce the risk of introducing errors.”

Martin Fowler

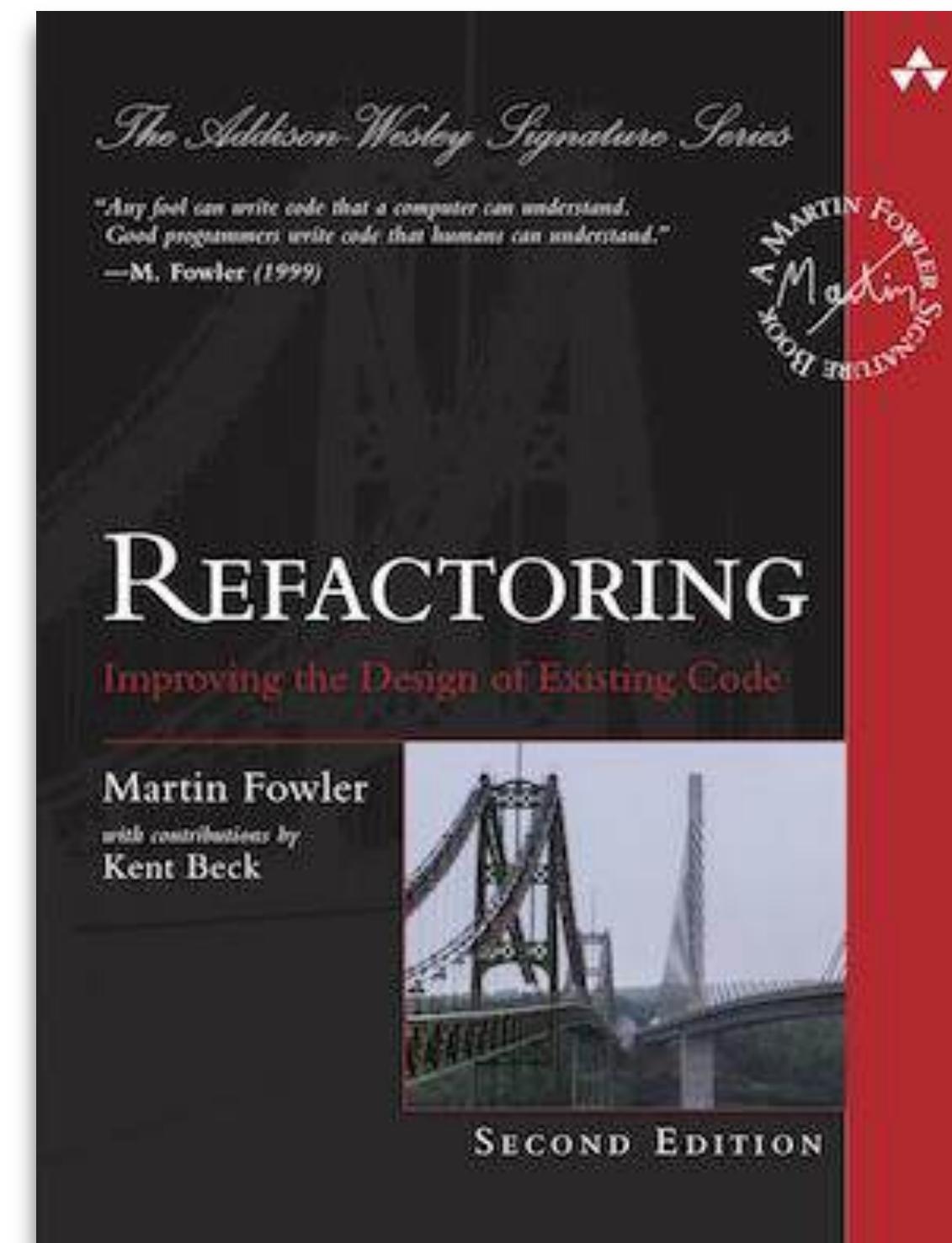
# Refactoring

- **Refactoring** (noun): a change made to the internal structure of software to make it easier to understand and cheaper to modify *without changing its observable behavior*.
- **Refactor** (verb): to restructure software by applying a series of refactoring operations *without changing its observable behavior*.

GOAL: Improving non-functional properties of the software under analysis

# Common Refactoring Techniques

- Extract Class
- Extract method
- Introduce Parameter Object
- Move Method
- Replace Delegate With Inheritance
- Replace Conditional with Polymorphism



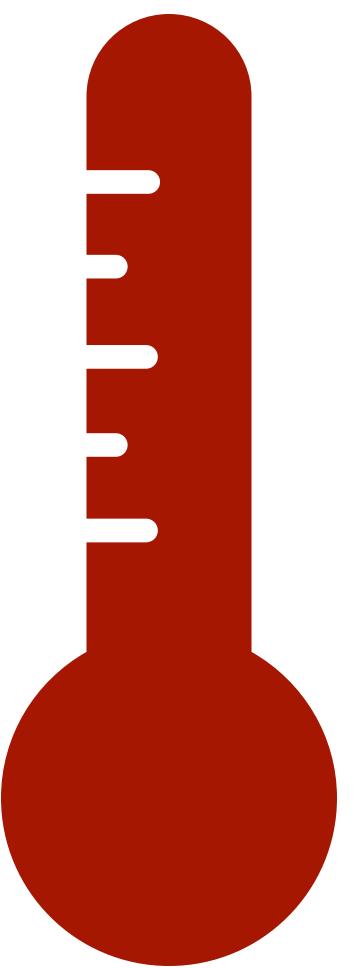
More than 20  
refactoring operations

# Code Smells



(Symptoms)

# Software Metrics



(Thermometer)

# Refactoring



(How to fix)

# Code Examples

<https://github.com/apache/commons-lang>

The screenshot shows the GitHub repository page for `apache / commons-lang`. The page includes navigation tabs for Code, Pull requests (22), Projects (0), and Insights. Key statistics are displayed: 5,335 commits, 8 branches, 84 releases, 106 contributors, and Apache-2.0 license. A commit list shows recent changes, including a merge from `mureinik` and `PascalSchumacher` to clean up assertions. The repository has 167 watchers, 1,457 stars, and 814 forks.

Mirror of Apache Commons Lang

5,335 commits 8 branches 84 releases 106 contributors Apache-2.0

Branch: master New pull request Create new file Upload files Find file Clone or download

`mureinik` and `PascalSchumacher` Clean up assertions (closes #376) ... Latest commit 3609993 2 days ago

`src` Clean up assertions (closes #376) 2 days ago

`.gitattributes` Fixed line ending issues once and for all 3 years ago

`.gitignore` Ignore eclipse-cs generated file 2 years ago

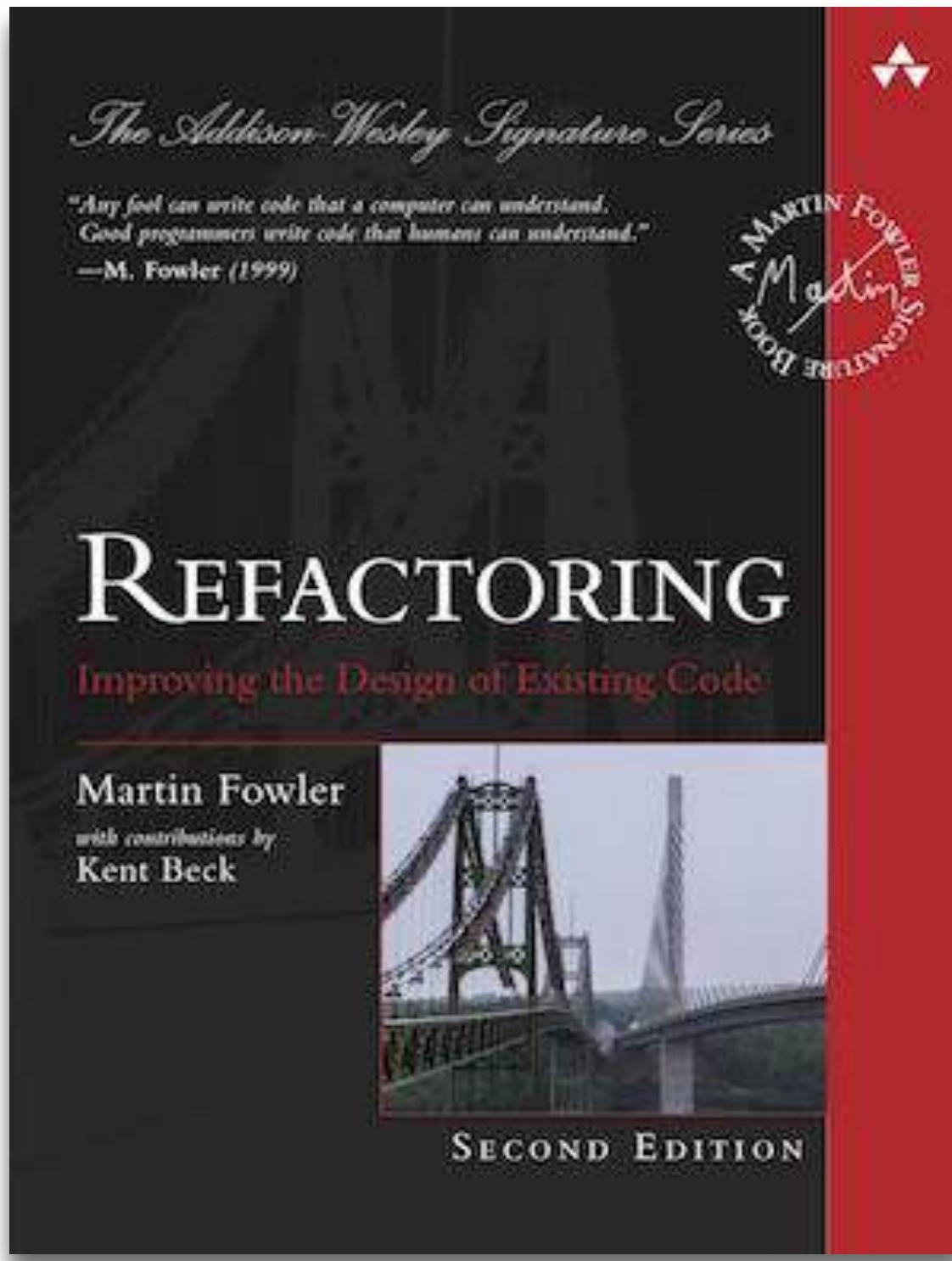
`.travis.yml` Travis: Also build with openjdk-ea, but allow openjdk-ea builds to fail. 9 days ago

`CONTRIBUTING.md` Prepare for release 3.8. 3 months ago

`LICENSE.txt` Add blank line 15 years ago

`NOTICE.txt` Prepare for release 3.8. 3 months ago

# Code Examples



Other examples are taken from the book  
Chapters 5, 6, 7, 8

# Tool: CKJM-extended

<https://github.com/mjureczko/CKJM-extended>

[http://gromit.iiar.pwr.wroc.pl/p\\_inf/ckjm/intro.html](http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/intro.html)

mjureczko / CKJM-extended

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights

An extended version of Tool for Calculating Chidamber and Kemerer Java Metrics (and many other metrics)

#3 fixing 'Value mismatch of CC in the generated XML file'  
- test case identifying the issue

1 branch 1 release 1 contributor View license

Branch: master New pull request Create new file Upload files Find file Clone or download

mjureczko Download section for version 2.2. Latest commit dd03b8c on Jul 26, 2015

.idea/libraries	#3 fixing 'Value mismatch of CC in the generated XML file'	3 years ago
doc	Download section for version 2.2.	3 years ago
src	#4 fix in XML printer	3 years ago
LICENSE.txt	Copied source from old repository.	3 years ago
README.md	#4 project web page	3 years ago
manifest.mf	Copied source from old repository.	3 years ago
pom.xml	[maven-release-plugin] prepare for next development iteration	3 years ago

# Tool: CKJM-extended

## Command Line

```
java -jar ckjm_ext.jar <path to your jar file>
```

```
java -jar ckjm_ext.jar <path to the class files>*.class
```

## Output

*Metrics at class level*

```
test.TestClass 3 1 0 1 8 1 1 1 1 0 0000 115 0 0000 0 0 0000 0 5556 0 0 37.0000
```

```
~ void m2(): 6
```

```
~ public void <init>(): 1
```

```
~ void m1(): 7
```

*Cyclomatic Complexity of each  
method in the class*

```
test.TestClass2 5 1 0 1 11 4 1 1 0 0.4167 53 0.3333 1 0.0000 0.6250 0 0 9.0000
```

```
~ int m3(int jk): 1
```

```
~ void <init>(): 1
```

```
~ void m2(): 1
```

```
~ static void <clinit>(): 1
```

```
~ void m1(): 2
```

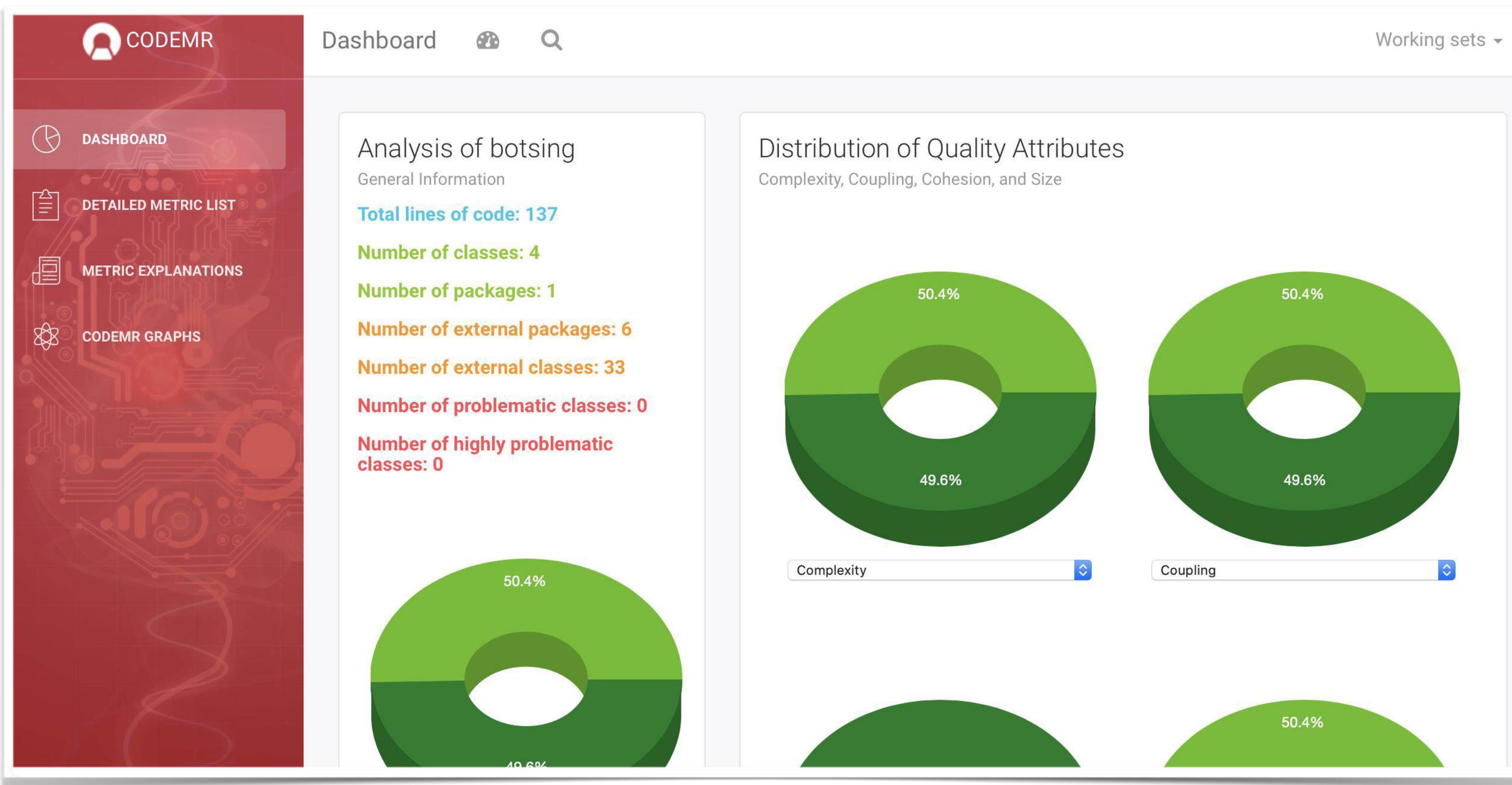
# CodeMR (IntelliJ)

<https://plugins.jetbrains.com/plugin/10811-codemr/>

The screenshot shows the CodeMR plugin page on the JetBrains plugin marketplace. The top navigation bar includes links for IDEs, .NET, Team Tools, Dev Guide, Sign In, and a search icon. Below the navigation is a section titled "Inspection" featuring the "CodeMR" logo, a five-star rating, and the developer name "CodeMR". A large blue "Get" button is prominently displayed. To the right of the button, the text "Compatible with all IntelliJ-based IDEs" is visible. Below this, there are tabs for "Overview" and "Versions", with "Overview" being the active tab. Two large windows below the tabs display network graphs of code dependencies. The left window shows a dense graph with many nodes and connections, while the right window shows a more sparse, circular cluster of nodes.

# CodeMR (IntelliJ)

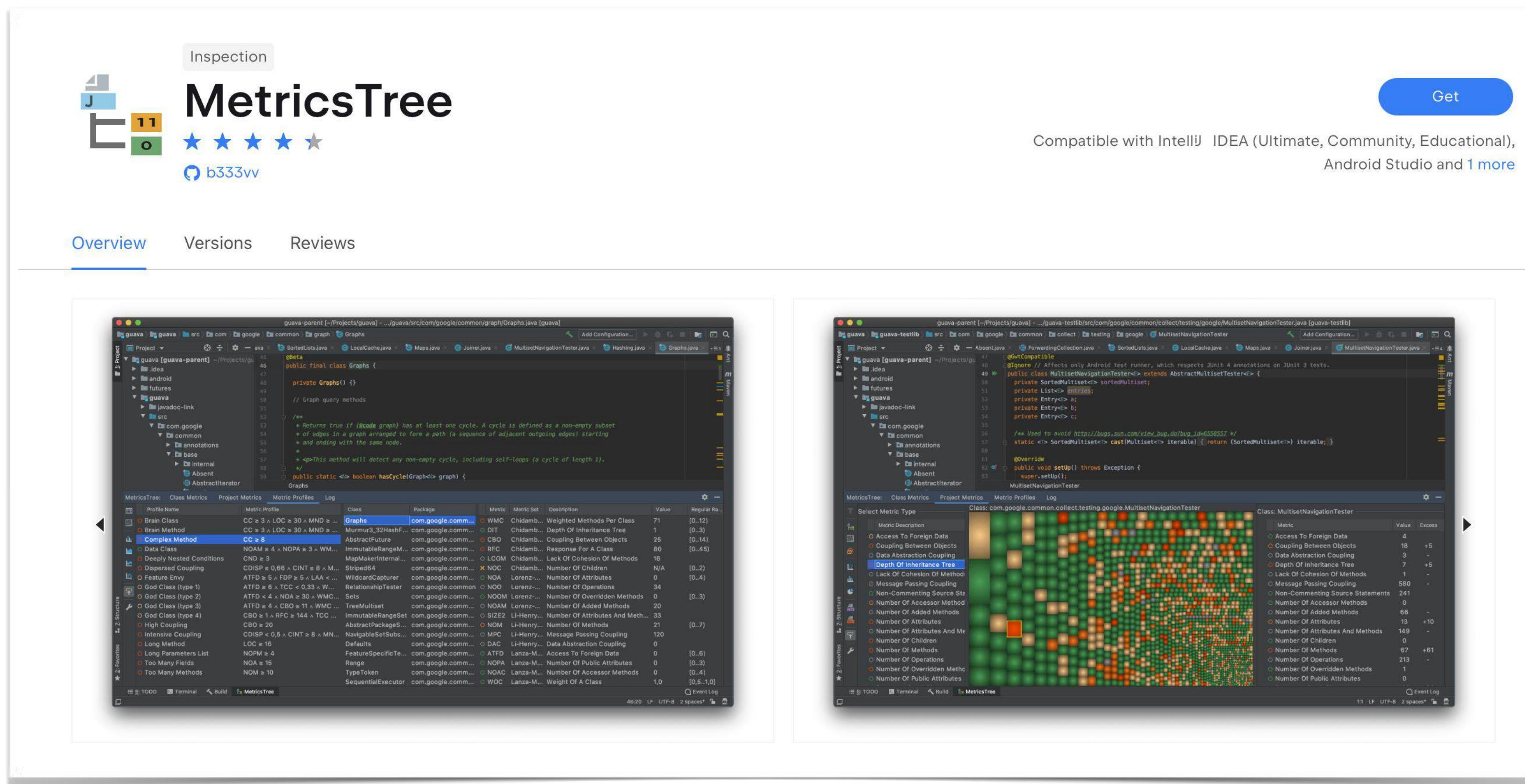
<https://plugins.jetbrains.com/plugin/10811-codemr/>



Recommended

# MetricsTree (IntelliJ)

<https://plugins.jetbrains.com/plugin/13959-metricstree>



Let's get Started... .



# Blob code

# Code Smell: Long Method

- A long method is a method that has too many lines of code (to read, maintain, and test)
- The longer the method, the harder is to understand, change and reuse it
  - Often a long method tries to do >1 thing (goes against single responsibility principle)
- What metrics should we use to measure size?

# Measuring Code Size

GOAL: Size of the code

**LOC:** Lines Of Code

**KLOC:** 1000 Lines Of Code

**eLOC:** Effective Lines Of Code

Pros: Easy to use, Easy to compute

Cons: Language and programmer dependent

# Lines Of Code

Java

```
public class HelloWorldApp {  
    public static void main(String args[]) {  
        System.out.println("Hello World!");  
    }  
}
```

LOC = 5

Python

```
print "Hello World!"
```

LOC = 1

C

```
#include <iostream.h>  
main() {  
    cout << "Hello World!";  
}  
return 0;
```

LOC = 4

# Lines Of Code

Java

```
public class HelloWorldApp {  
    public static void main(String args[]) {  
        System.out.println("Hello World!");  
    }  
}
```

LOC = 5

eLOC = 1

Java

```
public class HelloWorldApp  
{  
    public static void main(String args[])  
    {  
        System.out.println("Hello World!");  
    }  
}
```

LOC = 7

eLOC = 1

# Measuring Code Complexity

**GOAL:** Evaluating the complexity of the code

**CC:** Cyclomatic Complexity / McCabe's number

The number of independent linear paths in the source code

**Pros:** It can determine the minimal number of white box test cases needed to reach high code coverage

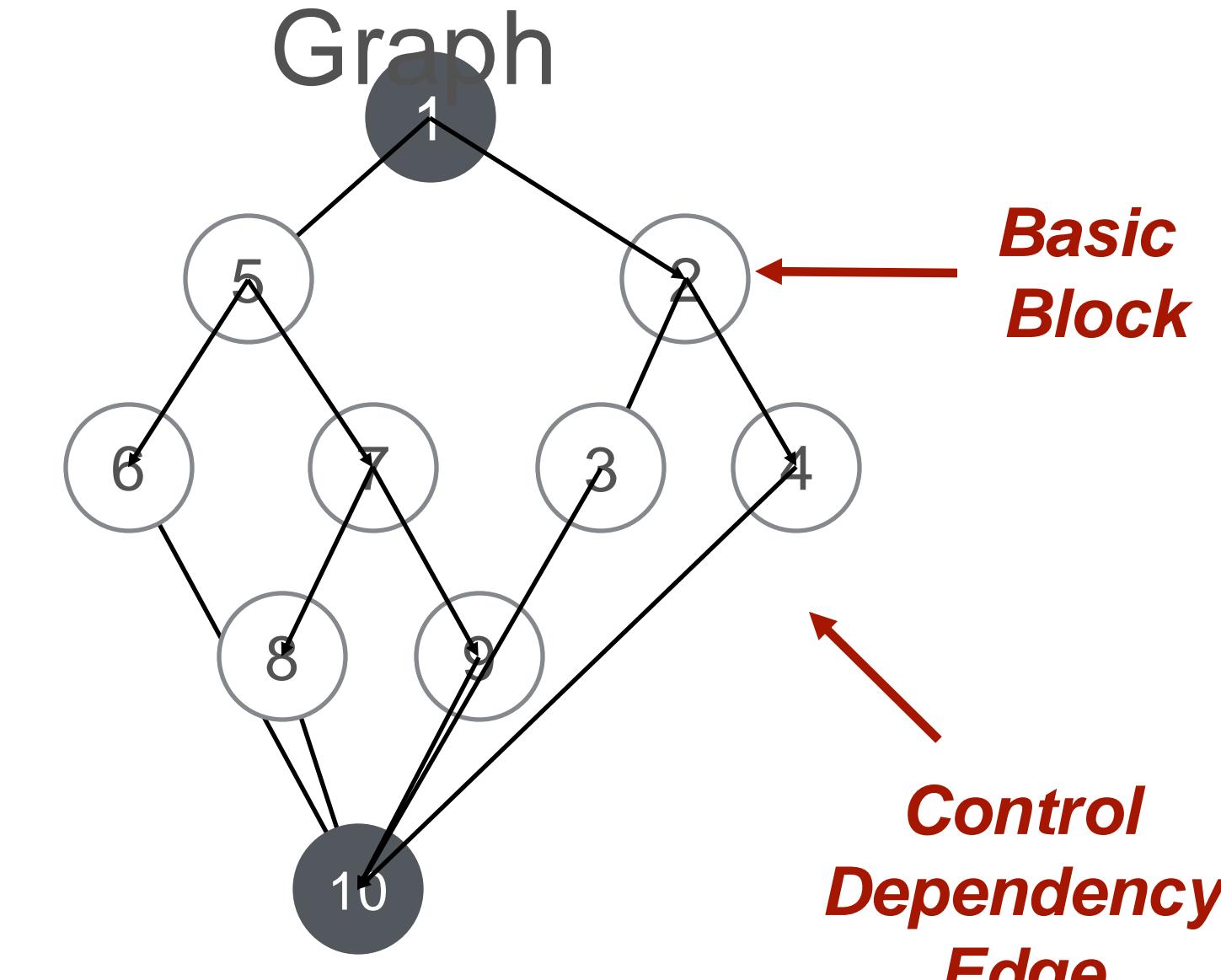
**Cons:** Not so easy to compute

# McCabe's Cyclomatic Complexity (CC)

## Source Code

```
class Triangle {  
    int a, b, c; //sides  
    String type = "NOT_TRIANGLE";  
  
    Triangle(int a, int b, int c) {  
    }  
  
    void computeTriangleType() {  
        1. if(a==b){  
        2. ...if(b==c)...  
        3. ....type="EQUILATERAL";  
            else  
        4. ....type="ISOSCELES";  
        } else {  
        5. if(a==c){  
        6. ....type="ISOSCELES";  
            } else {  
        7. if(b==c)...  
        8. ....type="ISOSCELES";  
            else  
        9. ....type="SCALENE";  
            }  
        }  
    }  
}
```

## Control Flow Graph



McCabe's metric is based on a control flow representation of the code

# McCabe's Cyclomatic Complexity (CC)

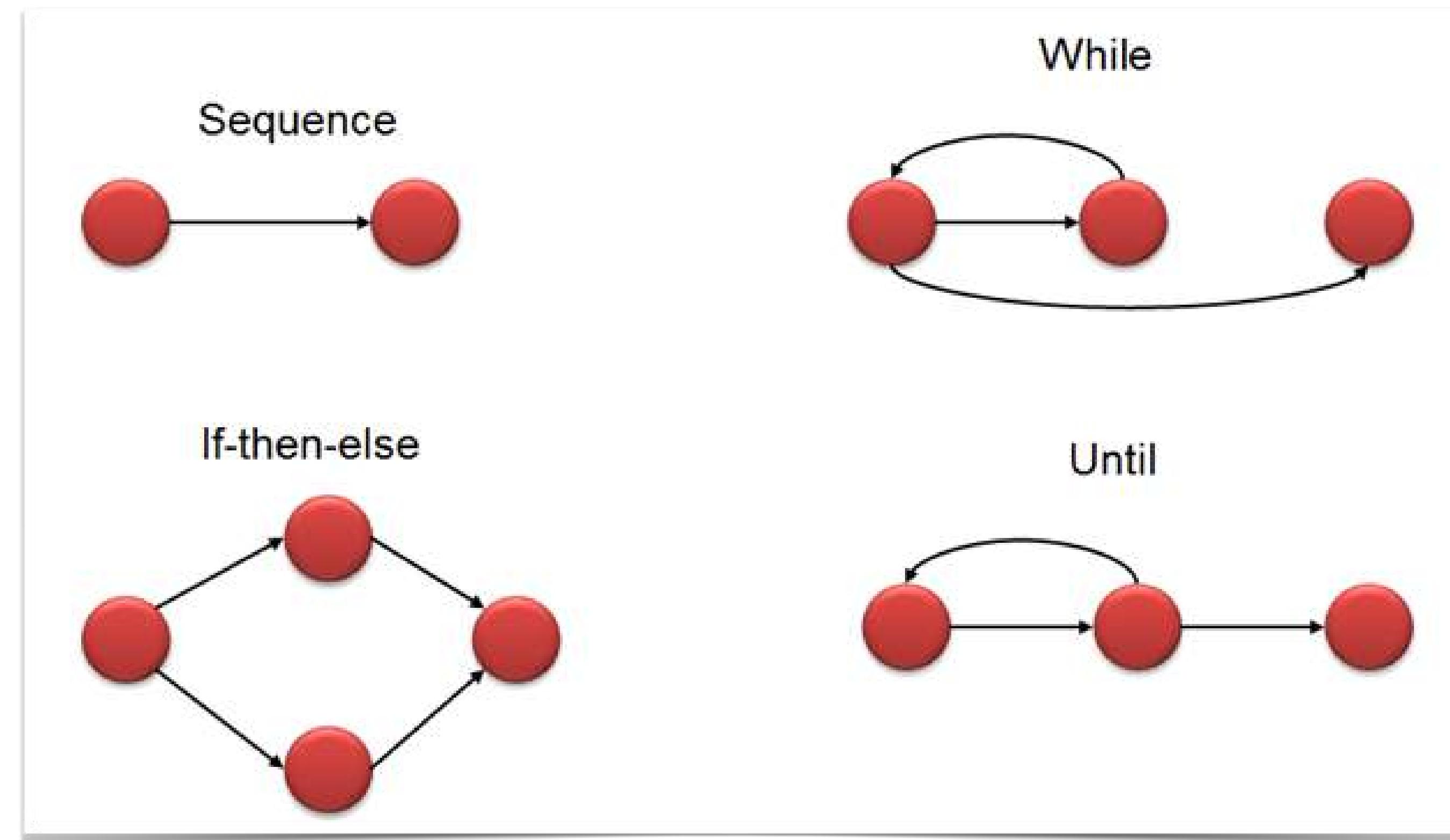
**Control Flow Graph:** it is a graphical representation of all paths that might be traversed through a program during its executions. Nodes in the graph are basic blocks while edges represent jumps in the control flow.

**Basic Block:** it is a straight-line code sequence with no branches in except to the entry and no branches out except at the exit

**Dependency edge:** it represents a jump in the control flow

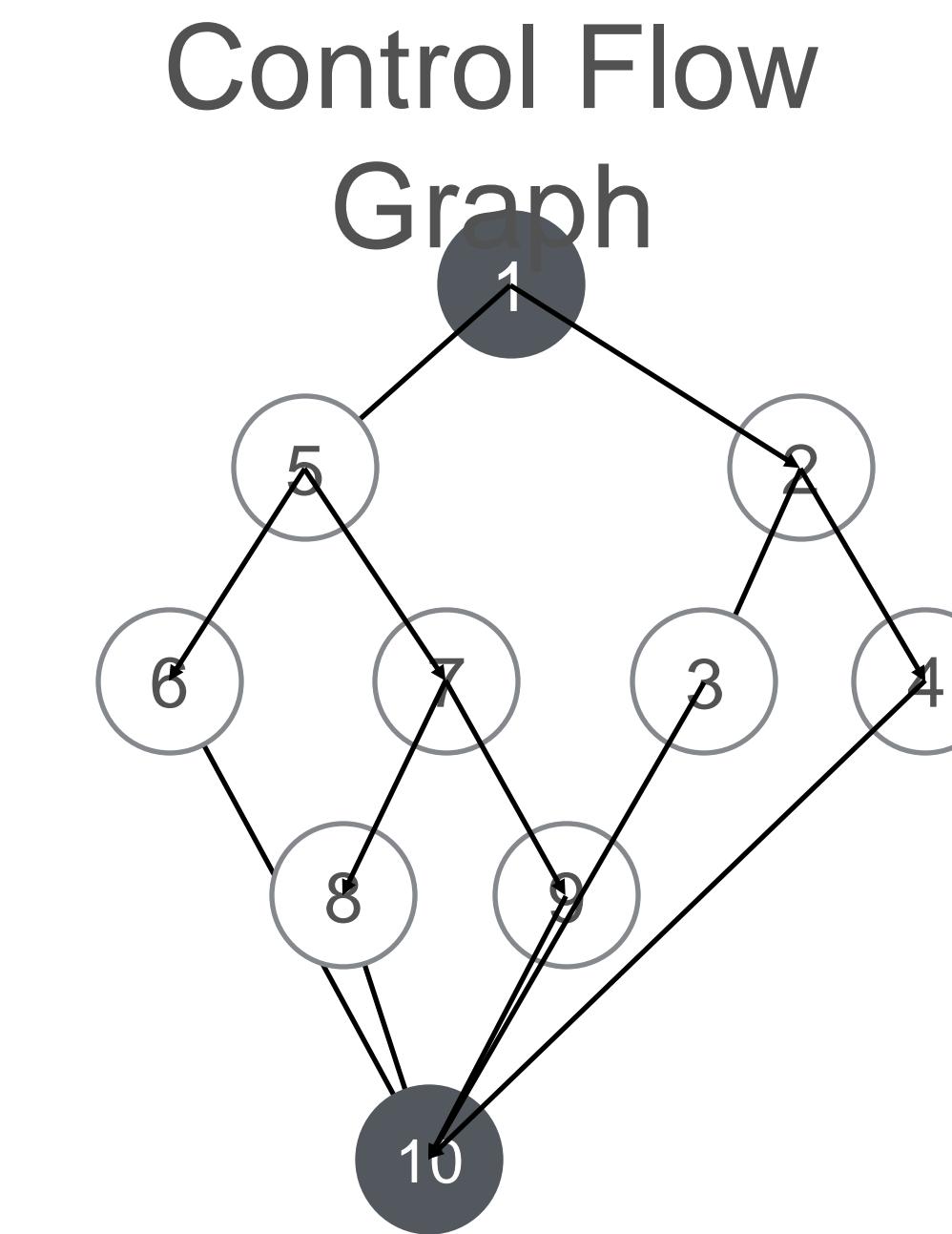
**White-box testing:** the control flow graph is often used to define testing adequacy criteria (e.g., branch coverage) in white-box testing

# McCabe's Cyclomatic Complexity (CC)



# McCabe's Cyclomatic Complexity (CC)

McCabe's metric: the number of independent linear paths in the code



# McCabe's Cyclomatic Complexity (CC)

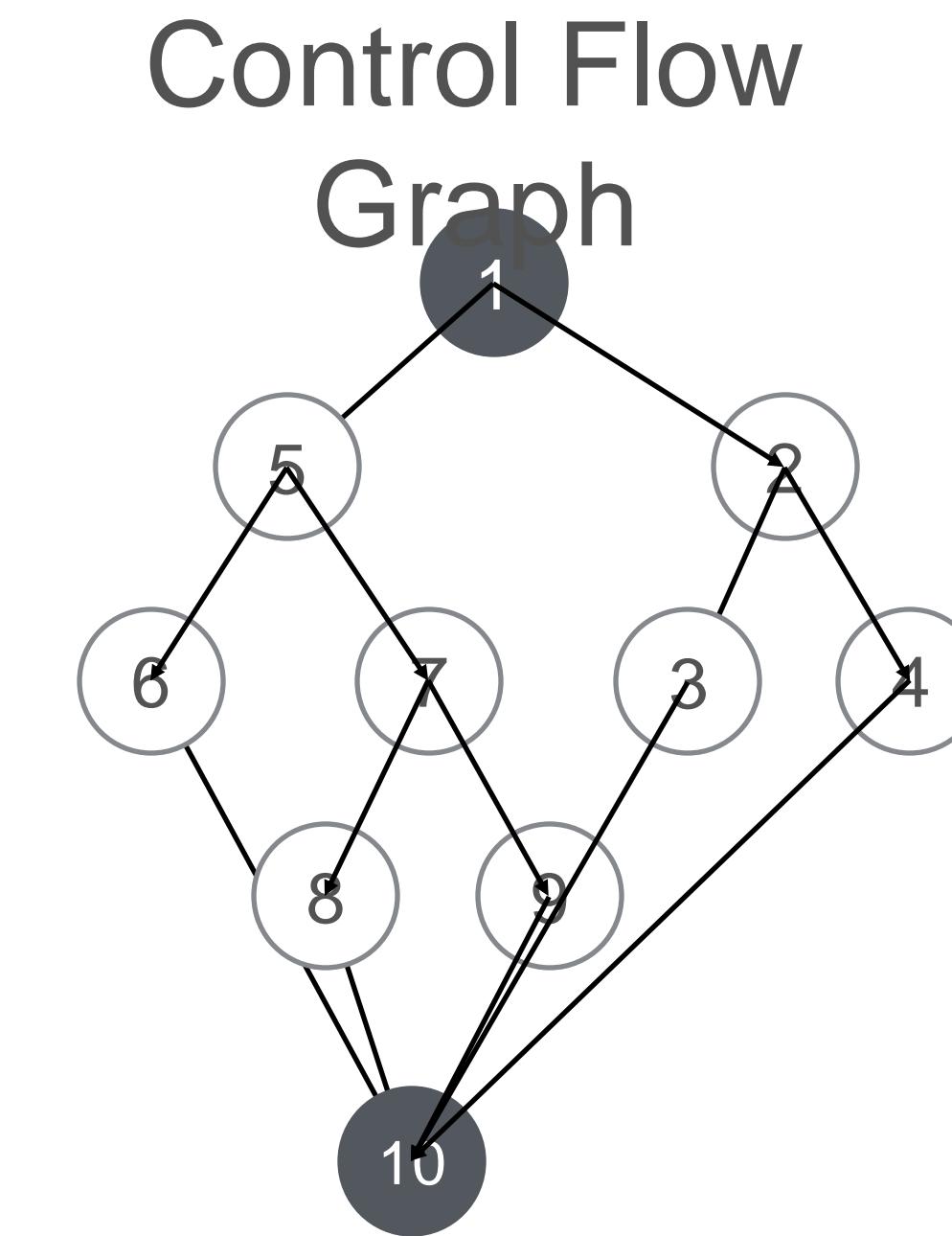
McCabe's metric: the number of independent linear paths in the code

How to compute it:

$$1. V(G) = E - N + 2$$

where  $E = \#$  of edges

$N = \#$  of nodes



*In our example:*  
 $V(G) = 13 - 10 + 2 = 5$

# McCabe's Cyclomatic Complexity (CC)

McCabe's metric: the number of independent linear paths in the code

How to compute it:

$$1. V(G) = E - N + 2$$

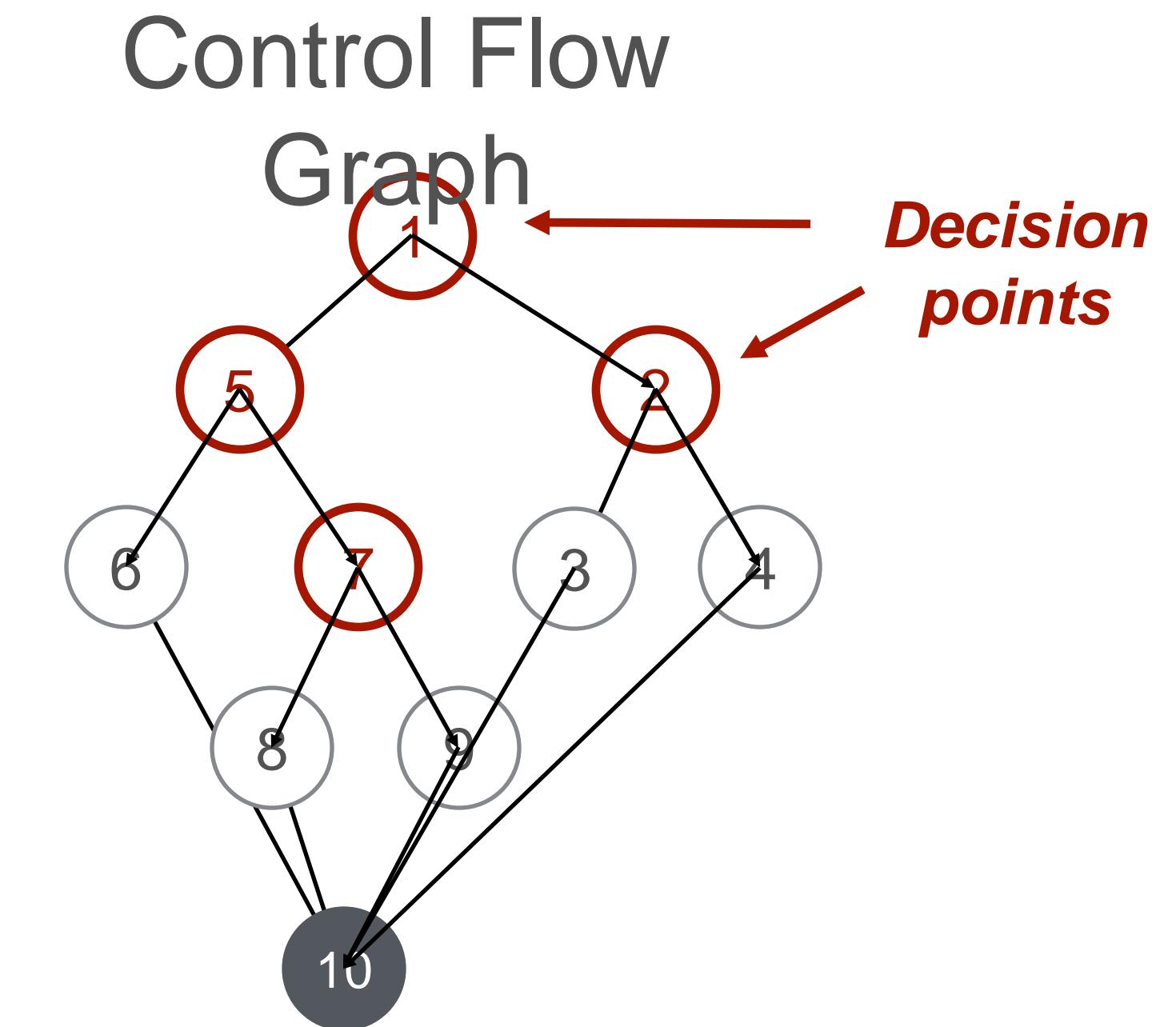
where  $E = \# \text{ of edges}$   
 $N = \# \text{ of nodes}$

$$2. V(G) = D + 1$$

where  $D = \# \text{ Decision points}$

*In our example:*  
 $V(G) = 13 - 10 + 2 = 5$

*In our example:*  
 $V(G) = 4 + 1$



# Real-world Example

Class = org.apache.commons.lang3.time.DurationFormatUtils

Method = formatPeriod(...)

LOC = 132

CC = 24

```
public static String formatPeriod(final long startMillis, final long endMillis, final String format, final boolean padWithZeros,
                                 final TimeZone timezone) {
    Validate.isTrue(startMillis <= endMillis, "startMillis must not be greater than endMillis");

    final Token[] tokens = lexx(format);

    // timezones get funky around 0, so normalizing everything to GMT
    // stops the hours being off
    final Calendar start = Calendar.getInstance(timezone);
    start.setTime(new Date(startMillis));
    final Calendar end = Calendar.getInstance(timezone);
    end.setTime(new Date(endMillis));

    // initial estimates
    int milliseconds = end.get(Calendar.MILLISECOND) - start.get(Calendar.MILLISECOND);
    int seconds = end.get(Calendar.SECOND) - start.get(Calendar.SECOND);
    int minutes = end.get(Calendar.MINUTE) - start.get(Calendar.MINUTE);
    int hours = end.get(Calendar.HOUR_OF_DAY) - start.get(Calendar.HOUR_OF_DAY);
    int days = end.get(Calendar.DAY_OF_MONTH) - start.get(Calendar.DAY_OF_MONTH);
    int months = end.get(Calendar.MONTH) - start.get(Calendar.MONTH);
    int years = end.get(Calendar.YEAR) - start.get(Calendar.YEAR);

    // each initial estimate is adjusted in case it is under 0
    while (milliseconds < 0) {
        milliseconds += 1000;
        seconds -= 1;
    }
    if (seconds < 0) {
        seconds += 60;
        minutes -= 1;
    }
    if (minutes < 0) {
        minutes += 60;
        hours -= 1;
    }
    if (hours < 0) {
        hours += 24;
        days -= 1;
    }
    if (days < 0) {
        days += 30;
        months -= 1;
    }
    if (months < 0) {
        months += 12;
        years -= 1;
    }
}
```

# Real-world Example

Class = org.apache.commons.lang3.time.DurationFormatUtils

Method = formatPeriod(...)

LOC = 132

CC = 24

Long method this is...  
Extract methods you must



Long method...

- Difficult to understand – and maintain
- Difficult to test

# Extract Method Refactoring

- Apply to methods that have code fragments that can be grouped together and moved into new methods
- General Mechanism:
  1. Create a new method, and name it after the intention of the method
  2. Copy the extracted code from the source method into the new target method
  3. Add parameters and temporary variables to the new methods
  4. Replace the extracted code in the source method with a call to the target method.
  5. Update (if needed) and run your tests

# Extract Method Refactoring

## Long Method

```
public static String formatPeriod(...) {
    Validate.isTrue(startMillis <= endMillis, "...");

    final Token[] tokens = lex(tokens);
    // timezones get funky around 0, so normalizing everything to GMT
    // stops the hours being off
    ...

    // initial estimates
    ...
    // each initial estimate is adjusted in case it is under 0
    ...

    if (Token.containsTokenWithValue(tokens, M)) {
        ...
    } else {
        // there are no M's in the format string
        ...
    }

    // The rest of this code adds in values that
    // aren't requested. This allows the user to ask for the
    // number of months and get the real count and not just 0->11.
    ...

    return ...
}
```

Usually, a long method is chopped up in blocks separated by comments.

These blocks represent multiple responsibilities.

Let us consider as an example the long method we detected in DurationFormatUtils.java

# Code Smell: large Class

- A Large class (or blob) implements too many functionalities (or too many responsibilities). It violates the single-responsibility principle.
- Characteristics: very long (size), too many methods, and many instance variables
- Hard to be maintained/controlled
- When a class has too many instance variables, some duplicated code cannot be far behind

# Software Metrics

- Software metrics useful to detect blobs:
  - Lines of Code (LOC)
  - Cyclomatic complexity (CC)
  - Number of methods
  - Number of attributes
  - Number of instance variables
  - Cohesion
  - Coupling

# Measuring Cohesion

**GOAL:** Evaluating how much a class is cohesive, i.e., if it implements a single responsibility

**LCOM:** Lack of Cohesion of Methods

**CM:** Connectivity Metric

**Pros:** Different metrics identify different cohesion aspects

**Cons:** Not so easy to compute

# Lack of Cohesion of Methods

$$\text{LCOM} = \begin{cases} |P| - |Q| & \text{if } |P| > |Q| \\ 0 & \text{otherwise} \end{cases}$$

$$P = \{(m_i, m_j) \mid A_i \cap A_j = \emptyset\}$$

# method pairs that do not share any attribute

$$Q = \{(m_i, m_j) \mid A_i \cap A_j \neq \emptyset\}$$

# method pairs that share at least one attribute

**Main idea:** methods of a **cohesive** class should share (interact with) common attributes

# Lack of Cohesion of Methods

What is the LCOM of the following class?

```
public class NumberManipulator {
    private int _number;

    public NumberManipulator(int value) {
        this._number = value;
    }

    public void AddOne() {
        this._number++;
    }

    public void SubtractOne() {
        this._number--;
    }
}
```

Method pairs	Share Attributes	Do not share Attributes
NumberManipulator AddOne	X	
NumberManipulator SubtractOne	X	
AddOne SubtractOne	X	
Sum	3	0

$$\text{LCOM} = \begin{cases} |P| - |Q| & \text{if } |P| > |Q| \\ 0 & \text{otherwise} \end{cases} = 0$$

# Lack of Cohesion of Methods

```

public class Person {
    protected String name;
    protected String surname;
    protected String phone;
    protected String creditCard;

    public Person(String pName, String surname) {
        this.name = pName;
        this.surname = pName;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    public void setCreditCard(String creditCard) {
        this.creditCard = creditCard;
    }

    public boolean canPay(){
        if (creditCard == null || creditCard == "") {
            return false;
        }

        if (phone == null || phone == "") {
            return false;
        }

        return true;
    }
    ...
}

```

$$\# \text{Pairs} = 4! / (2! * 2!) = 6$$

(Binomial Coefficient)

Method pairs	Share Attributes	Do not share Attributes
Person + setPhone		✗
Person + setCreditCard		✗
Person + canPay		✗
setPhone + setCreditCard		✗
setPhone + canPay	✗	
setCreditCard + canPay	✗	
Sum	Q	P

$$LCOM = \begin{cases} |P| - |Q| & \text{if } |P| > |Q| \\ 0 & \text{otherwise} \end{cases}$$

= 2

# Connectivity Metric

# method pairs sharing attributes or  
having method calls among them

*Connectivity Metric (CM) =*

# method pairs in the class

High CM = High Cohesion



Low CM = Low Cohesion



**CM considers both  
attributes and method  
calls to check for a  
cohesion of the class**

# Real-world Example

**Class** = org.apache.commons.lang3.ArrayUtils

**# of Methods** = 321

**LOC** = 8748

**CC** = 1096 (sum of all methods complexity)

**LCOM** = 51136

```
public class ArrayUtils {  
  
    /**  
     * An empty immutable {@code Object} array.  
     */  
    public static final Object[] EMPTY_OBJECT_ARRAY = new Object[0];  
    /**  
     * An empty immutable {@code Class} array.  
     */  
    public static final Class<?>[] EMPTY_CLASS_ARRAY = new Class[0];  
    /**  
     * An empty immutable {@code String} array.  
     */  
    public static final String[] EMPTY_STRING_ARRAY = new String[0];  
    /**  
     * An empty immutable {@code long} array.  
     */  
    public static final long[] EMPTY_LONG_ARRAY = new long[0];  
    /**  
     * An empty immutable {@code Long} array.  
     */  
    public static final Long[] EMPTY_LONG_OBJECT_ARRAY = new Long[0];  
    /**  
     * An empty immutable {@code int} array.  
     */  
    public static final int[] EMPTY_INT_ARRAY = new int[0];  
    /**
```

# Real-world Example

**Class** = org.apache.commons.lang3.ArrayUtils

**# of Methods** = 321

**LOC** = 8748

**CC** = 1096 (sum of all methods complexity)

**LCOM** = 51136

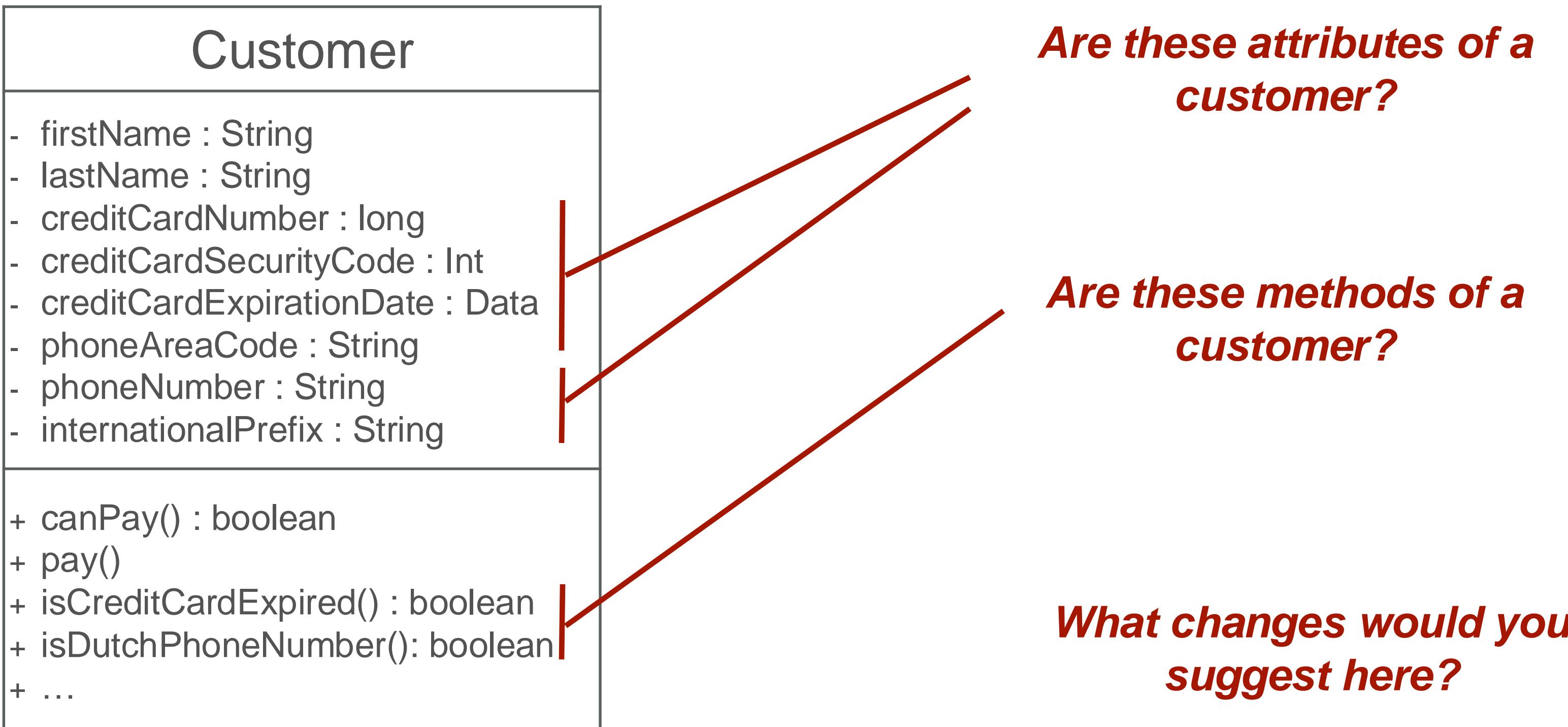
A blob this is...  
Extract class you must



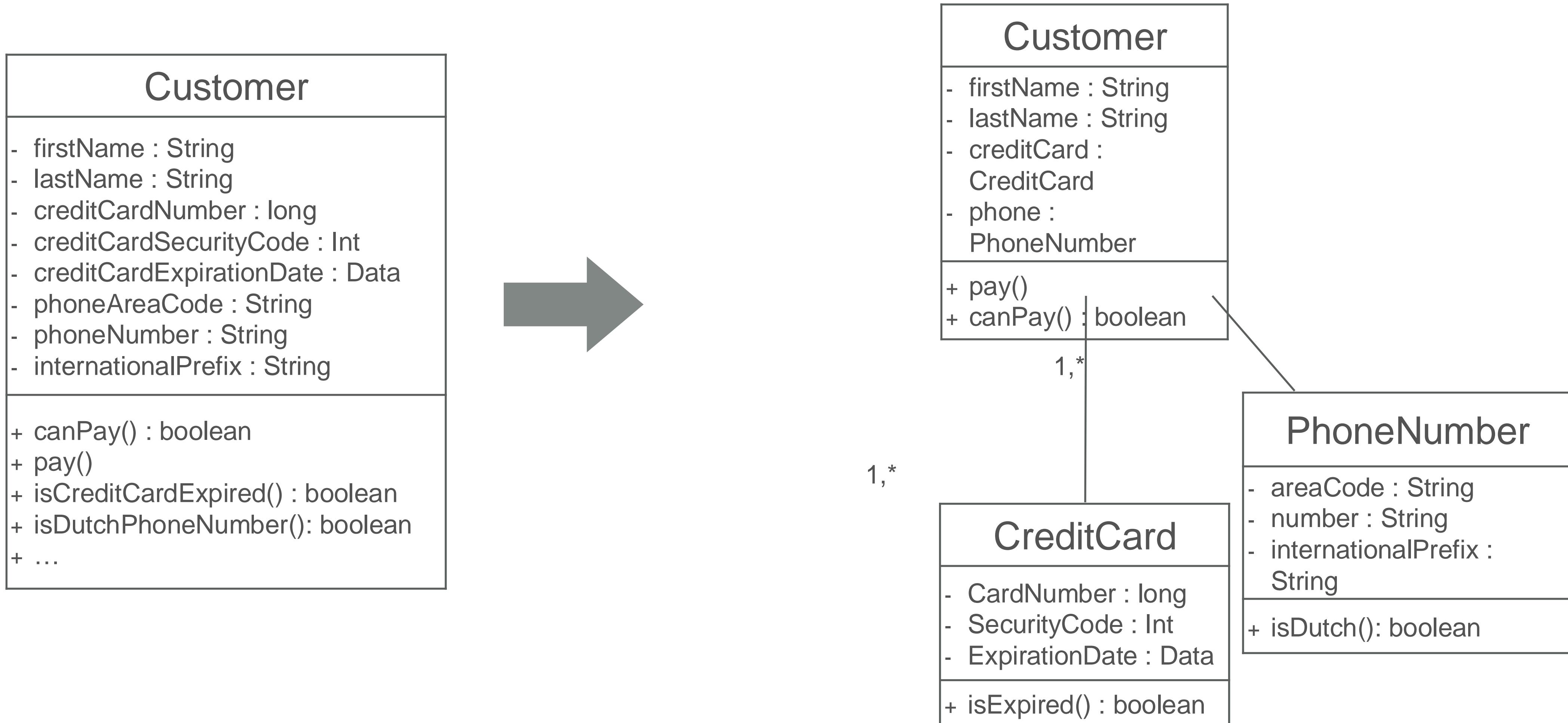
# Extract Class Refactoring

- Apply to classes with too many responsibilities and that should be split up in several classes
- General Mechanism:
  1. Decide how to split the responsibilities of the class
  2. Create a new class to express the split-off responsibilities
  3. Make a link from the old to the new class
  4. Use **Move Method** refactoring to move methods over from old to new
  5. Update and run your tests

# Extract Class Refactoring



# Extract Class Refactoring



# Code Smell: Long Parameter List

- A long parameter list is a method with too many input parameters
- Methods with more than 4 parameters should be considered as **Long parameters** lists
- This code bad smell makes methods complicated and closed to extensions (e.g., if we need to add new parameters)
- **Metric:** number of parameters in a method

# Real-world Example

**Class** = org.apache.commons.lang3.time.DurationFormatUtils

**Method** = format(...)

**LOC** = 42, CC = 11

**# Parameters** = 9

```
//-----
/**
 * <p>The internal method to do the formatting.</p>
 *
 * @param tokens the tokens
 * @param years the number of years
 * @param months the number of months
 * @param days the number of days
 * @param hours the number of hours
 * @param minutes the number of minutes
 * @param seconds the number of seconds
 * @param milliseconds the number of millis
 * @param padWithZeros whether to pad
 * @return the formatted string
 */
static String format(final Token[] tokens, final long years, final long months, final long days, final long hours, final long minutes, final long seconds, final long
milliseconds, final boolean padWithZeros) {

    final StringBuilder buffer = new StringBuilder();
    boolean lastOutputSeconds = false;
    for (final Token token : tokens) {
        final Object value = token.getValue();
        final int count = token.getCount();
        if (value instanceof StringBuilder) {
            buffer.append(value.toString());
        } else {
            if (value.equals(y)) {
```

# Real-world Example

**Class** = org.apache.commons.lang3.time.DurationFormatUtils

**Method** = format(...)

**LOC** = 42, **CC** = 11

**# Parameters** = 9

A bad smell this has...

Object parameters use you must



# Introduce Parameter Objects

- Apply to methods with too many parameters
- General Mechanism:
  1. Create a new class to represent the group of parameters you are replacing.  
Make the class immutable
  2. Add the parameters for the new data clump
  3. For each parameter in the data clump, remove the parameter from the signature. Modify the callers and method body to use the parameter object for that value
  4. Update and run your tests

# Real-world Example

## Original Code

```
static String format(final Token[] tokens, final long years, final long months,
final long days, final long hours, final long minutes, final long seconds, final
long milliseconds, final boolean padWithZeros) {
    ...
}
```

## Refactored Code

```
static String format(final Token[] tokens, final Date dates, final boolean
padWithZeros) {
    ...
}

public class Date {
    private long years;
    private long months;
    private long days;
    private long hours;
    private long minutes;
    private long seconds;
    private long milliseconds;

    public Date(long years, long months, long days, long hours, long minutes,
long seconds, long milliseconds){
        ...
    }
}
```

**Can we do it better?**

# Real-world Example

## Original Code

```
static String format(final Token[] tokens, final long years, final long months,
final long days, final long hours, final long minutes, final long seconds, final
long milliseconds, final boolean padWithZeros) {
    ...
}
```

## Refactored Code

```
static String format(final Token[] tokens, final Date dates, final boolean
padWithZeros) {
    ...
}

public class Date {
    private long years;
    private long months;
    private Day day;

    public Date(long years, long months, Day day){
        ...
    }
}

public class Day {
    ...
}

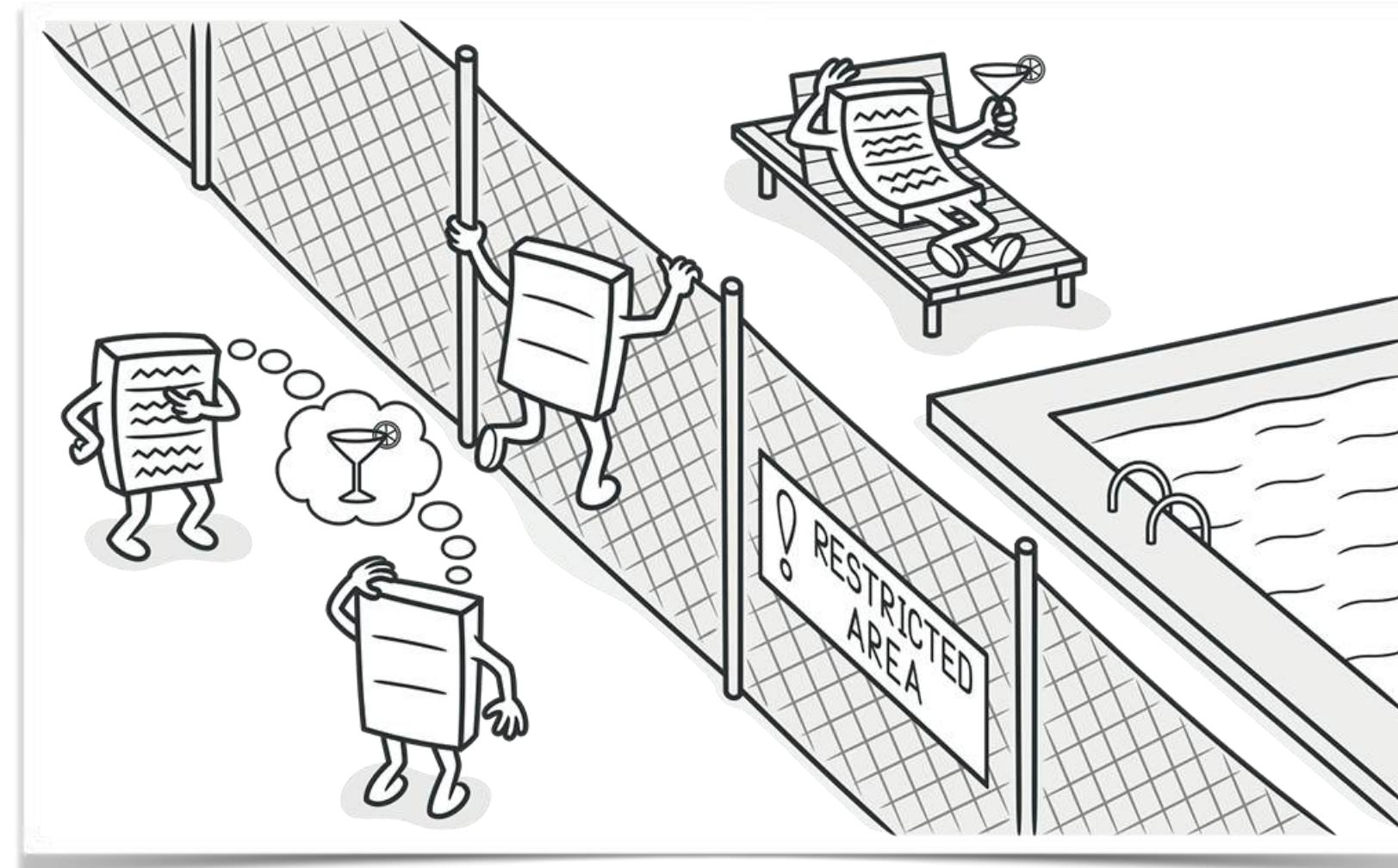
public Day(...){
    ...
}
```



# Couplers

# Feature Envy

- A feature envy is a method that is more interested in other classes rather than the one it actually is in (i.e., a method that is highly coupled with other classes).
- The most common focus of envy is the data.



# Measuring Coupling

**GOAL:** Evaluating the modularity of the system

**CBO:** Coupling Between Objects

**MPC:** Message Passing Coupling

**Pros:** They might be used to evaluate how much a maintenance operation might impact on other classes

**Cons:** Sometimes expensive to compute

# Measuring Coupling

- **Coupling Between Objects** (CBO) = # dependencies with other classes.  
Dependencies are established through parameters, attributes modified/read, method calls, object instantiation
- **Message Passing Coupling** (MPC) = # of messages (method calls) exchanged by two classes

# Example

```
public class Person {  
  
    private String name;  
    private String surname;  
    private String address;  
    private Phone mobilePhone;  
  
    public Person(String name, String surname,  
                 String address){  
        this.surname = surname;  
        this.name = name;  
        this.address = address;  
    }  
  
    public void setPhoneNumber(Phone phone){  
        this.mobilePhone = phone;  
    }  
  
    public String getMobilePhoneNumber() {  
        return "(" +  
               mobilePhone.getAreaCode() + ")" +  
               mobilePhone.getPrefix() + "-" +  
               mobilePhone.getNumber();  
    }  
}
```

```
public class Phone {  
    private final String unformattedNumber;  
  
    public Phone(String unformattedNumber) {  
        this.unformattedNumber = unformattedNumber;  
    }  
  
    public String getAreaCode() {  
        return unformattedNumber.substring(0,3);  
    }  
  
    public String getPrefix() {  
        return unformattedNumber.substring(3,6);  
    }  
  
    public String getNumber() {  
        return unformattedNumber.substring(6,10);  
    }  
}
```

*This method invokes three  
methods of the class Phone and it  
has no interaction (low cohesion)  
with the other methods of the  
class Person*

# Move Method Refactoring

- Apply this refactoring to move methods from one class to another one to reduce the overall code coupling
- General Mechanism:
  1. Examine all features used by the source method and that are defined on the source class
  2. Check the sub- and super-classes of the source class for other declarations of the method.
  3. Declare the method in the target class
  4. Copy the code from the source method to the target and adapt it
  5. Turn the source method into a delegating method
  6. Update and run your tests

# Move Method Refactoring

## Original Code

```
public class Person {  
  
    private String name;  
    private String surname;  
    private String address;  
    private Phone mobilePhone;  
  
    public Person(String name, String surname,  
                 String address){  
        this.surname = surname;  
        this.name = name;  
        this.address = address;  
    }  
  
    public void setPhoneNumber(Phone phone){  
        this.mobilePhone = phone;  
    }  
  
    public String getMobilePhoneNumber() {  
        return "(" +  
               mobilePhone.getAreaCode() + ")" +  
               mobilePhone.getPrefix() + "-" +  
               mobilePhone.getNumber();  
    }  
}
```

```
public class Phone {  
    private final String unformattedNumber;  
  
    public Phone(String unformattedNumber) {  
        this.unformattedNumber = unformattedNumber;  
    }  
  
    public String getAreaCode() {  
        return unformattedNumber.substring(0,3);  
    }  
  
    public String getPrefix() {  
        return unformattedNumber.substring(3,6);  
    }  
  
    public String getNumber() {  
        return unformattedNumber.substring(6,10);  
    }  
}
```

# Move Method Refactoring

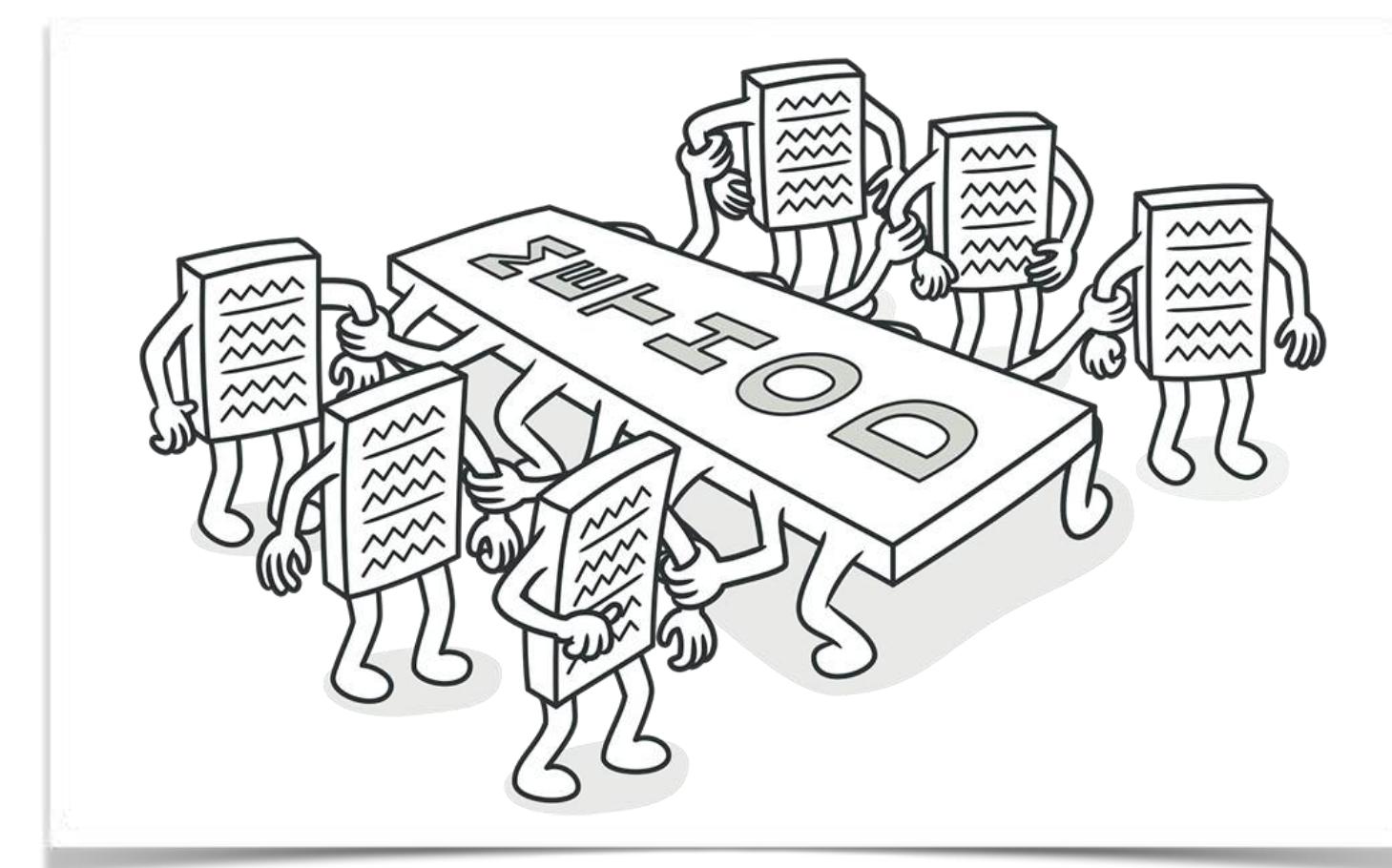
## Refactored code

```
public class Person {  
  
    private String name;  
    private String surname;  
    private String address;  
    private Phone mobilePhone;  
  
    public Person(String name, String surname,  
                 String address){  
        this.surname = surname;  
        this.name = name;  
        this.address = address;  
    }  
  
    public void setPhoneNumber(Phone phone){  
        this.mobilePhone = phone;  
    }  
  
    public String getMobilePhoneNumber() {  
        return mobilePhone.toFormattedString();  
    }  
}
```

```
public class Phone {  
    private final String unformattedNumber;  
  
    public Phone(String unformattedNumber) {  
        this.unformattedNumber = unformattedNumber;  
    }  
  
    public String getAreaCode() {  
        return unformattedNumber.substring(0,3);  
    }  
  
    public String getPrefix() {  
        return unformattedNumber.substring(3,6);  
    }  
  
    public String getNumber() {  
        return unformattedNumber.substring(6,10);  
    }  
  
    public String toFormattedString() {  
        return "(" + getAreaCode() + ")" +  
               getPrefix() + "-" + getNumber();  
    }  
}
```

# Coupled Classes

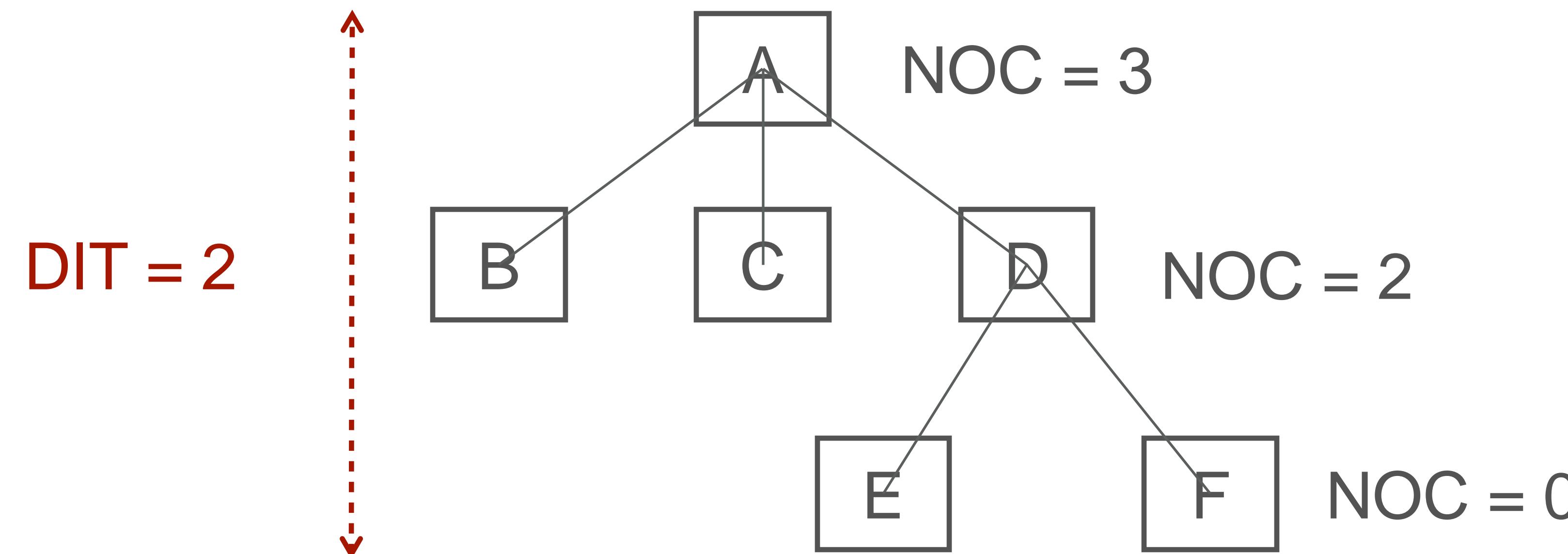
- Coupled classes (also called inappropriate intimacy) occurs when one class A uses the internal fields and methods of another class B and vice-versa
- Coupled classes can be viewed as a bi-directional "feature envy": each class is "envious" of some functionalities or data defined in the other class.
- This bad code smell is frequent when there is a high inheritance, where the child class knows more about the parent class than is permitted



# Software Metrics

Metrics:

- **Coupling between objects** (CBO)
- **Depth of Inheritance Tree** (DIT) is the maximum length from the node to the root of the inheritance tree
- **Number Of Children** (NOC) is the number of direct subclasses for a given class C



# Refactoring Operations

- Use **Move Method** refactoring to separate the pieces to reduce the coupling
- If the classes do have common interests, use **Extract Class** to put the commonality in a safe place
- **Replace Delegation with Inheritance**

# Replace Delegation with Inheritance

- Use this refactoring when you are using delegation and are often writing many simple delegations for the entire interface
- General Mechanism
  1. Make the delegating object a subclass of the delegate
  2. Set the delegate field to be the object itself
  3. Replace all other delegations with calls to the object itself
  4. Remove the delegate field
  5. Update and run your tests

# Example

*Employee delegates parts of its behavior to Person*

```
public class Person {  
  
    private String name;  
    private String surname;  
    private String address;  
    private Phone mobilePhone;  
  
    public Person(String name, String surname,  
                 String address){  
        this.surname = surname;  
        this.name = name;  
        this.address = address;  
    }  
  
    public void setPhoneNumber(Phone phone){  
        this.mobilePhone = phone;  
    }  
  
    public String getMobilePhoneNumber() {  
        return mobilePhone.toFormattedString();  
    }  
  
    // getter and setter  
}
```

```
public class Employee {  
    private Person person;  
    private double monthlySalary;  
  
    public Employee(String name, String surname,  
                   String address){  
        person = new Person(name, surname, address);  
        this.monthlySalary = 1500.00;  
    }  
  
    public String getName(){  
        return this.person.getName();  
    }  
  
    public void setName(String name){  
        this.person.setName(name);  
    }  
  
    public String getSurname(){  
        return this.person.getSurname();  
    }  
  
    // getter and setter  
}
```

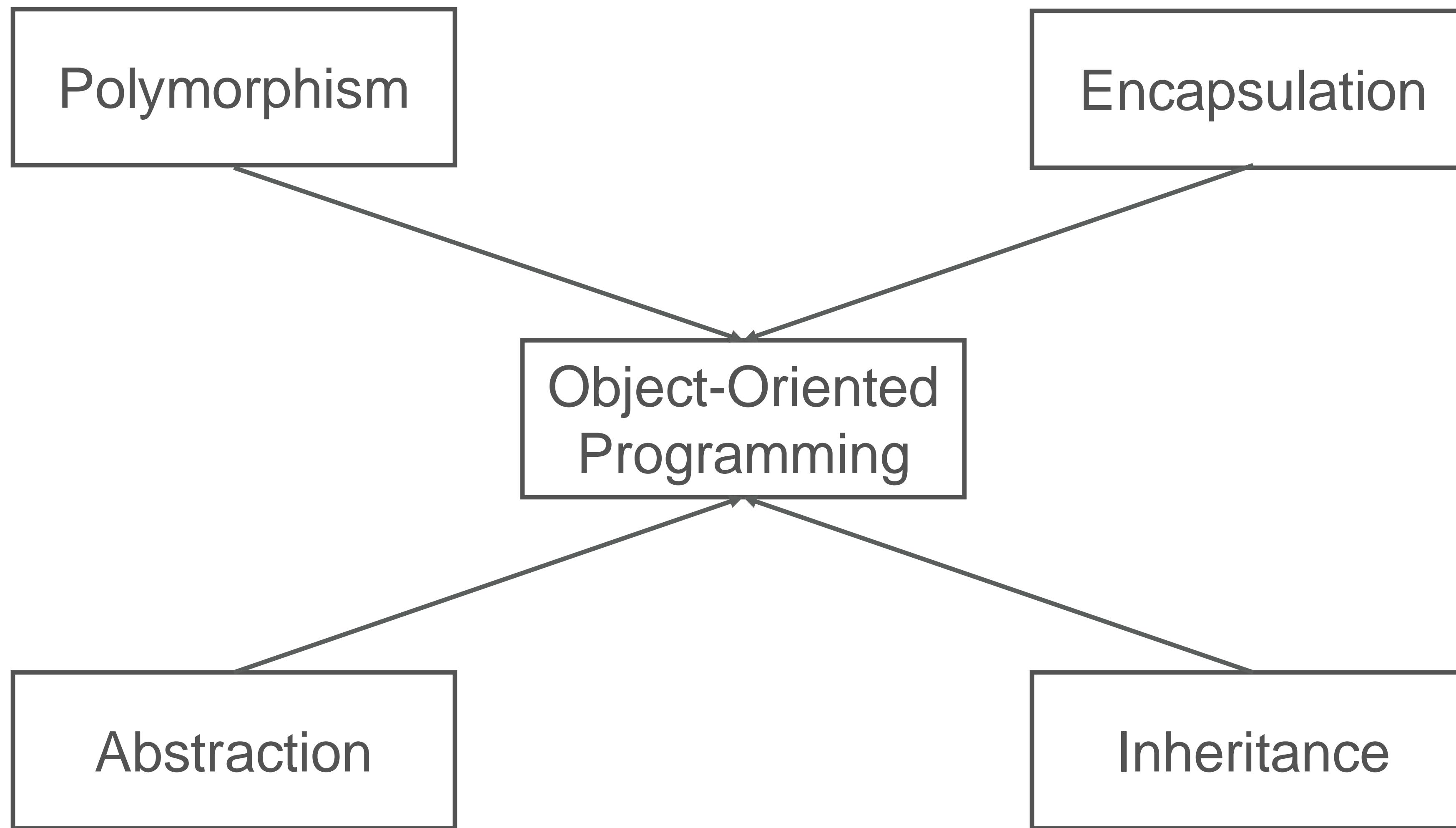
# Example

## Refactored code

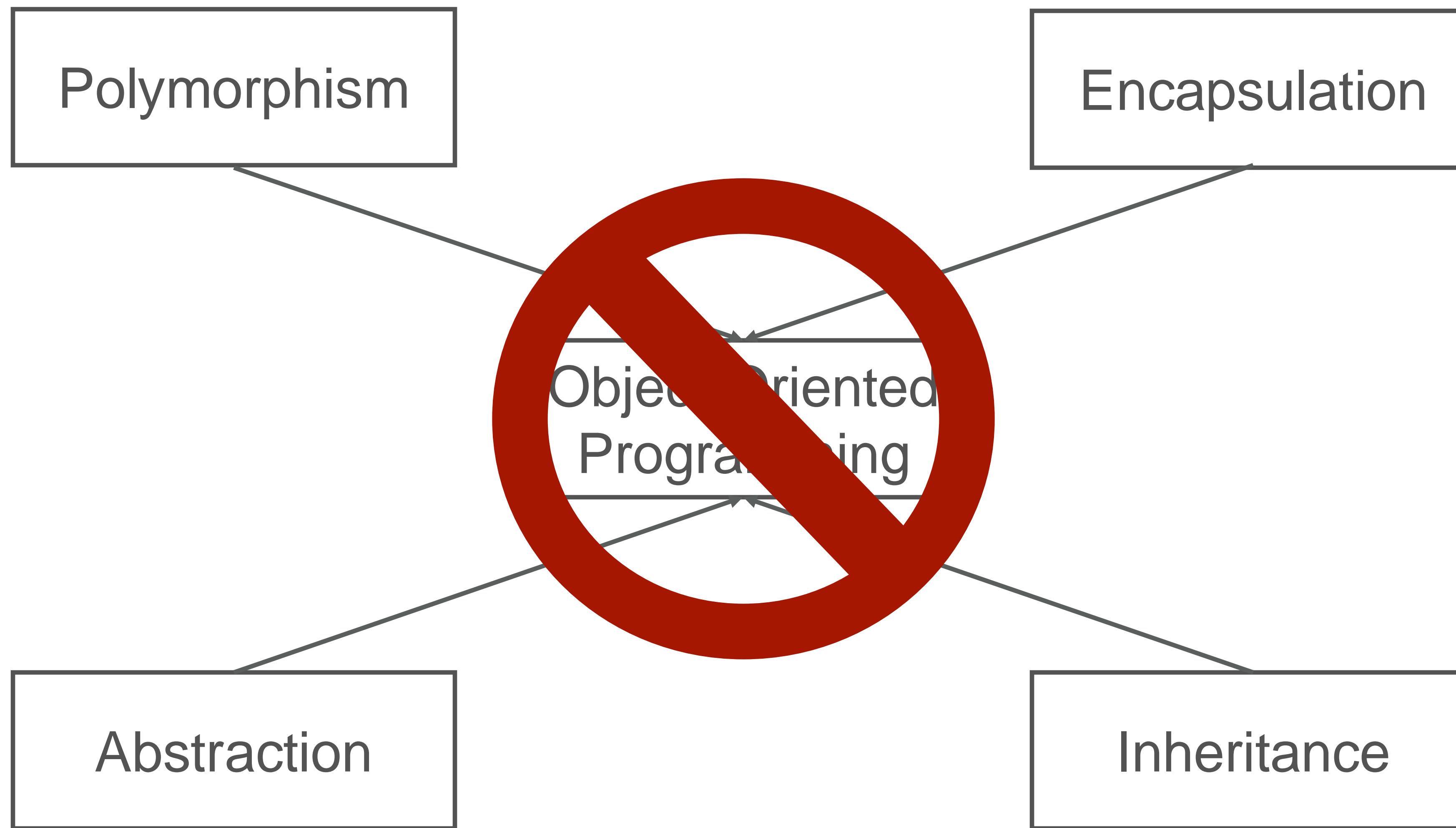
```
public class Person {  
  
    protected String name;  
    protected String surname;  
    protected String address;  
    protected Phone mobilePhone;  
  
    public Person(String name, String surname,  
                 String address){  
        this.surname = surname;  
        this.name = name;  
        this.address = address;  
    }  
  
    public void setPhoneNumber(Phone phone){  
        this.mobilePhone = phone;  
    }  
  
    public String getMobilePhoneNumber() {  
        return mobilePhone.toFormattedString();  
    }  
  
    //getter and setter  
}
```

```
public class Employee extends Person{  
  
    private double monthlySalary;  
  
    public Employee(String name, String surname,  
                   String address){  
        super(name, surname, address);  
        this.monthlySalary = 1500.00;  
    }  
  
    public double getMonthlySalary() {  
        return monthlySalary;  
    }  
  
    public void setMonthlySalary(double monthlySalary) {  
        this.monthlySalary = monthlySalary;  
    }  
}
```

# Object Oriented abusers



# Object Oriented abusers



# Switch Statements

- The presence of **switch statements** (or nested if-then-else statements) in the code the classic example of poor object-oriented design
- A switch statement is a violation of the **Open-Closed Principle** because each time the developer needs to modify the cases to accommodate a new condition (or remove a condition)

# Example

```
class Pokemon {  
    private PokemonType type;  
    private int level;  
    private int attackPower;  
  
    public Pokemon(PokemonType type, int level){  
        this.type = type;  
        this.level = level;  
        this.attackPower = 10;  
    }  
  
    public String getAttack() {  
        switch (type) {  
            case PICKACHU:  
                return getElectricAttack();  
            case BULBASAUR:  
                return getGrassAttack();  
            case CHARMANDER:  
                return (level >= 25) ? getFireAttack(1) : getFireAttack(2);  
        }  
        throw new RuntimeException("Should be unreachable");  
    }  
  
    private String getFireAttack(int type) {  
        return (type == 1) ? "Charmander uses Fire Blast" : "Charmander uses Ember";  
    }  
  
    private String getGrassAttack() {  
        return "Bulbasaur uses Cut";  
    }  
  
    private String getElectricAttack() {  
        return "Pikachu uses ThunderShock";  
    }  
}  
  
public enum PokemonType {  
    PICKACHU, BULBASAUR, CHARMANDER  
}
```

# Replace Conditional with Polymorphism

- Apply this refactoring when you have conditional statements that choose different behaviors depending on the type of an object.
- General Mechanism:
  1. If the conditional statement is one part of a larger method, take apart the conditional statement and use **Extract Method**
  2. If necessary use **Move Method** to place the conditional at the top of the inheritance structure
  3. Create a subclass method that overrides the conditional statement method
  4. Remove the (copied) conditional code
  5. Update and run the tests

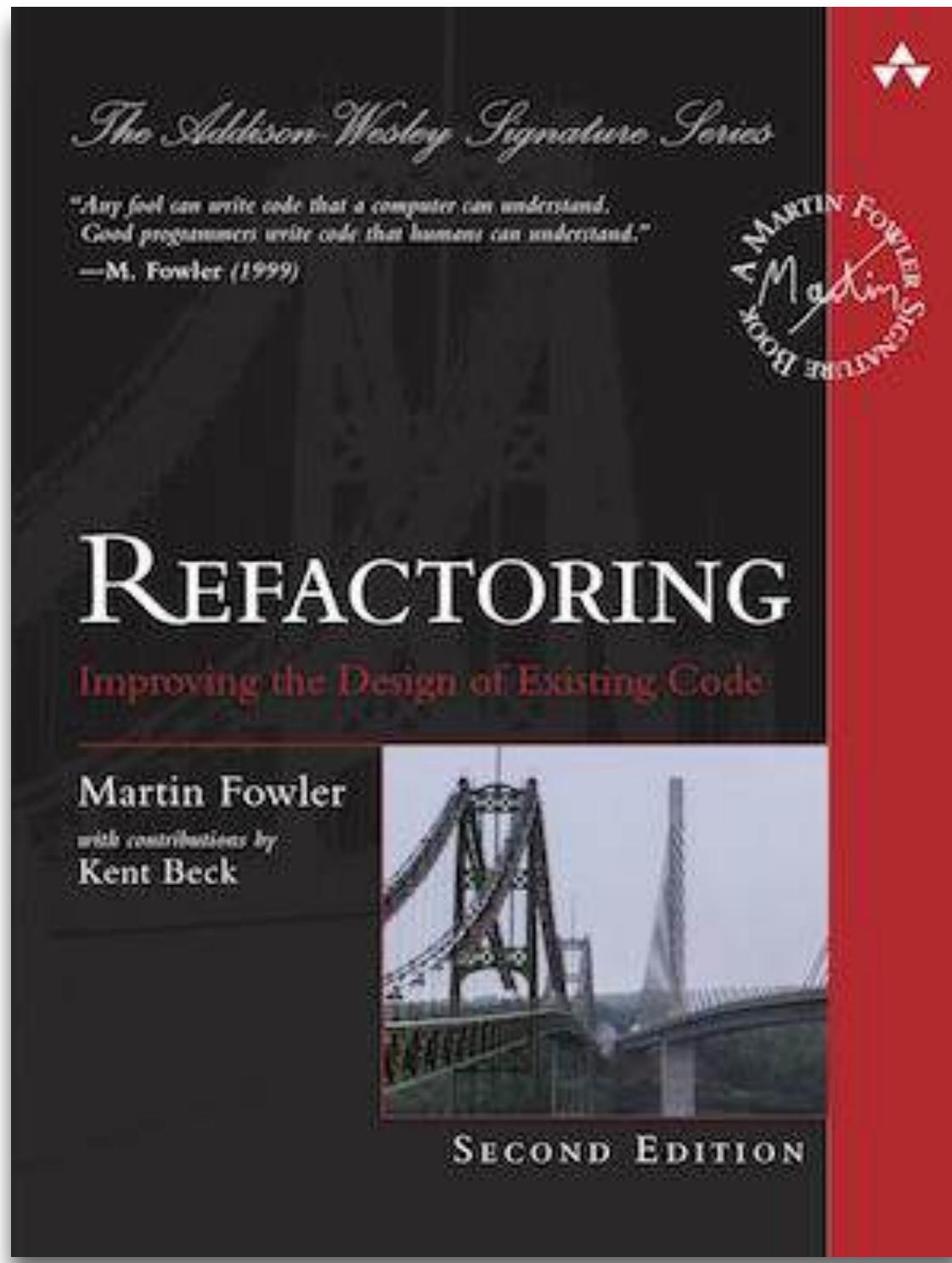
# Replace Conditional with Polymorphism

## Refactored Code

```
public abstract class AbstractPokemon {  
  
    protected PokemonType type;  
    protected int level;  
    protected int attackPower;  
    protected String attackName;  
  
    public AbstractPokemon(){  
        this.attackPower = 10;  
    }  
  
    abstract public String getAttack();  
}
```

```
public class Pikachu extends AbstractPokemon {  
  
    public Pikachu(int level) {  
        super();  
        this.type = PokemonType.PICKACHU;  
        this.level = level;  
        this.attackName = "ThunderShock";  
    }  
  
    @Override  
    public String getAttack() {  
        return "Pikachu uses "+this.attackName;  
    }  
}  
  
public class Charmander extends AbstractPokemon {  
  
    public Charmander(int level) {  
        super();  
        this.type = PokemonType.CHARMANDER;  
        this.level = level;  
  
        if (level >= 25)  
            this.attackName = "Fire Blast";  
        else  
            this.attackName = "Ember";  
    }  
  
    @Override  
    public String getAttack() {  
        return "Charmander uses "+this.attackName;  
    }  
}
```

# Warning



By Martin Fowler  
with Kent Beck

“A code smell is a surface indication  
that usually corresponds to a deeper  
problem in the system...”

**“... smells don't always  
indicate a problem...”**

Martin Fowler

# Some Remarks

- In this lecture, we have seen only a subset of the existing bad code smells, metrics and refactoring operations
- For the exam, you will get questions only for code smells, metrics and refactoring operations seen in this lecture
- For the assignment, you are free to use other metrics, to detect other code smells and to apply other refactoring operations (with proper motivation)

# To realize....

- Technical debt is a useful metaphor to explain the issue to non-software engineers
- Not all technical debt is equal:
  - a file that you often have to change that has high technical debt
  - a file that you change every 2 years that has high technical debt
- It is always a trade-off (clean code versus time, clean code versus performance, ...)

# Defects

- **Functional defects** are the ones that lead to bugs
- **Maintainability defects** are the ones that potentially lead to future difficulties in understanding, maintainability, ...

# Tools for Computing Software Metrics

There are many tools available to compute both **method-level** and **class-level** metrics (such as CBO=Coupling between Objects, LOC=lines of code, CC = Cyclomatic Complexity).

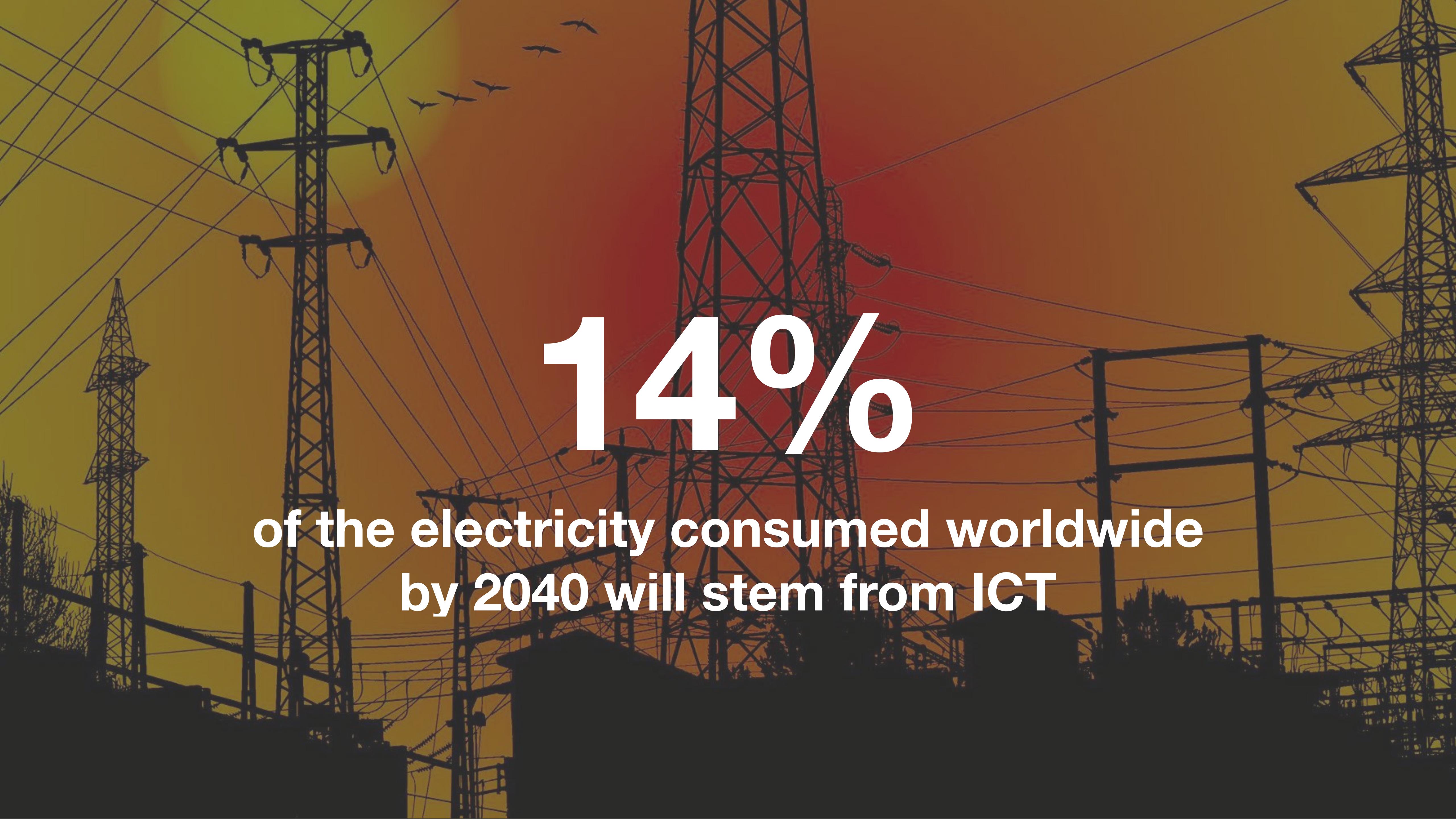
Some well-known tools are:

- Command line tool: CKJM (<https://www.spinellis.gr/sw/ckjm/>)
- Command line tool: the extended CKJM ([http://gromit.iiar.pwr.wroc.pl/p\\_inf/ckjm](http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm))
- Command line too: <https://github.com/mauricioaniche/ck>
- For IntelliJ: CodeMR (<https://marketplace.eclipse.org/content/codemr-software-quality-tool#group-metrics-tab>)
- For IntelliJ: MetricsTree (<https://plugins.jetbrains.com/plugin/13959-metricstree>)

For the second assignment, you are free to choose any tool. The minimum set of software metrics to consider are the **Chidamber & Kemerer metrics**.



# 1. Sustainable SE – What and Why?



**14%**

of the electricity consumed worldwide  
by 2040 will stem from ICT

bbc.com

Home News US Election Sport Business Technology Culture Arts Travel Earth Video Live

# AI drives 48% increase in Google emissions

3 July 2021

Share Save

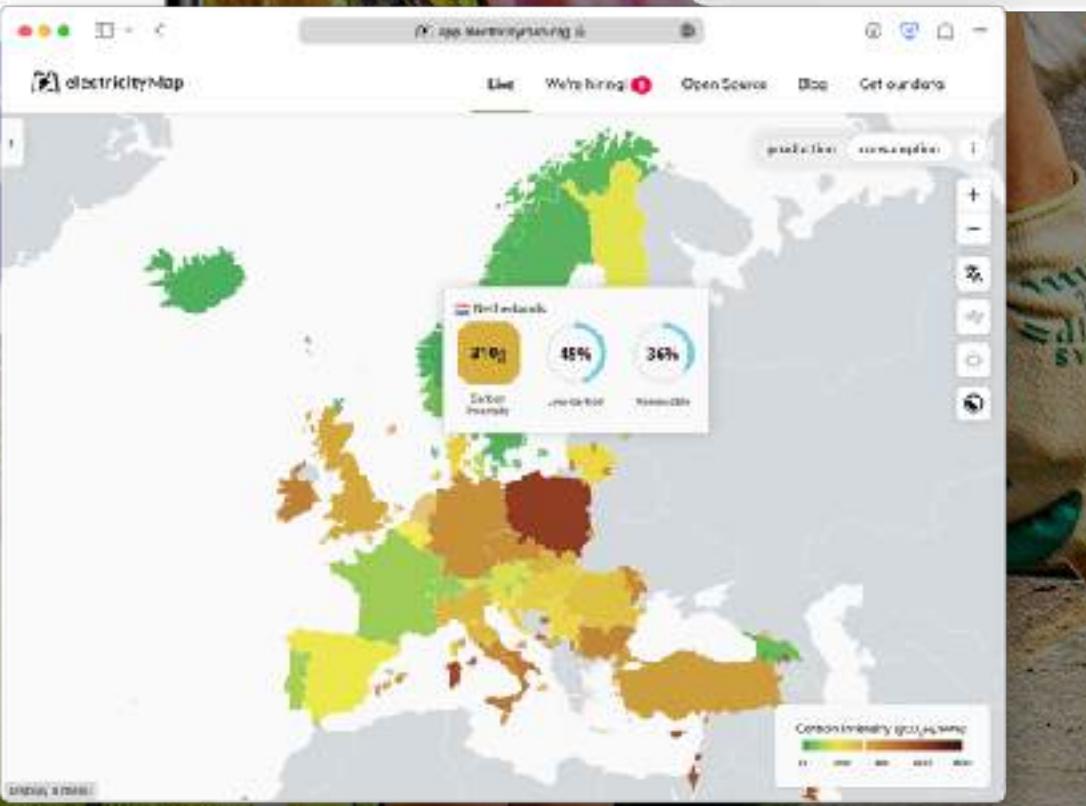
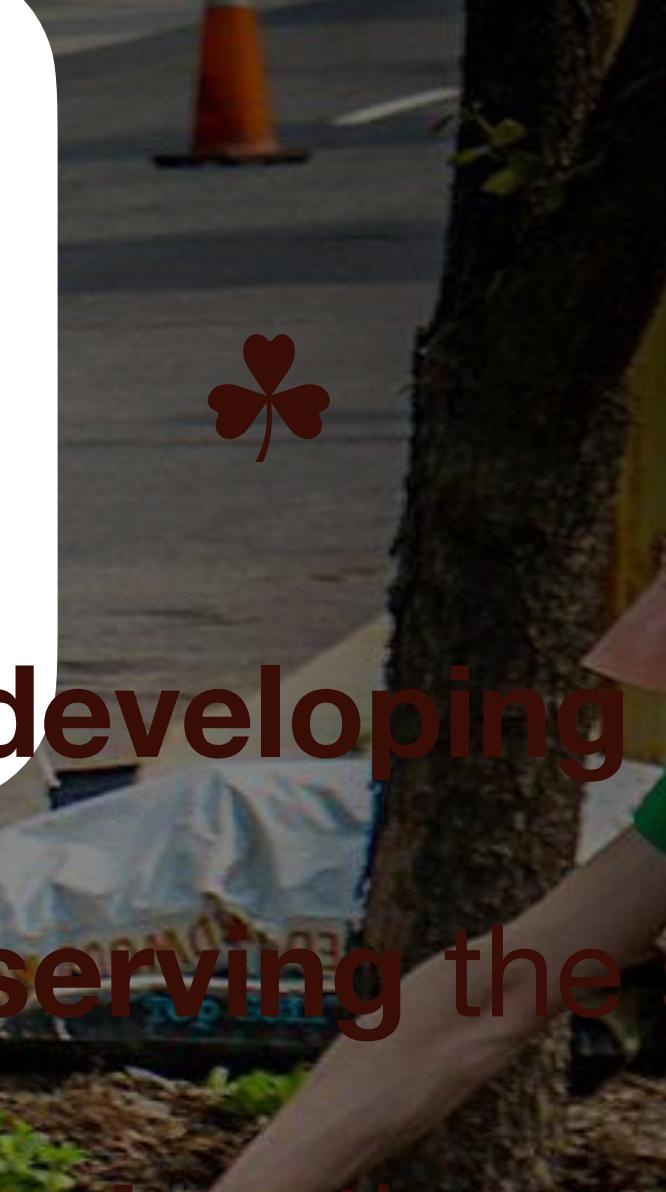
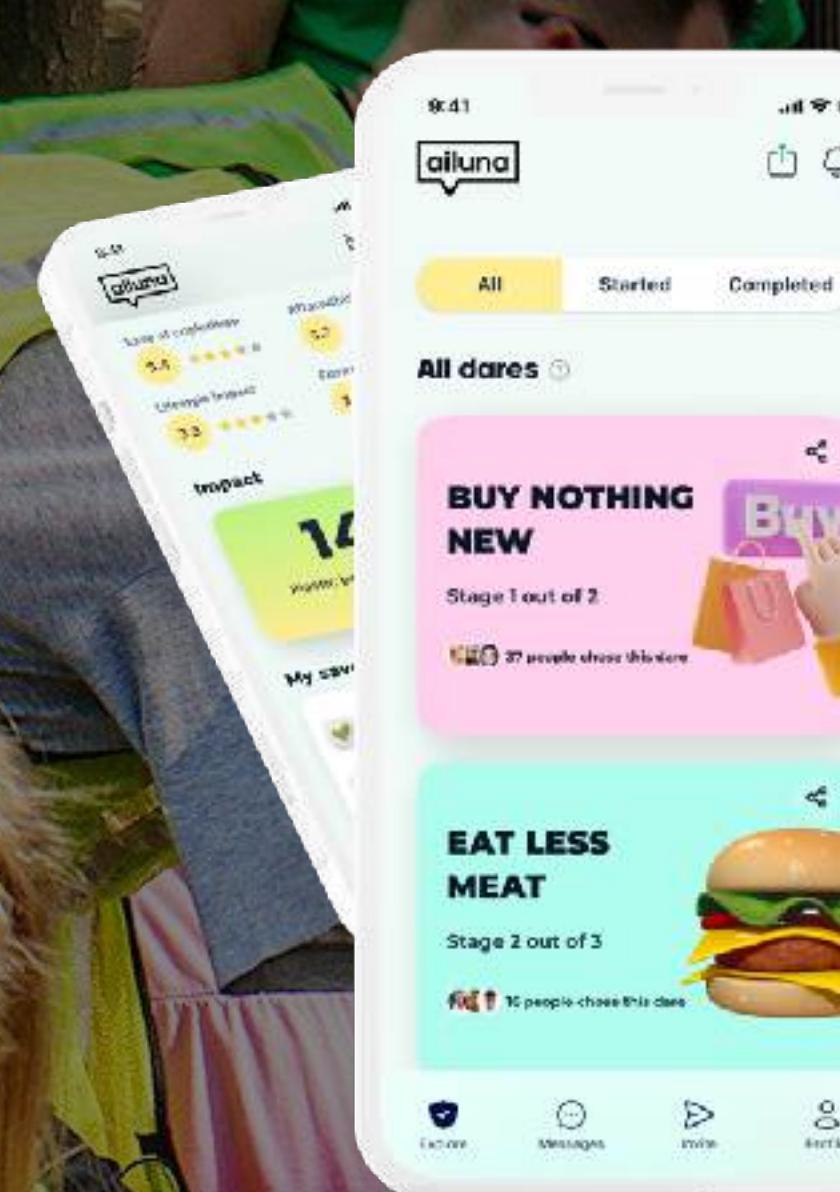
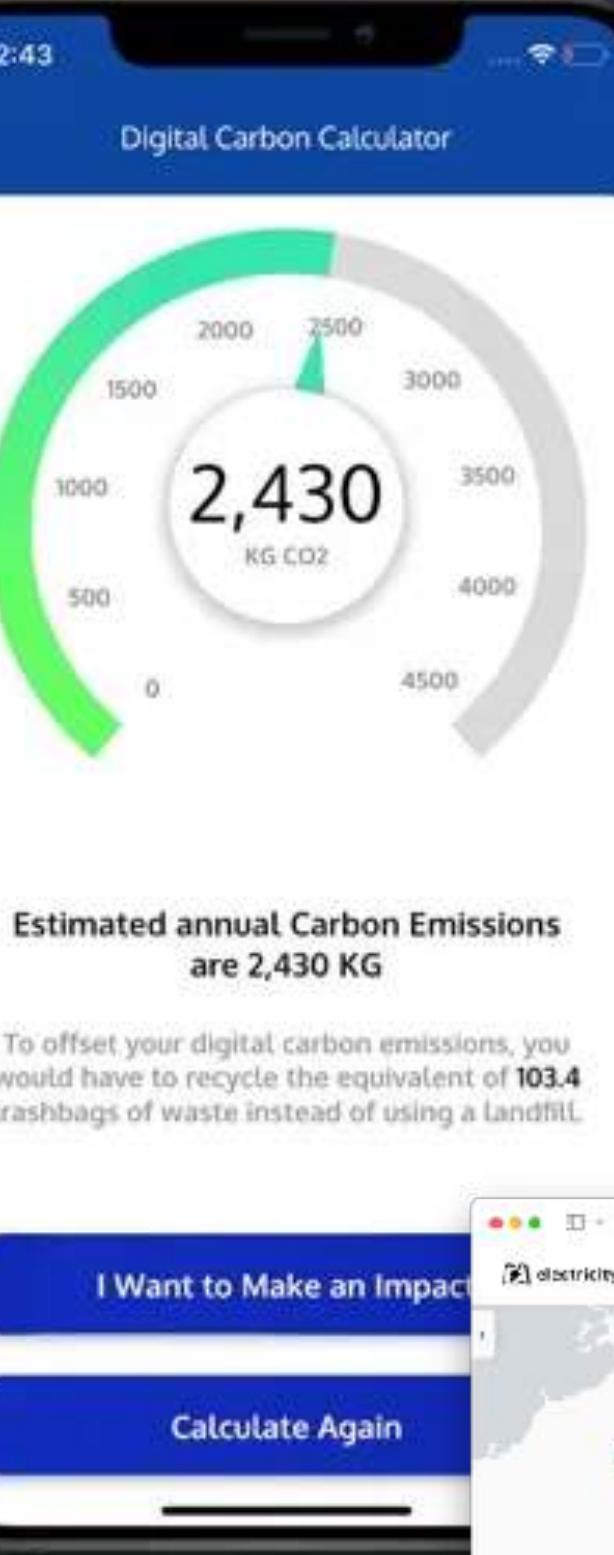
Imran Rahman-Jones  
Technology reporter

Display a menu

- Mobile phones are **low-power** devices with extraordinary computing capacity (700Gflops?)
- Power: **5W**
- Power for **8B people/phones**:
  - 40GW
  - Equivalent to **10 Fukushima nuclear power plants** !

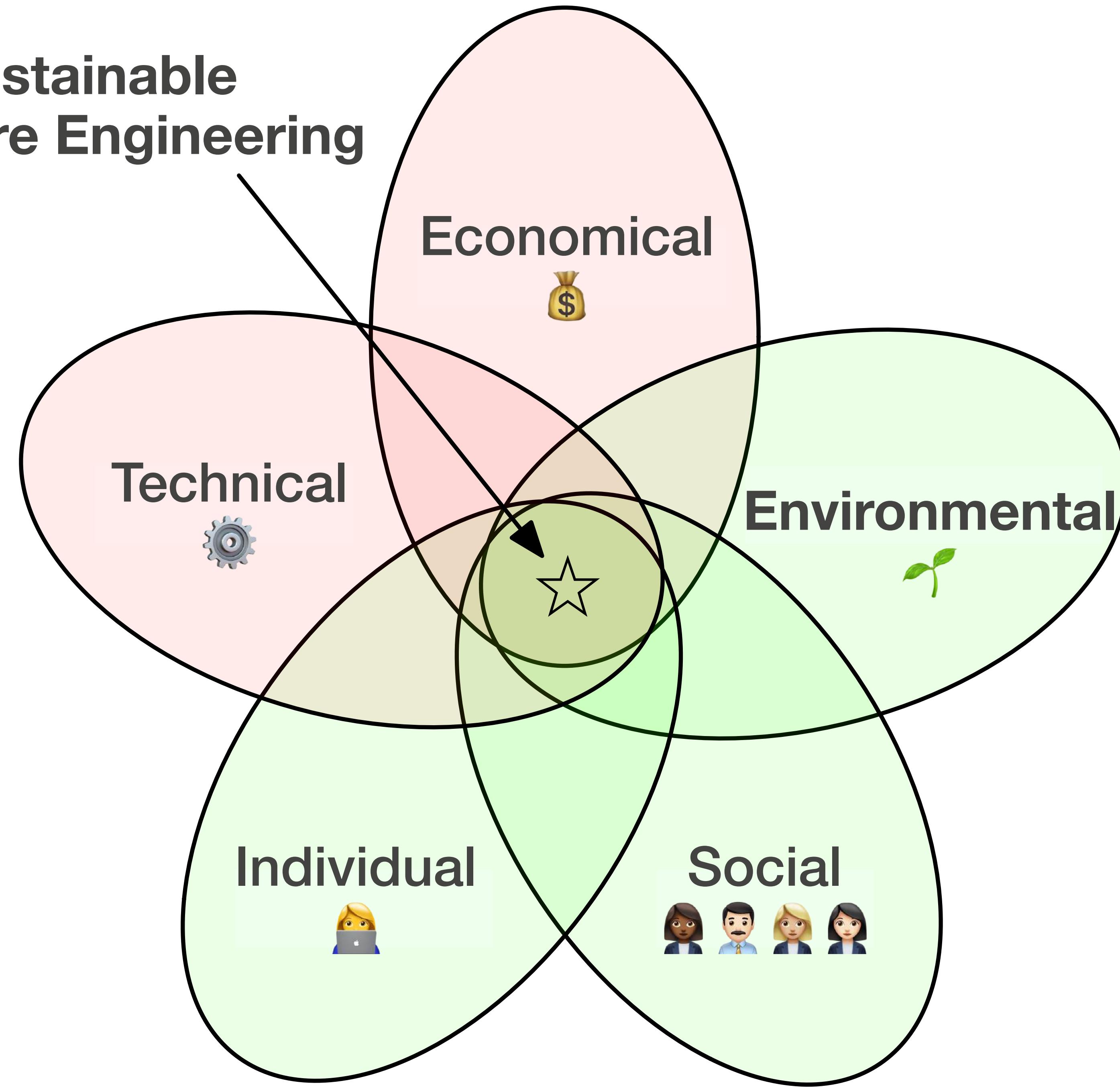


# Software for Sustainability

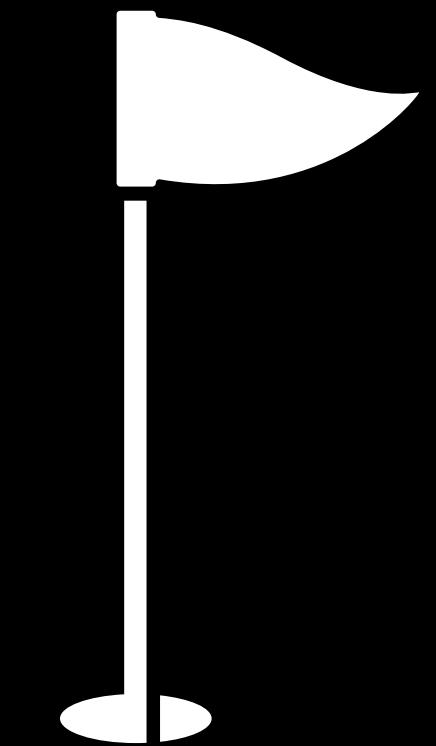


# Sustainability of Software

# Sustainable Software Engineering



# Learning Activity



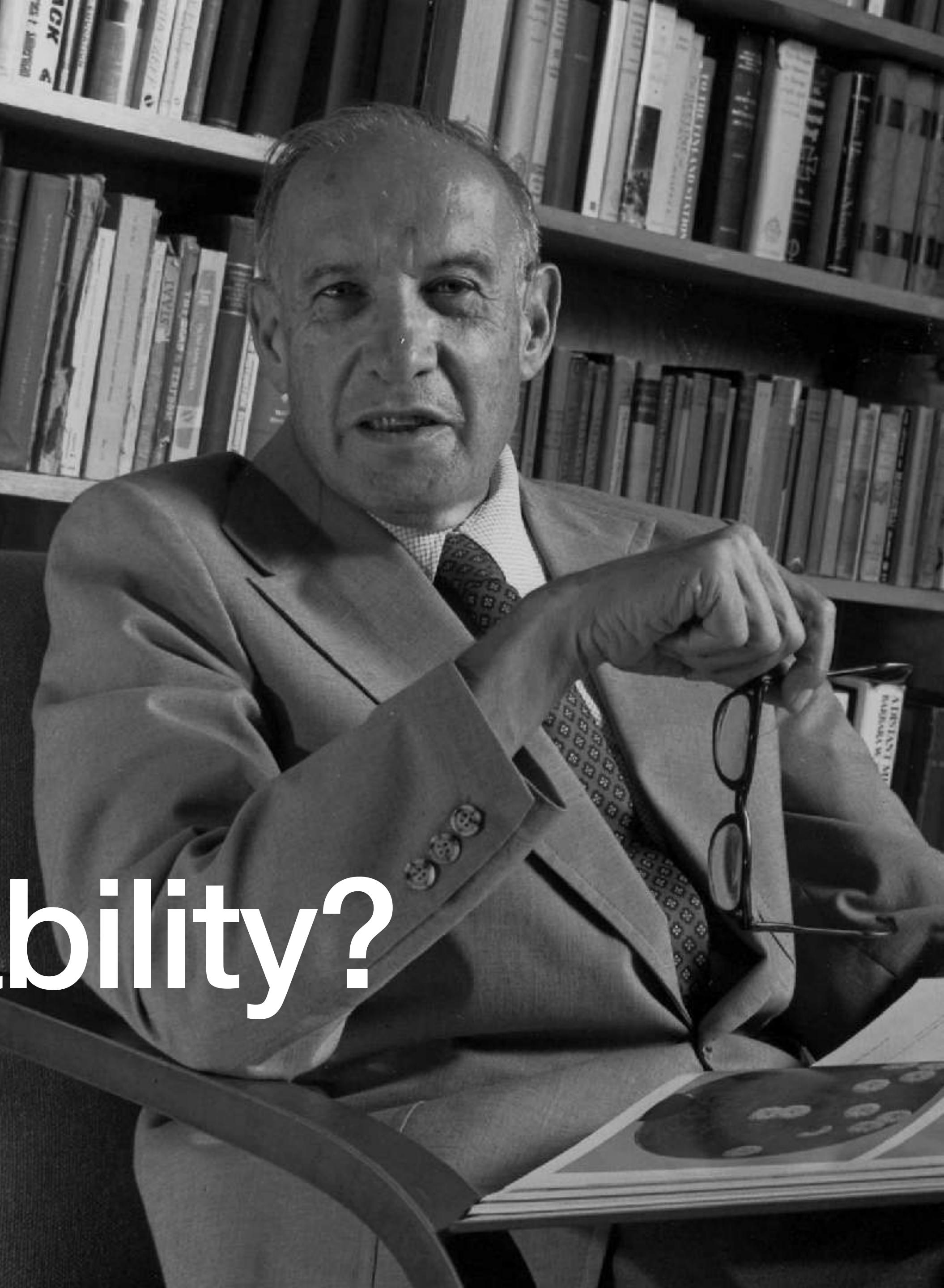
- 5 min
- Pair with a colleague
- Choose one of the sustainability dimensions and think about one of your previous software projects:
  - Reflect on a sustainability issue that you had to address. What was it and how did you work around it?
  - In the end, some groups will share their answers with the class.

# 2. Measuring

*“You can't improve what you  
don't measure.”*

– Peter Drucker

# How to measure software sustainability?



# Metrics

- Carbon emissions
  - From executing the software
  - From producing the required hardware
  - From cooling down datacenters
- Water consumption
- Electricity
  - energy consumption?
  - Power consumption?
- Memory usage?
- CPU usage?
- Network traffic?
- ...

carbon  
energy  
watts  
mAh  
kWh  
efficiency  
battery  
joules  
credits  
consumption  
power  
credits

# Bitcoin example

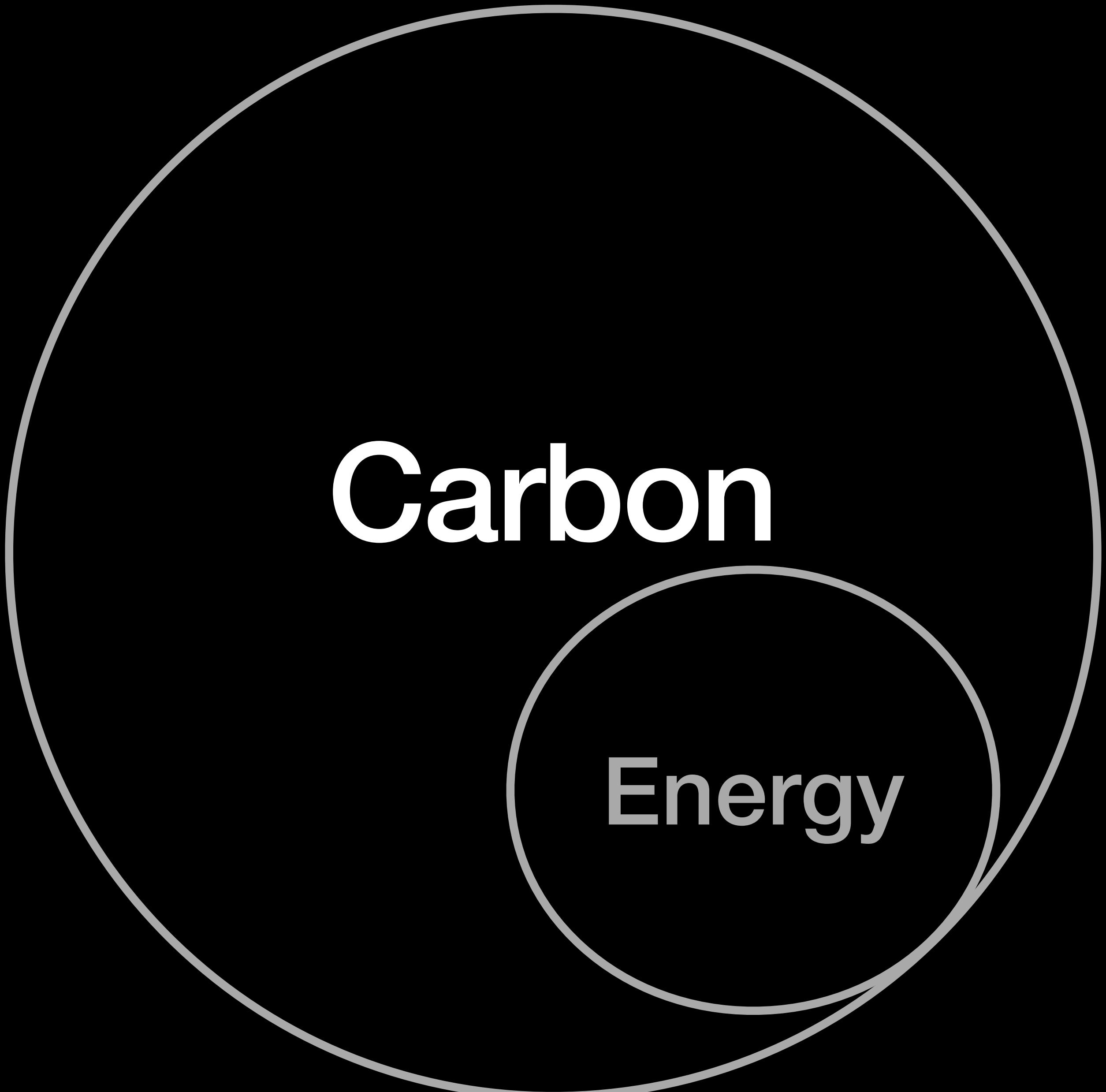
- 1 **bitcoin** transaction is equivalent to more than 1.5 million **VISA** transactions.



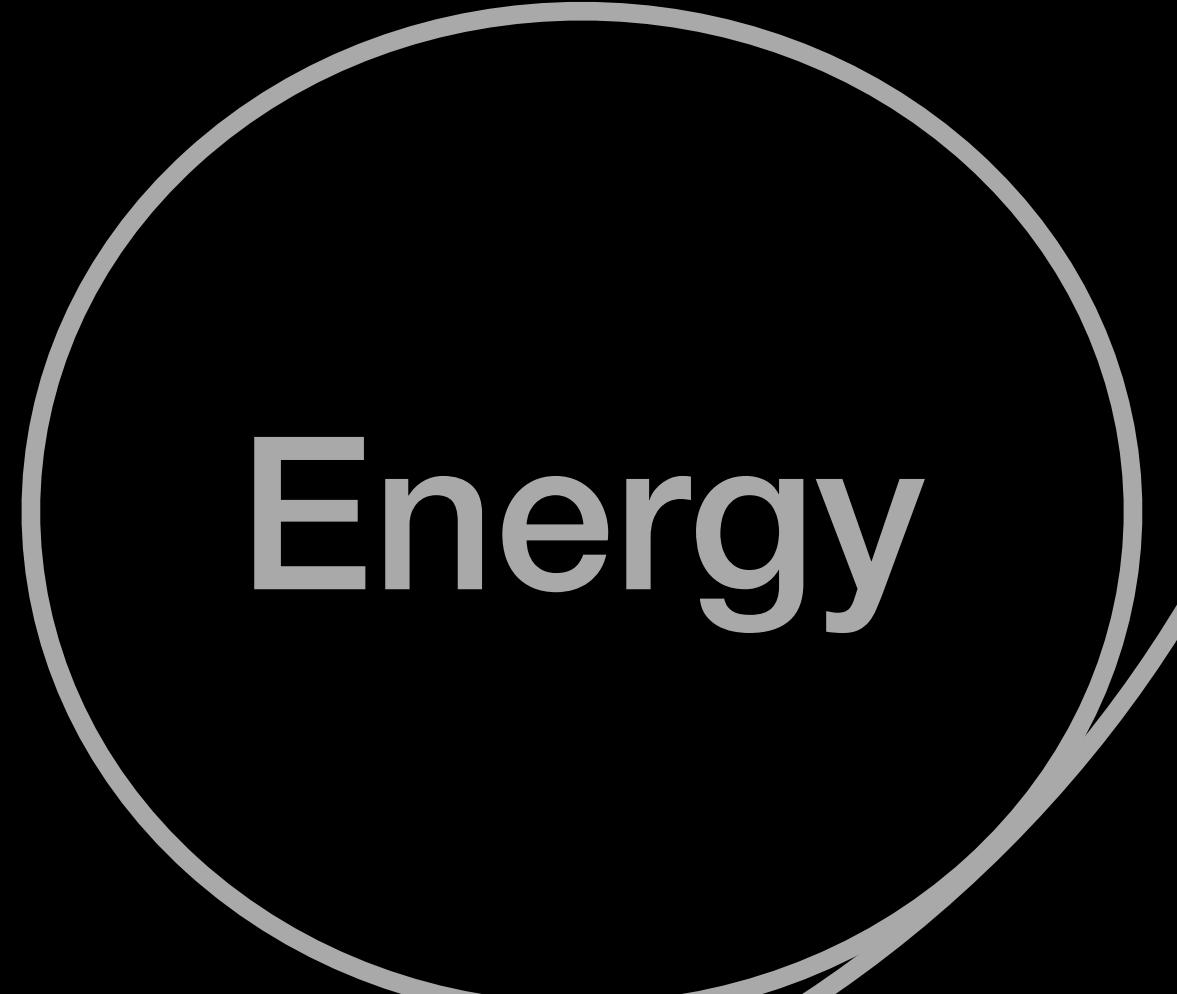
- Day-to-day metrics are easy to grasp
  - If we say 8 **gigajoules**, it's a bit more difficult to understand.
- Different **stakeholders** require **different metrics**.
- These numbers keep changing (check them here: <https://www.statista.com/statistics/881541/bitcoin-energy-consumption-transaction-comparison-visa/>)

# Other units

- Whenever talking to a general audience use relative units:
  - Compare to the other well-known things:
    - Yearly **household** energy consumption
    - Yearly **country** electrical energy consumption.  
(e.g., <https://ccaf.io/cbeci/index/comparisons>)
    - **Driving kms** with a standard car
    - Percentage of a normal **battery** charge cycle.
  - Compare to other software artefacts/usecases:
    - E.g, percentage of **Version A over Version B**.



Carbon



Energy

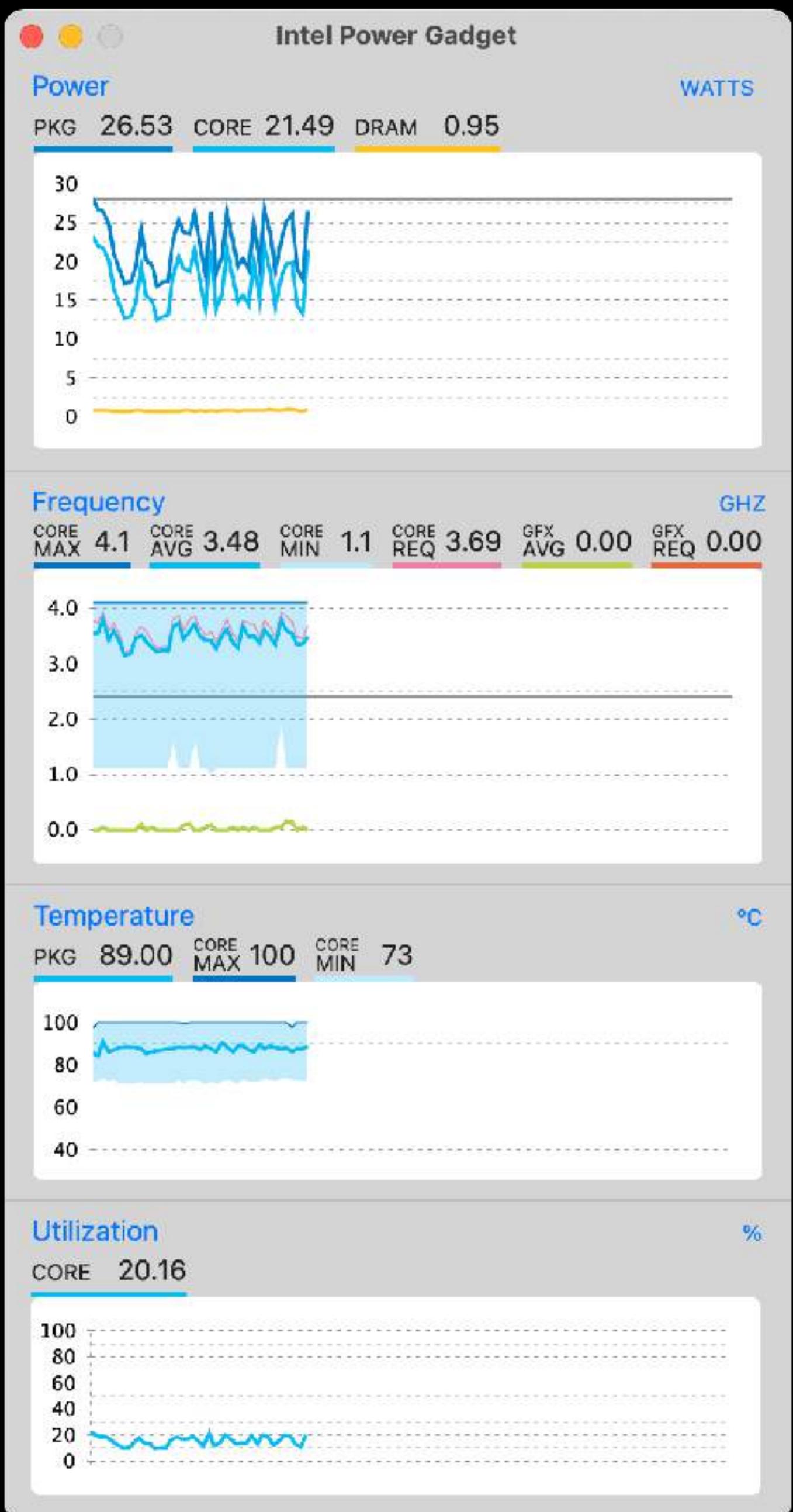
# When should we use Carbon vs Energy?

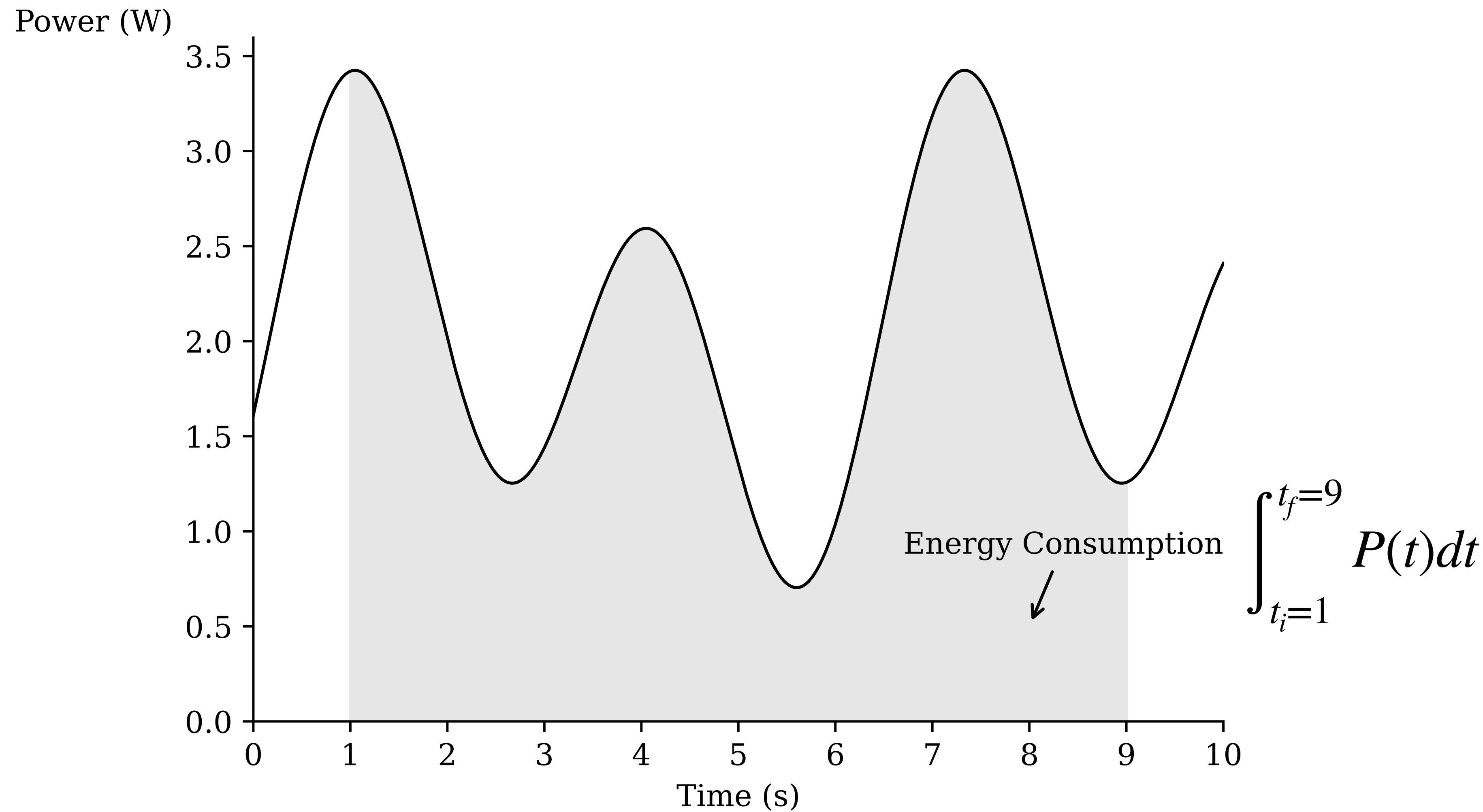
- Energy/Power is more useful at the **software usecase** level.
- Carbon is more useful at the **infrastructure** level (e.g., datacenter) or at the **project** level (e.g., the impact of developing a full software project).
- Choose your metrics wisely ;)

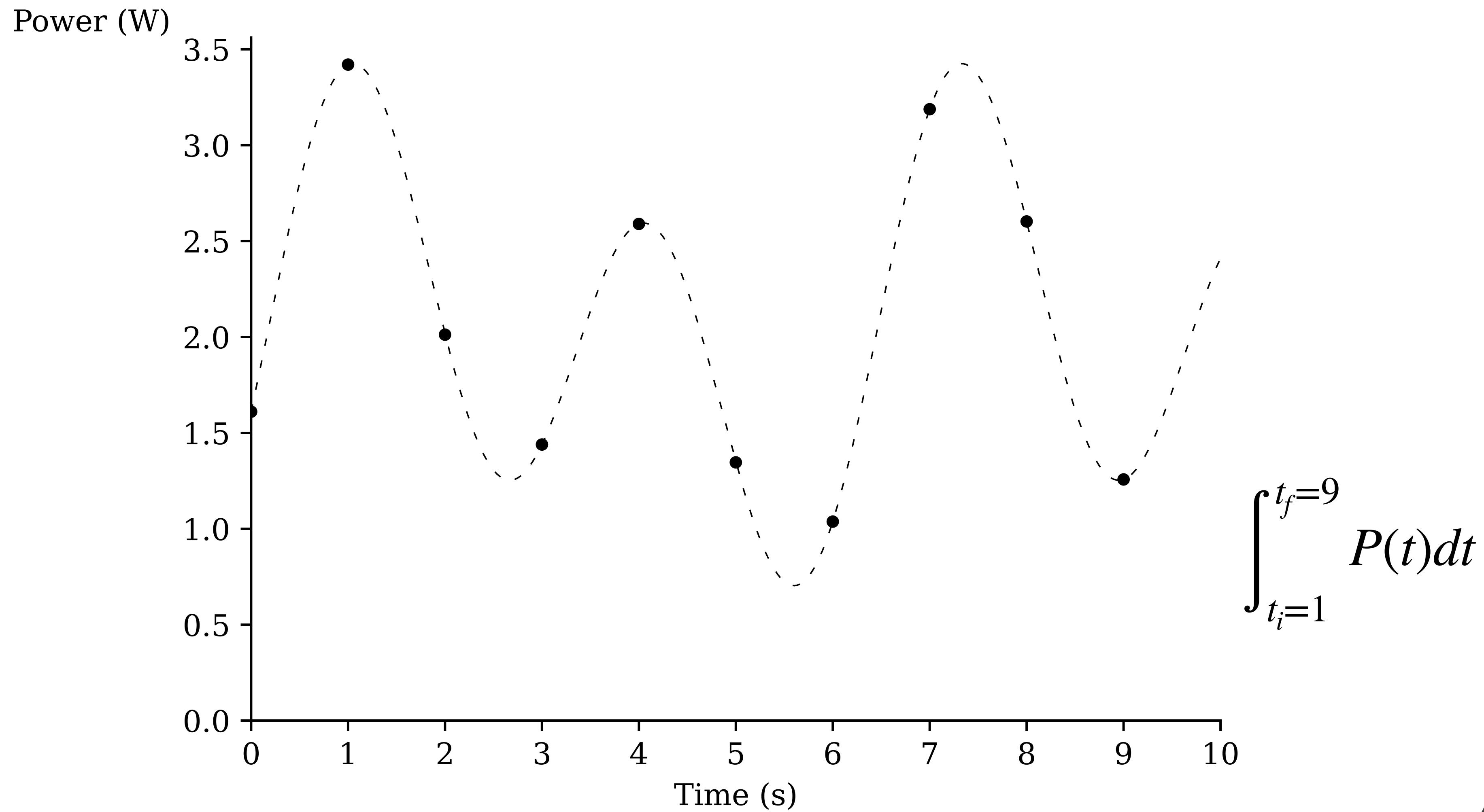
- **Measuring Energy Consumption (one possible way)**
- **To collect:** Logging > Log to File



- It will store a **CSV file** with all the collected power data. (File location is specified in the settings)
- Based on Intel **RAPL**. Works with Intel-based Windows and Macs.
- Alternative-twin for M1-based Macs: **Mx Power Gadget**.  
<https://www.seense.com/menuubarstats/mxpg/>

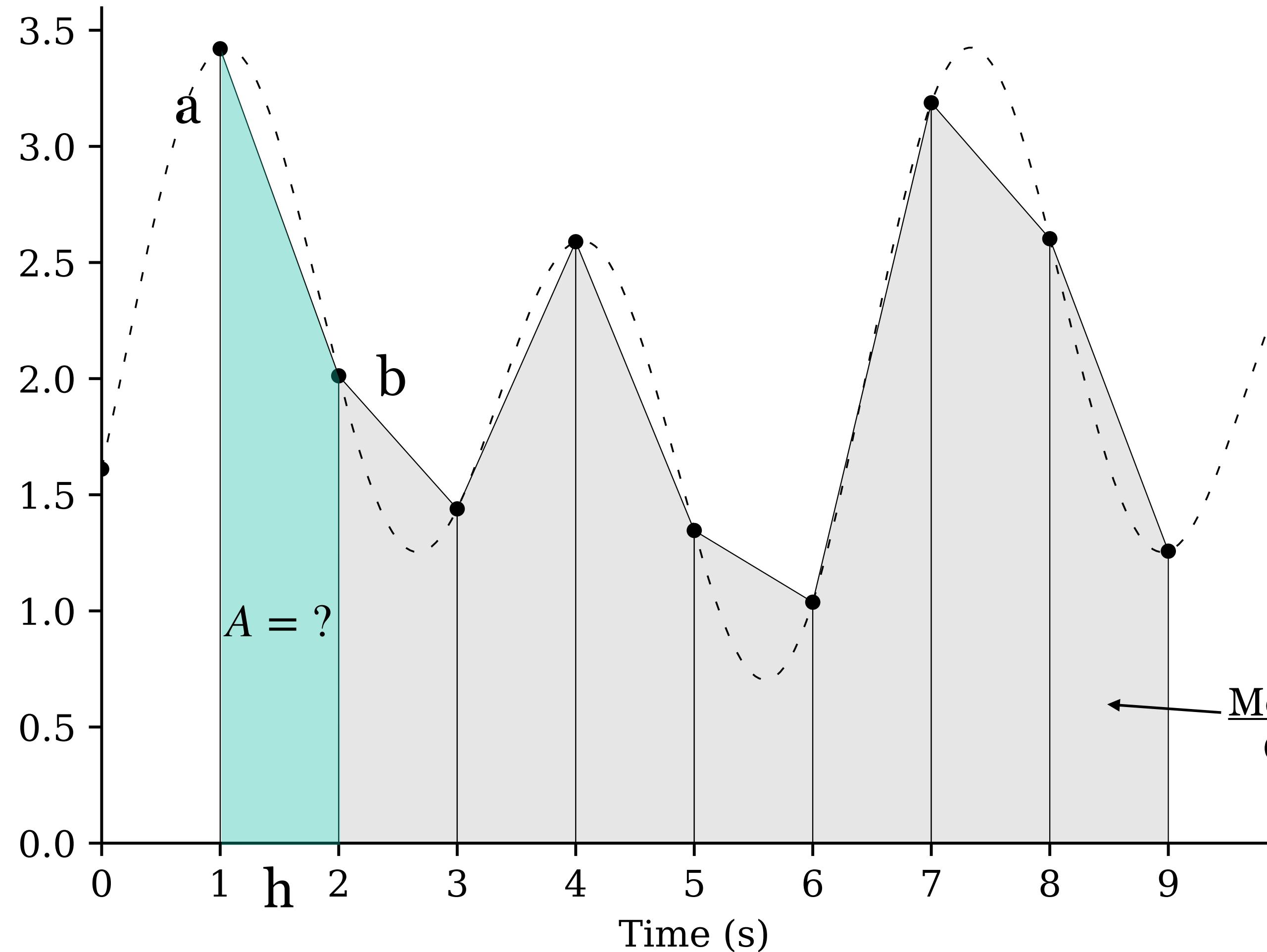






$$A = \frac{a + b}{2}h = \frac{P_{t_n} + P_{t_{n+1}}}{2} \cdot \Delta t = \frac{3.5\text{W} + 2.0\text{W}}{2} \cdot 1\text{s} = 2.75\text{J}$$

Power (W)



### Trapezoid Rule

$$\int_{t_0}^{t_n} P(t)dt \approx \frac{\Delta t}{2} [P(t_0) + 2P(t_1) + 2P(t_2) + \dots + 2P(t_{n-1}) + P(t_n)]$$

Power (W)

3.5  
3.0  
2.5  
2.0  
1.5  
1.0  
0.5  
0.0

0 1 2 3 4 5 6 7 8 9 10

Time (s)

Measured Energy Consumption

⚠ Sometimes you cannot assume that the sampling interval ( $\Delta t$ ) is always the same. ⚠

### **3. Use case**

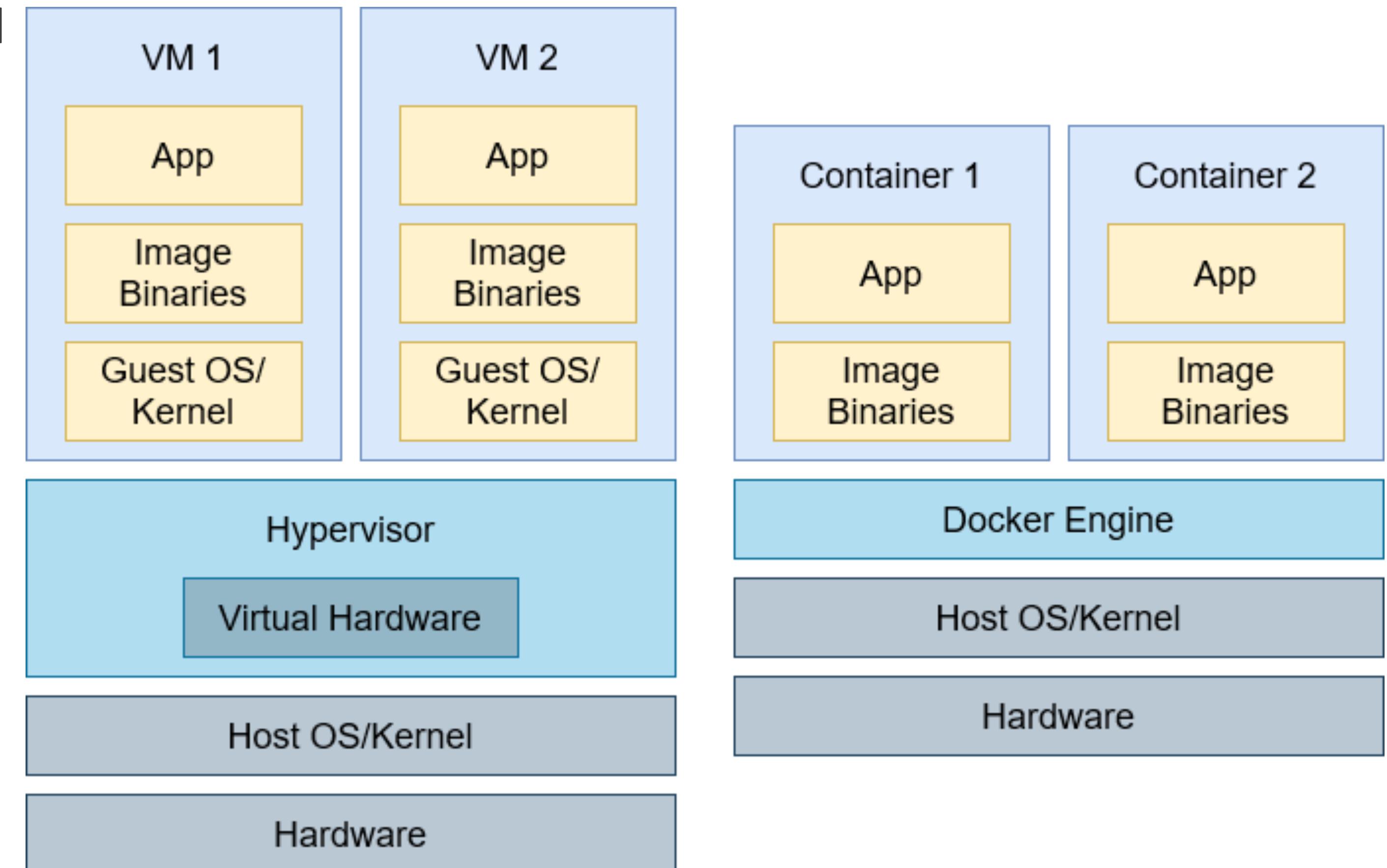
**Discovering energy inefficiencies in Docker through tracing: A case study with Redis**

# Background

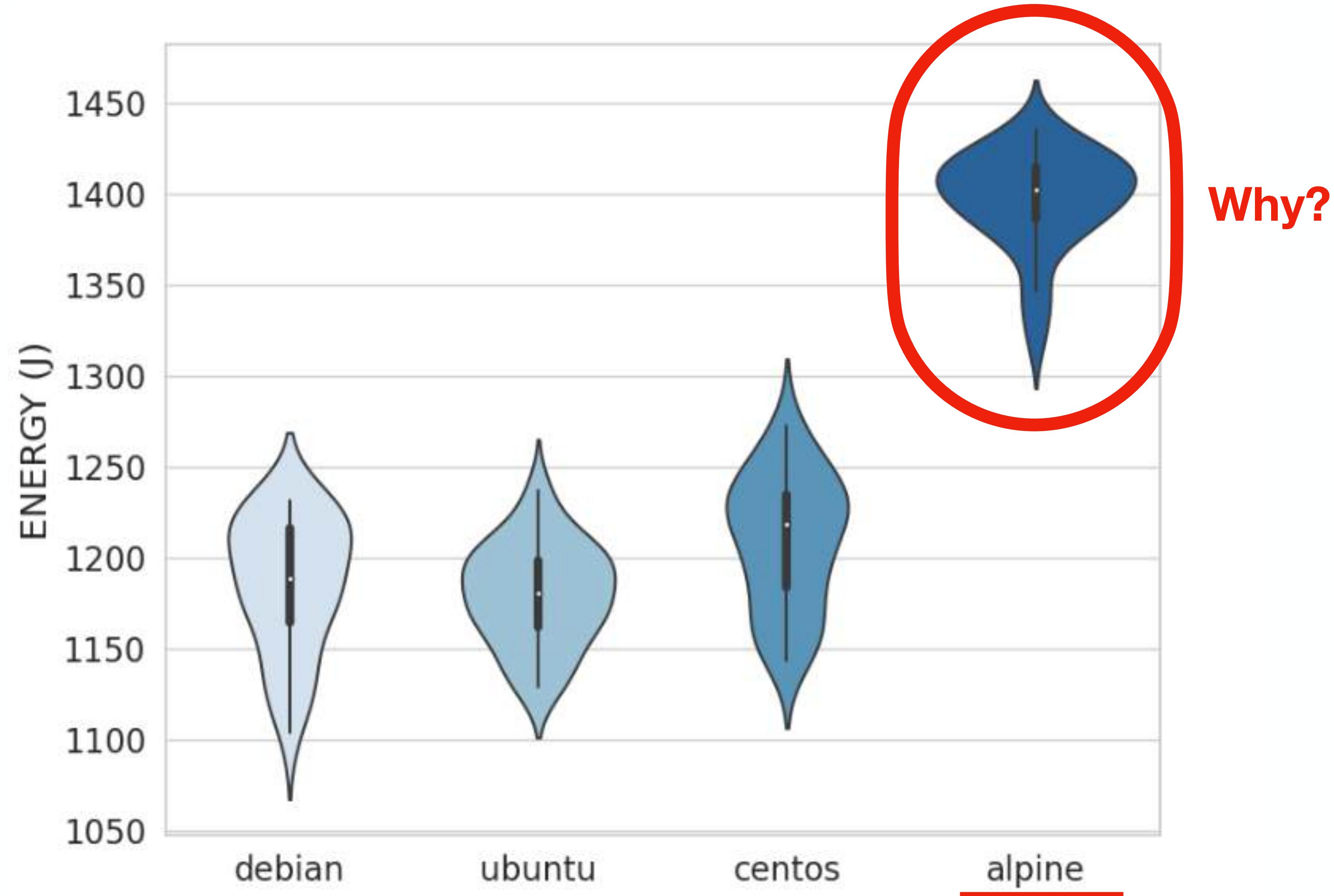
- Virtualization: Technique to isolate and scale workloads
- Two main variants: Hypervisor and Containers

## We know that:

- Virtualization introduces performance and energy overhead
- Containerization is more lightweight and efficient



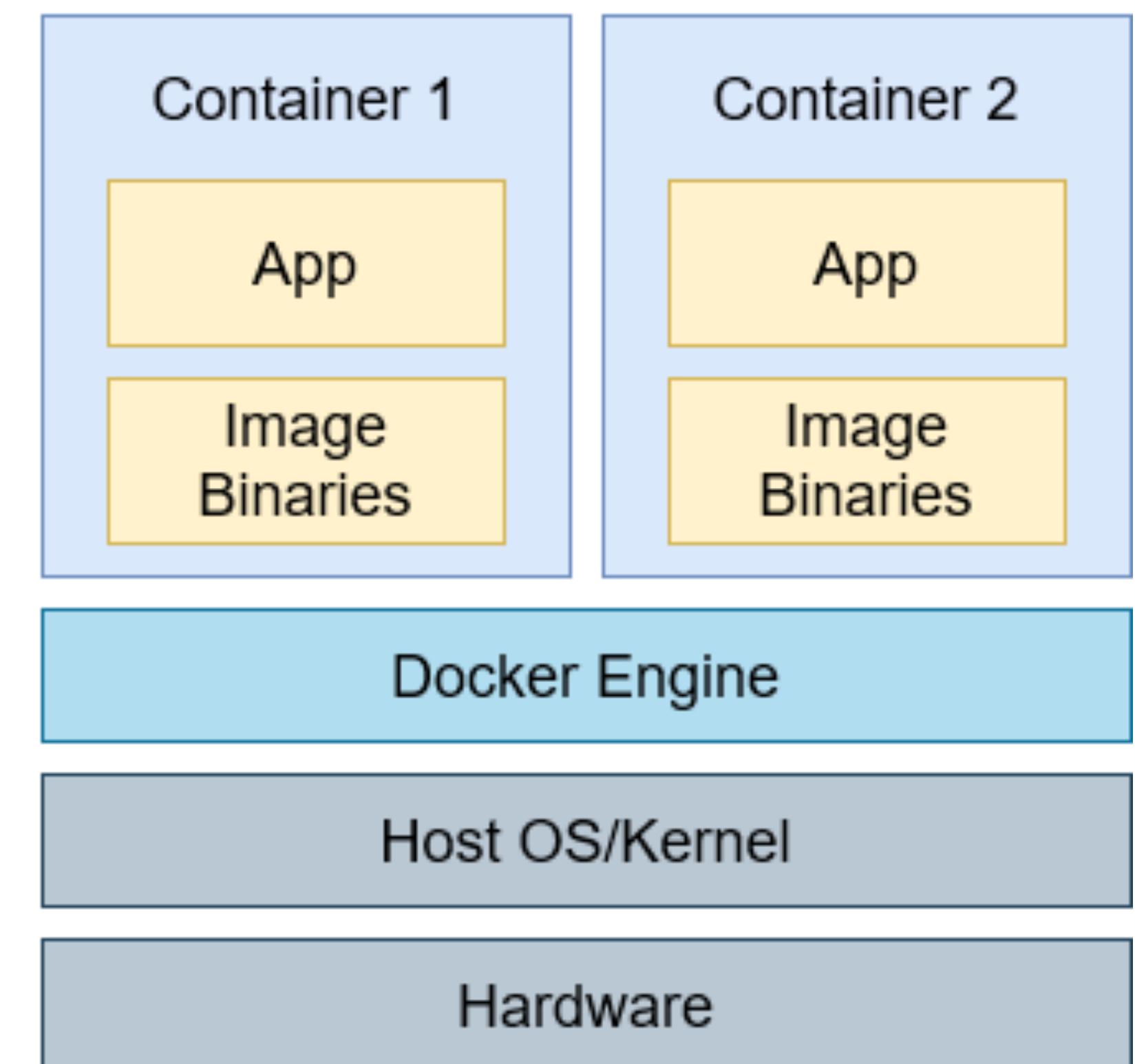
**What is the impact of using different base images in Docker containers?**

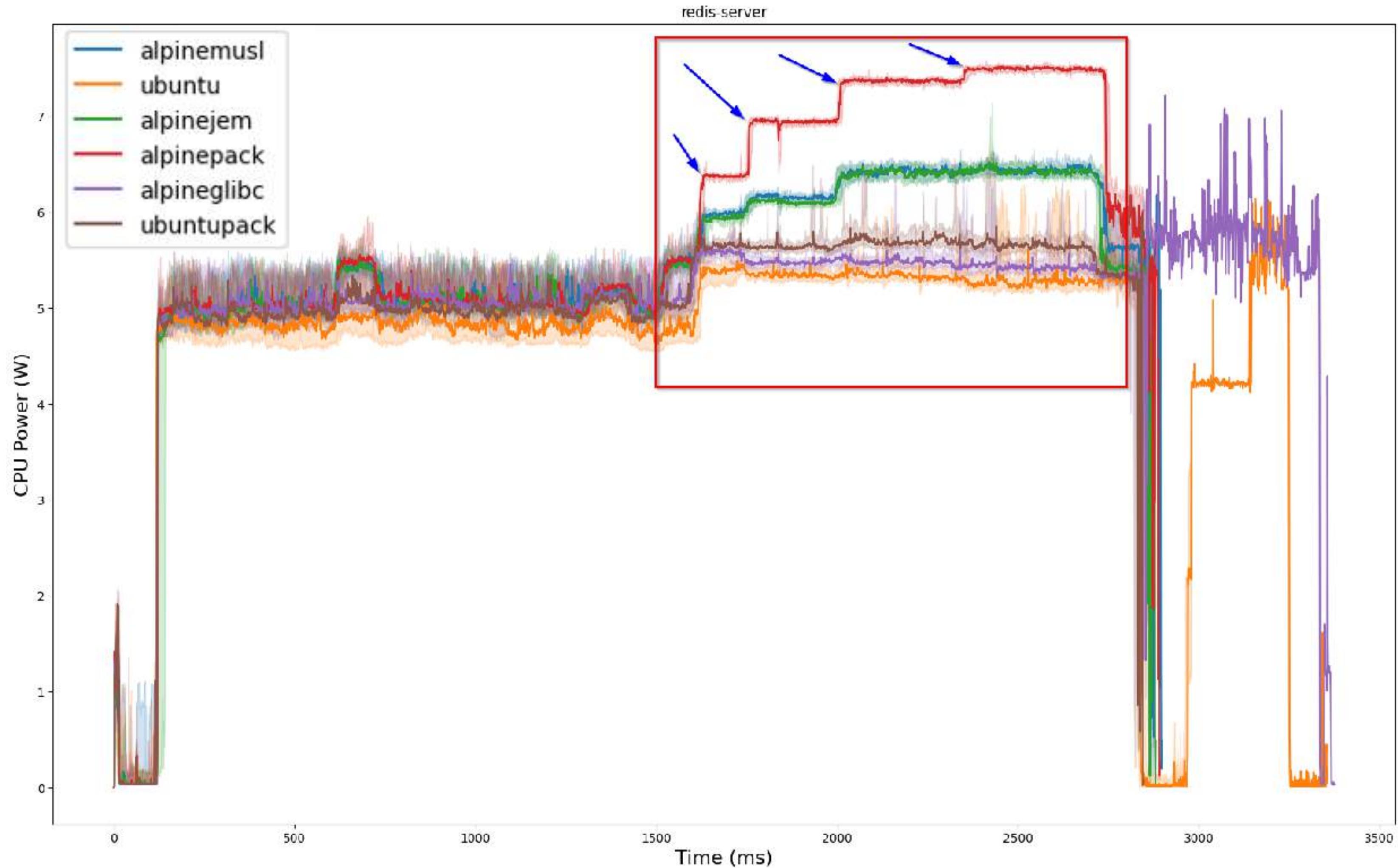


Why?

# Why is Alpine more energy inefficient than others?

(Alpine is known to be a simple and lightweight linux distribution)



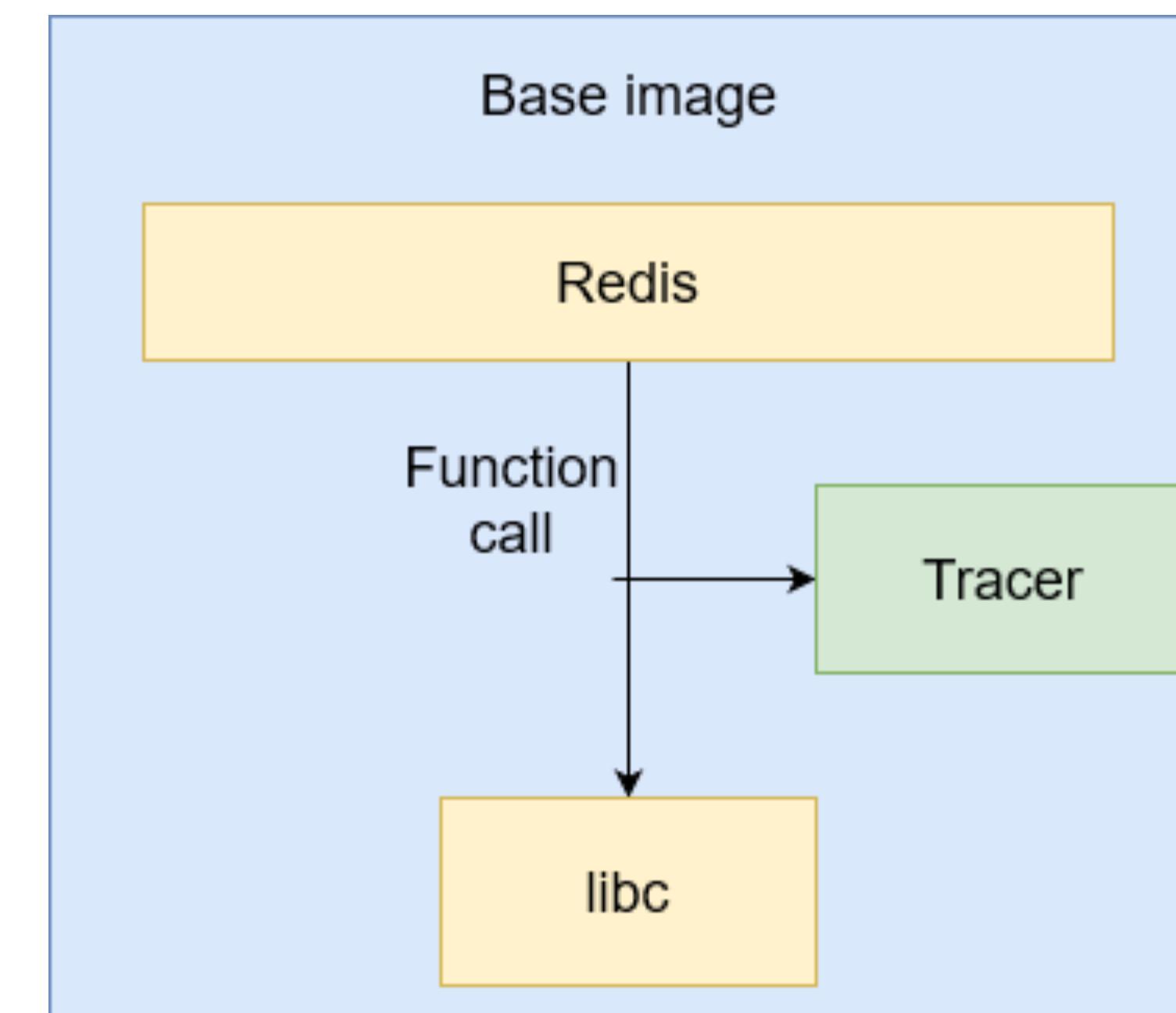


```
===== LRANGE_100 (first 100 elements) =====
1708946306 client-1 | 1000000 req/s
1708946306 client-1 | 50 parallel
1708946306 client-1 | 3 bytes payload
1708946306 client-1 | keep alive: 0
1708946306 client-1 | host config: 1
1708946306 client-1 | host config: 1
1708946306 client-1 | multi-threaded
1708946306 client-1 | Latency by percentile:
1708946306 client-1 | 0.000% <= 0.300µs
1708946306 client-1 | 50.000% <= 1.100µs
1708946306 client-1 | 75.000% <= 1.900µs
1708946306 client-1 | 87.500% <= 2.100µs
1708946306 client-1 | 93.750% <= 2.400µs
1708946306 client-1 | 96.875% <= 3.100µs
1708946306 client-1 | 98.438% <= 5.000µs
1708946306 client-1 | 99.219% <= 6.100µs
1708946306 client-1 | 99.609% <= 7.000µs
1708946306 client-1 | 99.805% <= 9.000µs
1708946306 client-1 | 100.000% <= 11.000µs
```

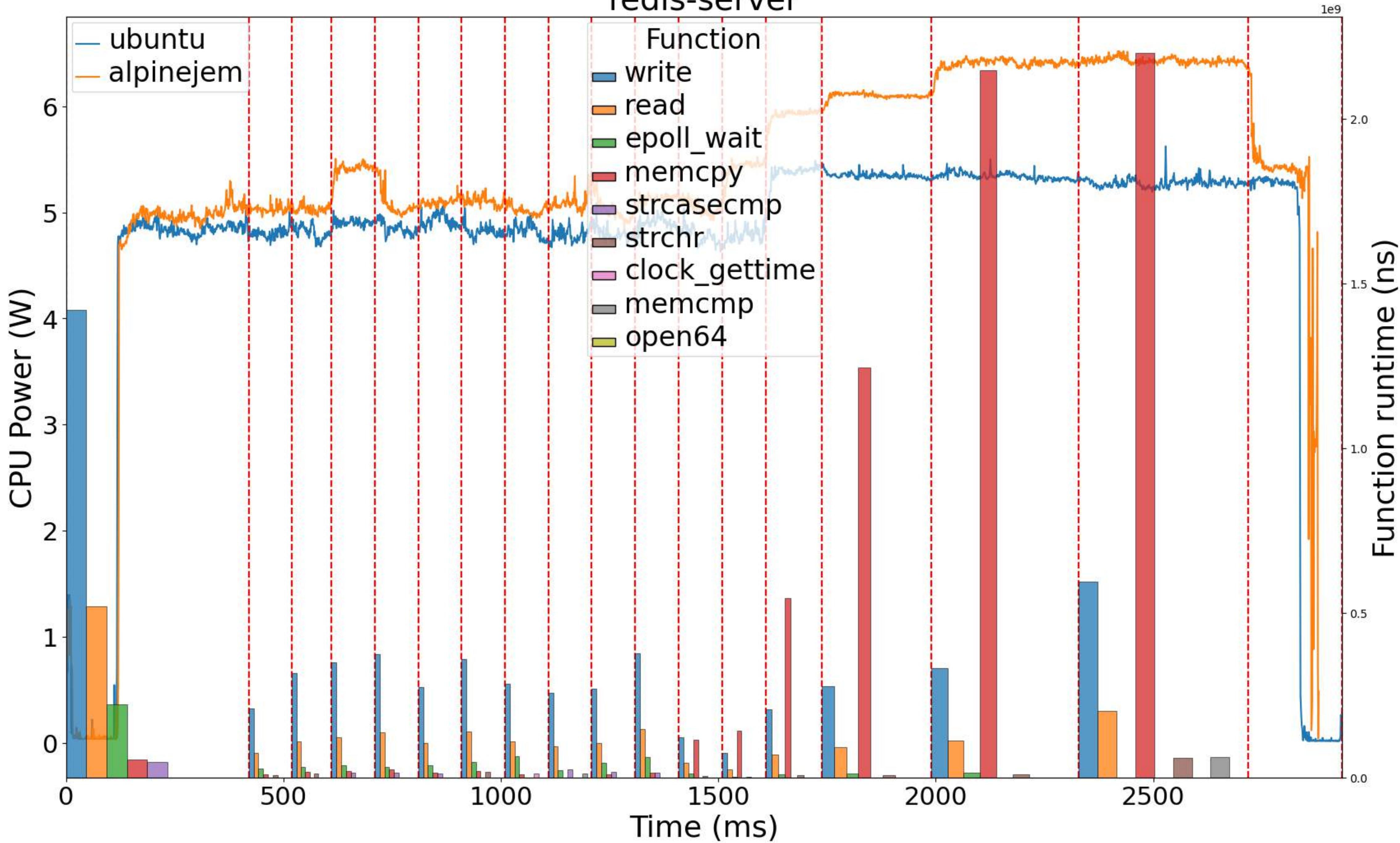
What is behind these step changes in power consumption?

# Tracing

- We want to go further and find the specific function/features that cause this difference
- Tracing: Record calls made by the workload to shared libraries in the distribution
- Problem: tracing adds a non-linear **overhead**!



# redis-server



# Long story short:

- **memcpy** is implemented by a library called **libc**
- **libc** has multiple implementations that are used by different distributions. **Alpine** uses the **musl libc** implementation; **Ubuntu** uses the **glibc** implementation.
- **musl** was implemented with the goal of being **simple** and **easy to maintain**. It **misses several optimisations** and that can explain the different power traces.

# **4. Energy Efficiency vs Code Quality**

# Do Energy-oriented Changes Hinder Maintainability?



U.PORTO



2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)

## Do Energy-oriented Changes Hinder Maintainability?

Luis Cruz<sup>\*</sup>, Rui Abreu<sup>†</sup>, John Grundy<sup>‡</sup>, Li Li<sup>‡</sup> and Xin Xia<sup>‡</sup>

<sup>\*</sup> INESC-TEC, University of Porto, Porto, Portugal

Email: lvcruz@fe.up.pt

<sup>†</sup> INESC-ID, University of Lisbon, Lisbon, Portugal

Email: rui@computer.org

<sup>‡</sup> Faculty of Information Technology, Monash University, Melbourne, Australia

Email: [john.grundy, li.li, xin.xia]@monash.edu

**Abstract**—Energy efficiency is a crucial quality requirement for mobile applications. However, improving energy efficiency is far from trivial as developers lack the knowledge and tools to aid in this activity. In this paper we study the impact of changes to improve energy efficiency on the maintainability of Android applications. Using a dataset containing 559 energy efficiency-oriented commits, we measure maintainability – as computed by the Software Improvement Group’s web-based source code analysis service *Better Code Hub* (BCH) – before and after energy efficiency-related code changes. Results show that in general improving energy efficiency comes with a significant decrease in maintainability. This is particularly evident in code changes to accommodate the *Power Save Mode* and *Wakeful Addition* energy patterns. In addition, we perform manual analysis to assess how real examples of energy-oriented changes affect maintainability. Our results help mobile app developers to 1) avoid common maintainability issues when improving the energy efficiency of their apps; and 2) adopt development processes to build maintainable and energy-efficient code. We also support researchers by identifying challenges in mobile app development that still need to be addressed.

**Index Terms**—Energy Consumption, Software Maintenance, Mobile Computing

### I. INTRODUCTION

Modern mobile applications, popularly known as apps, provide users with a number of features in multi-purpose mobile computing devices – smartphones. The convenience of using smartphones to pervasively accomplish important daily tasks has a big limitation: smartphones have a limited battery life. Apps that drain battery life of smartphones can ruin user experience, and are likely to be uninstalled unless they offer a key feature.

Thus, it is critically important that apps efficiently use the battery of smartphones. However, many developers still lack knowledge about best practices to deliver energy efficient mobile applications [1], [2]. Important efforts have been carried out to help developers ship energy efficient mobile apps [3]. Novel tools have been built to suggest energy improvements to the codebases of mobile apps [4]–[7] and to help developers measure the energy consumption of their apps [8]–[12].

Despite these efforts, improving the energy efficiency of mobile applications is not a trivial task. It requires implementing new features and refactoring existing ones [13], only for the sake of better energy usage, i.e., predominantly a non-functional rather than functional change. However, the extent to which these changes affect the maintainability of the

mobile app software has not yet been studied. In this work, we are interested in studying the trade-off between the energy efficiency and the maintainability of mobile applications.

The International Standards on software quality ISO/IEC 7810 define software maintainability as “the degree of effectiveness and efficiency with which a software product or system can be modified to improve it, correct it or adapt it to changes in environment and in requirements” [14]. The standard defines the core characteristics of maintainability: modularity, reusability, analyzability, modifiability, and testability. The Software Improvement Group (SIG) has developed a web-based source code analysis toolset *Better Code Hub* (BCH) [15] that maps the ISO/IEC 25010 standard on maintainability into a set of 10 guidelines, such as *write short units of code* and *write code once*, derived from static analysis [15]–[18]. The code metrics used by the SIG model were empirically validated in previous work [19]. We use this toolset in our work to provide an assessment of maintainability in mobile app codebases.

Specifically, we want to explore whether there is a trade-off between applying energy efficiency patterns and keeping the maintainability of the apps, i.e., does improving energy efficiency have a negative impact on code maintainability? In this paper, we present the results of our analysis on the maintainability using 559 energy commits harvested from open source Android applications.

The key contributions of this work are:

- An empirical investigation of the impact of energy patterns on code maintainability.
- A dataset of energy commits and respective impact on maintainability.
- A software package with all scripts used in our experiments and a dataset of energy commits with respective impact on maintainability for reproducibility. Available here: <https://figshare.com/s/589c5102ae6a8423654d>.

Our empirical study finds evidence that energy efficiency-oriented code changes have a negative impact on code maintainability. In particular, careful thinking is required to implement the energy patterns *Power Save Mode* and *Wakeful Addition*. Furthermore, we show that energy patterns are more likely to require maintenance than regular code changes.

This paper is structured as follows. In Section II, we introduce an example of an energy improvement from a real-world

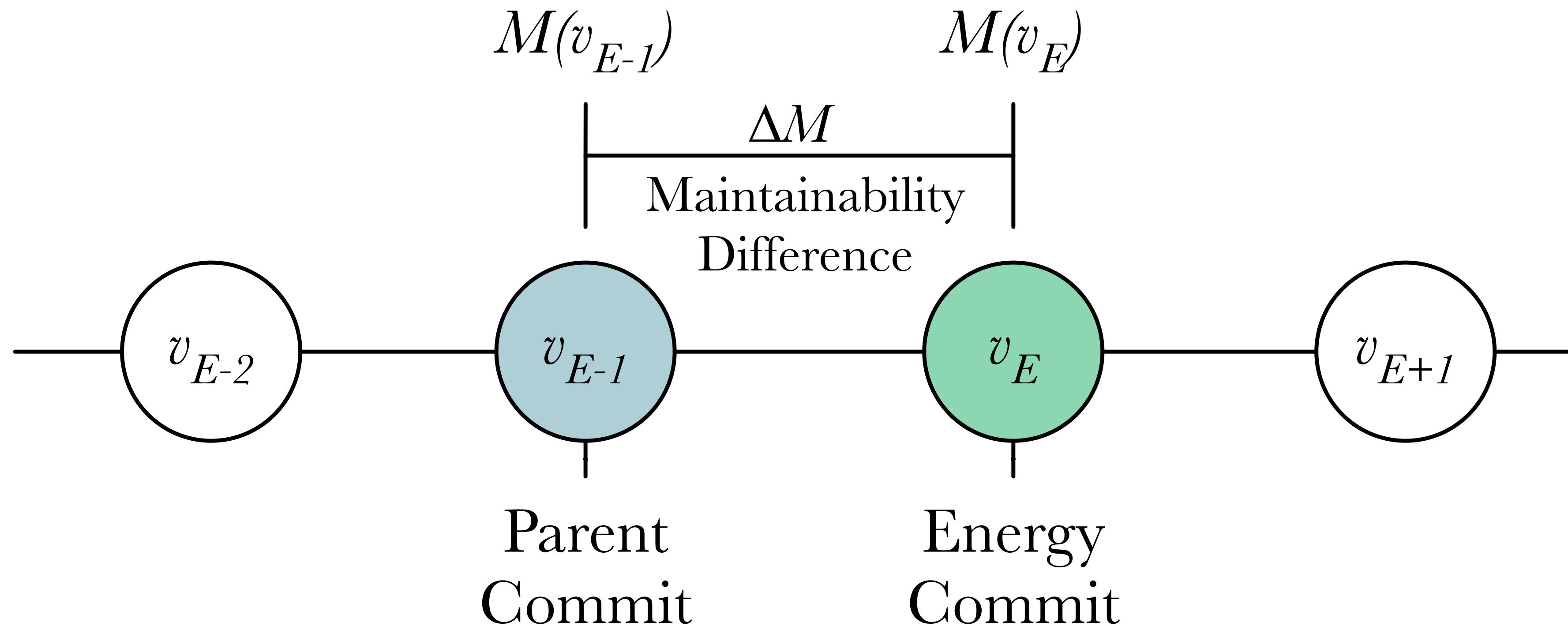
2579-3148/19/\$31.00 ©2019 IEEE  
DOI 10.1109/ICSM.2019.00013

29



# Maintainability of Energy Changes

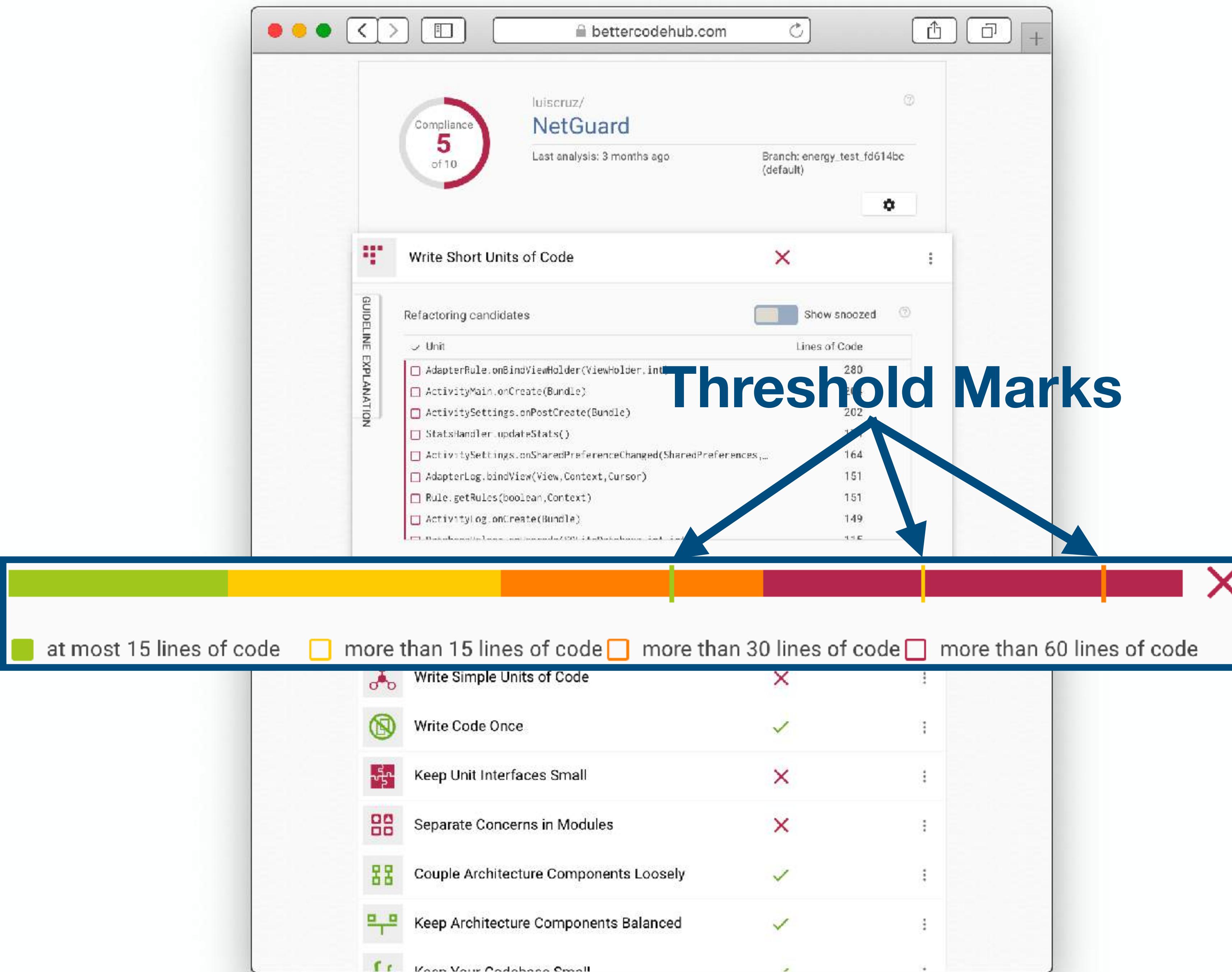
- What is the impact of making energy-oriented code changes on the maintainability of mobile apps?



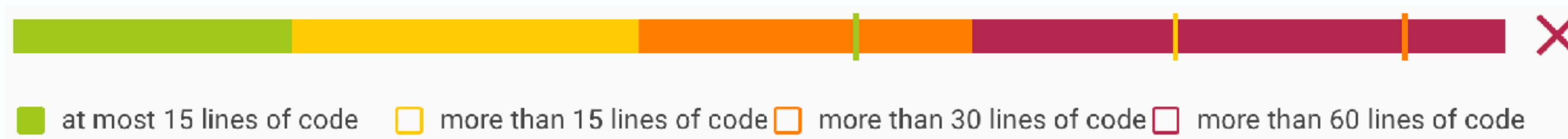
# Measuring Maintainability

- According to ISO/IEC 25010, Maintainability is “*the degree of effectiveness and efficiency with which a software product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements*”
- We use the code analysis tool Better Code Hub to assess maintainability
- Better Code Hub maps the ISO/IEC 25010 standard on maintainability into a set of guidelines derived from static analysis





# Numerical Score of Maintainability for a Software Version



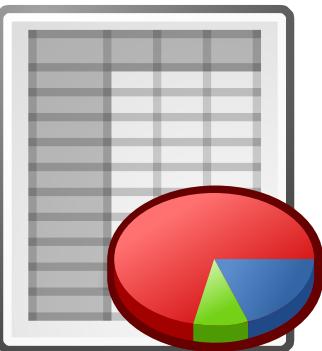
$$\begin{aligned} \text{Maintainability} = & \frac{1}{3} \left( SLOC_{green} - \frac{1 - T_{yellow}}{T_{yellow}} (SLOC_{yellow} + SLOC_{orange} + SLOC_{red}) + \right. \\ & + SLOC_{green} + SLOC_{yellow} - \frac{1 - T_{orange}}{T_{orange}} (SLOC_{orange} + SLOC_{red}) + \\ & \left. + SLOC_{green} + SLOC_{yellow} + SLOC_{orange} - \frac{1 - T_{red}}{T_{red}} SLOC_{red} \right) \end{aligned}$$

Explained in the paper

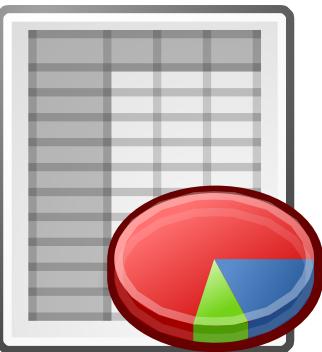
# Energy Code Changes

## Dataset

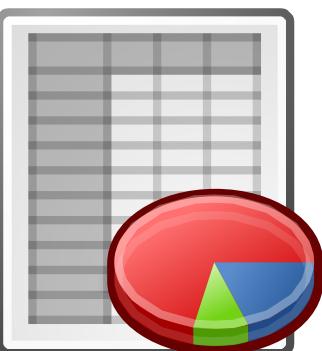
Bao et al.  
(2015)



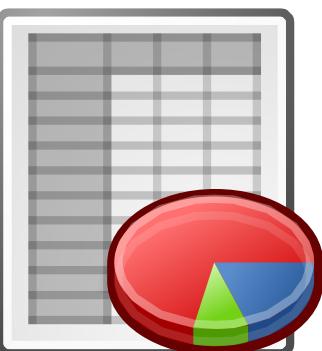
Moura et al.  
(2016)



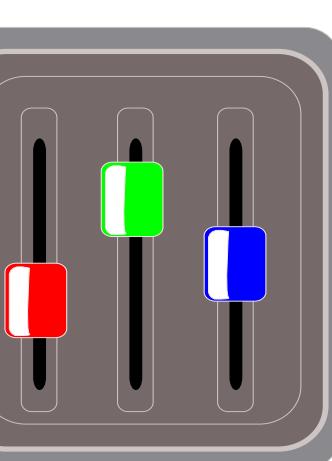
Cruz et al.  
(2018)



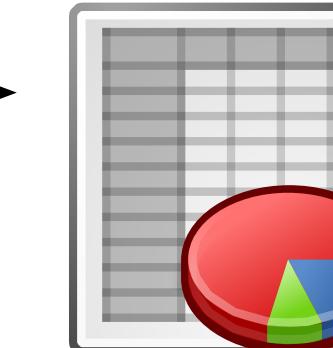
Cruz et al.  
(2019)



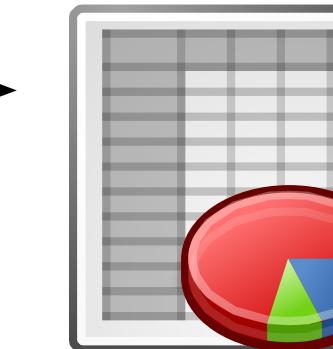
Combine  
datasets



Energy  
Commits

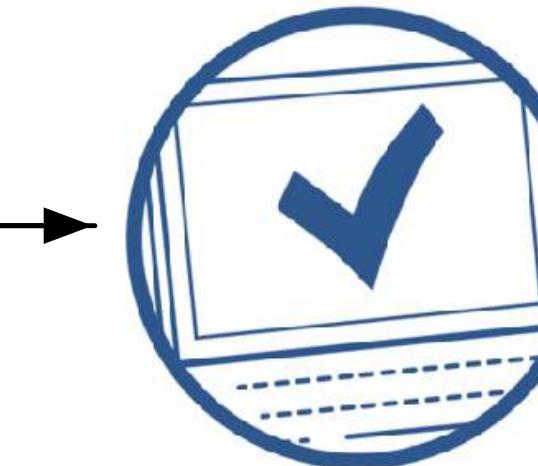


Baseline  
Commits



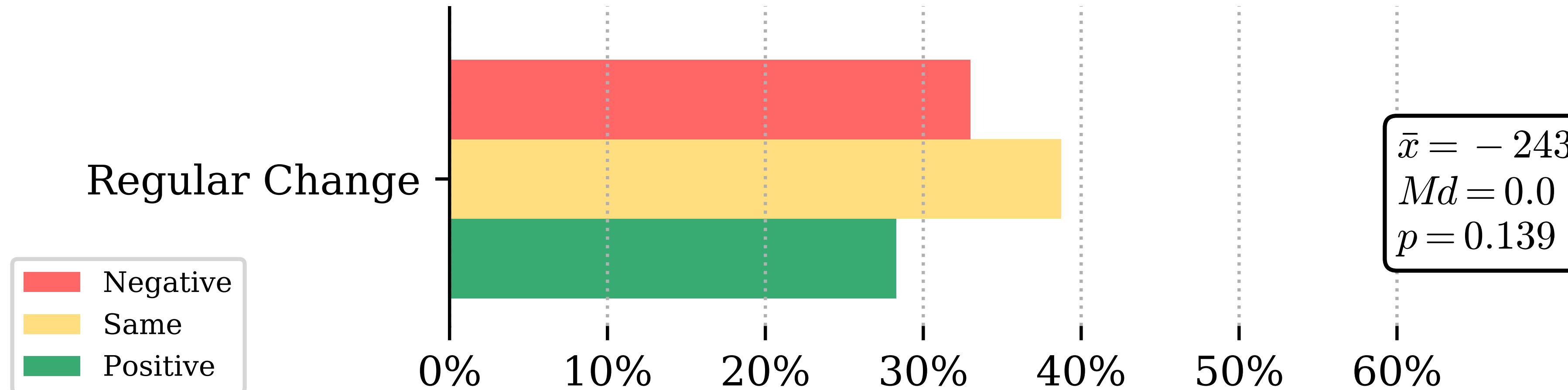
**539 commits  
from 306 mobile apps**

Better Code Hub  
Maintainability

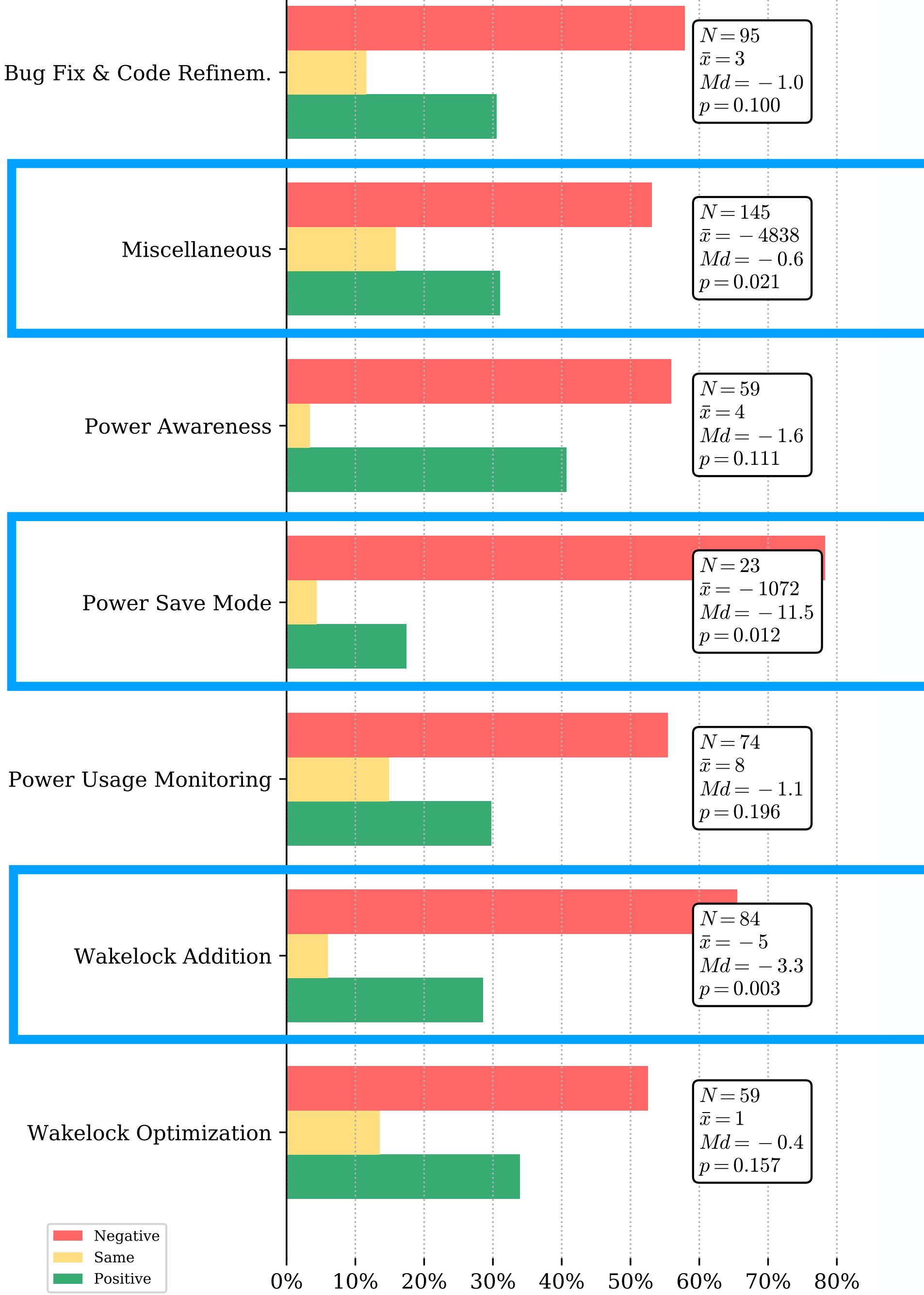


**539 baseline commits  
from 306 mobile apps**

# Impact of energy changes on maintainability



# Which energy patterns are more likely to affect maintainability?



# Typical maintainability issue I

<https://github.com/einmalfel/PodListen/commit/2ed5a65>

4 changed files with 28 additions and 0 deletions.



```
...
234     private synchronized void readPreference(Key key) {
235         switch (key) {
236             +     case AUTO_DOWNLOAD_AC:
237                 +         autoDownloadACOnly = sPrefs.getBoolean(Key.AUTO_DOWNLOAD_AC.toString(), false);
238                 +         if (!autoDownloadACOnly) {
239                     +             context.sendBroadcast(new Intent(DownloadReceiver.UPDATE_QUEUE_ACTION));
240                 +         } else if (!DownloadReceiver.isDeviceCharging()) {
241                     +             DownloadReceiver.stopDownloads(null);
242                 +         }
243                 +         break;
244             case PLAYER_FOREGROUND:
245                 playerForeground = sPrefs.getBoolean(Key.PLAYER_FOREGROUND.toString(), false);
246                 break;
...
}
```

# Typical maintainability issue II

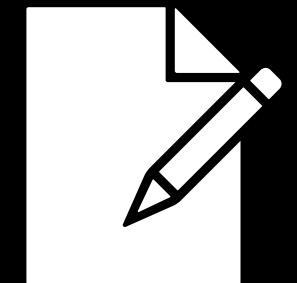
<https://github.com/mozilla/MozStumbler/commit/6ea0268>

**5 changed files with 66 additions and 14 deletions.**



```
161 +
162 +         // each time we reconnect, check to see if we're suppose to be
163 +         // in power saving mode.  if not, start the scanning. TODO: we
164 +         // shouldn't just stopScanning if we find that we are in PSM.
165 +         // Instead, we should see if we were scanning do to an activity
166 +         // recognition.  If we were, don't stop.
167 +         if (mConnectionRemote != null) {
168 +             try {
169 +                 if (mPrefs.getPowerSavingMode()) {
170 +                     mConnectionRemote.stopScanning();
171 +                 } else {
172 +                     mConnectionRemote.startScanning();
173 +                 }
174 +             } catch (RemoteException e) {
175 +                 Log.e(LOGTAG, "", e);
176 +             }
177 +         }
178     updateUI();
179 }
```

What was your biggest  
takeaway?



# CSE2115 Software Engineering Methods

**Software at Scale**  
Taico Aerts & Arie van Deursen

# What would happen?

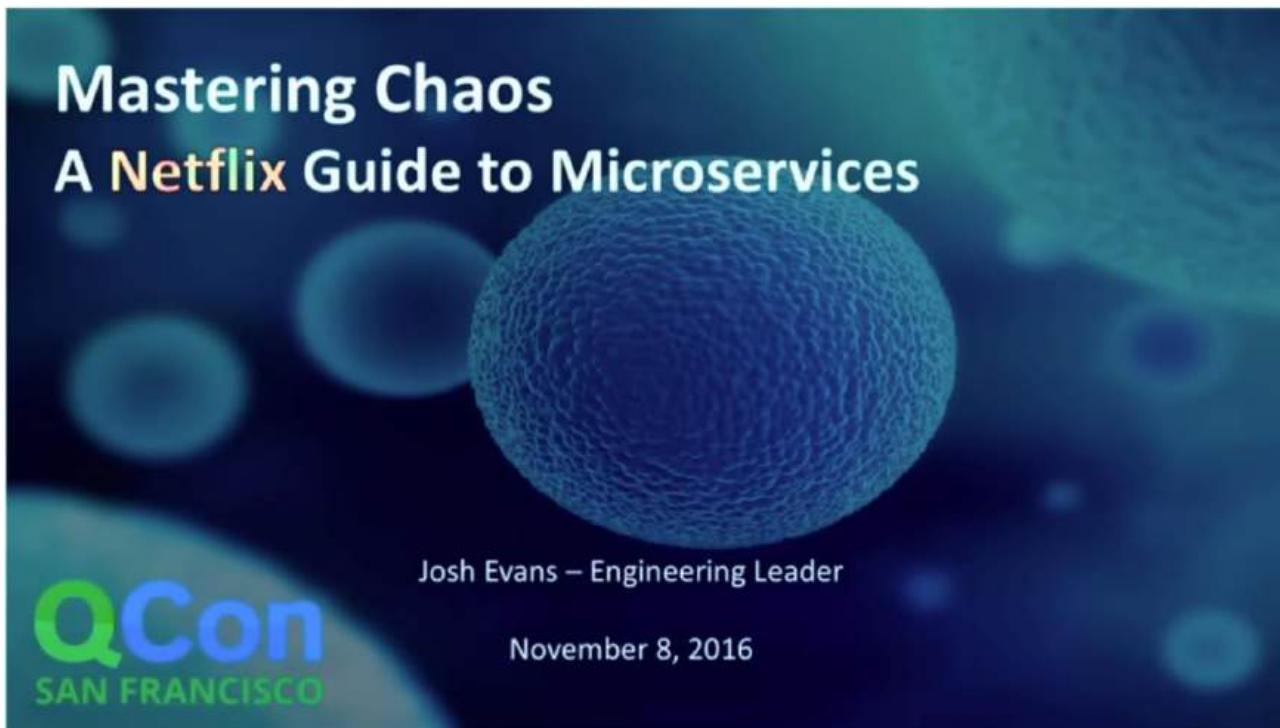
- A popular Y1 CSE course is doing sign-offs with timeslots in Queue
  - For mandatory assignments
- There are not enough timeslots
- What do CSE students do to solve this?

# Learning Objectives

- Use application monitoring software to analyse errors and performance of applications
- Know about different caching techniques to improve scalability
- Know different ways of scaling your application and infrastructure (Scale Cube)
- Convert existing systems to a microservice architecture
- Know how to use Docker to create images as container blueprints

# Further Reading

- Mastering Chaos – A Netflix Guide to Microservices  
(<https://youtu.be/CZ3wluvmHeM>)



# Outline

- Queue Showcase
- Medium-scale and bigger
  - Analyzing Issues at Scale
  - Debugging at Scale
  - Scaling up
- Large-scale
  - Microservices
  - Deploying to the cloud

# Last Time

# Software Quality

- Technical Debt
  - Really slows down software development
  - Increases likelihood of bugs
  - Measuring software quality is important. Various indicators exist (but indicators are not foolproof)

# Sustainability Metrics

- Sustainable Software Engineering
  - Sustainable software has more purposes than just environmental (cost, battery life, ...)
  - Energy efficiency and maintainability can be opposing concerns

# **Medium-Scale Software**

# Lab: Analysis of EIP Software

- What are your experiences with looking at Queue/Submit/TAM code?
- They are complex!
- Reading code is hard
- Understanding even the outlines is hard
- Making software is hard

# DDoSsing Queue (by accident?)

- A popular Y1 CSE course is doing sign-offs with timeslots
- There are not enough timeslots
- What do CSE students do to solve this?

# DDoSing Queue

- Students made a script to try to automatically enqueue
  - Retry every 5 ms
- With a few students doing this: total load increased several orders of magnitude (x100-1000)
  - Problem?
  - Yes, but also no
    - Queue itself can handle this!
    - Still it became unresponsive on most pages

# DDoSing Queue

- What did Queue do when loading the lab page?
  - Labracore: Request Person Information
  - Labracore: Check role in course edition
  - Labracore: Get group information
  - Labracore: Get basic session (lab) information
  - Database: Get additional session information
- Some of these requests happen multiple times

# DDoSing Queue

- For every page load, queue used to do 20-40 requests to Labracore
  - DDoS amplification!
  - Labracore is hit ~30x as hard as Queue is



# DDoSing Queue

- Labracore started queueing requests, and timing out on some of them
  - Long delays → People start refreshing
  - Errors → People retry
    - Even if only last labracore request fails, all others were useless and Queue still gives an error page!
  - So... problem just gets worse!

# DDoSing Queue

- Why so many requests, some multiple times?
- Maintainability / Small oversight
  - Code used `XYZService.getXyz(...)` in multiple different places
  - E.g. `AssignmentService.getAssignment(...)`
  - This was implemented to always fetch this information from Labracore
- On development (local labracore with small database), this is near-instant and not noticeable

# DDoSing Queue

- Why so many requests, some multiple times?
- Maintainability / Small oversight
  - Code used XYZService.get() in many places
  - E.g. This simple oversight alone was enough to cause the whole system to go down!
  - The developer implemented to always fetch this information from Labracore
- On development (local labracore with small database), this is near-instant and not noticeable

# DDoSing Queue

- What Scale is Queue meant to handle?
- How to analyze these situations?
- How to resolve this?
- How to prevent this?

# Analyzing Performance Issues @ Scale

# Sentry

- Application Monitoring Software
  - Error Tracking
  - Performance Monitoring



# Performance Metrics with Sentry



# Local Analysis

- Seeding with a lot of data
- Clicking around yourself
- Can be used to find paths that are always slow
- Generally less efficient than based on production monitoring

# Continuous Integration (CI) Analysis

- Pipeline steps to measure performance
  - Integration Test performance metrics
  - System Tests
  - Static page load times (e.g. Sitespeed.io)
- Good at preventing slow changes from making it into production.
- Mainly finds performance issues in common scenarios.

# Debugging @ Scale

# From Problem to Developer

- Report from student (customer):  
“The group enrolment doesn’t work.”
- How should an organisation resolve this?
  - Customer support (filter out incorrect usage)
- What if it is a real issue?

# Reproducing Issues

- You test on the development instance, everything works fine
- You test in production, it works fine
- What could be causing these issues?
  - Configuration of related entities
  - Corruption of entity
  - ... who knows ...

# Reproducing Issues

- You test on the development instance, everything works fine
- You test in production, it fails
- What causes these issues?
  - Configuration of related entities
  - Corruption of entity
  - ... who knows ...

# Use Production Data Locally

- More representative of circumstances on production
- Any concerns?
  - Privacy
  - Security
  - Size (production DB is large!)

# Impersonation

- Try to perform action as the user, more likely to run into the problem
- Any concerns?
  - Who should get this access?
  - Private information
  - Logging of actions  
(impersonation-aware?)



# Extensive Error Tracking

- Sentry
  - Tracking within one application
- Jaeger
  - Distributed across services



JAEGER

# Error Tracking with Sentry

**NullPointerException** POST /lab/8221/enqueue/lab

Cannot invoke "nl.tudelft.queue.model.TimeSlot.getLab()" because the return value of "nl.tudelft.queue.dto.id.TimeSlotIdDTO.apply()" is null

ISSUE # 3 EVENTS 3 USERS 3 ASSIGNEE

✓ Resolve ▾ Ignored ▾ Mark Reviewed 🗑️ ▾ Share ▾ Open in Discover ⚡ 🔍

Details Activity 0 User Feedback 0 Attachments Tags Events Merged Issues

Event Ocebcbbeee1c44aa6965ddde967ce157c | JSON (94.9 KiB) Older Newer Oct 7, 2024 15:28

TAGS

M XXX.XXX.XXX.XXX Username: redacted Firefox Version: 130.0 Linux Version: Unknown

browser Firefox 130.0 browser.name Firefox client\_os.name Linux environment production

level error logger nl.tudelft.queue.controller.ErrorControllerAdvice release a43ed84a3f6f

server\_name queue.tudelft.nl transaction POST /lab/8221/enqueue/lab

url http://queue.tudelft.nl/lab/8221/enqueue/lab user username:michalolszewsk

MESSAGE

A Request (/lab/8221/enqueue/lab) raised an exception

#0 /lab/8221/enqueue/lab

EXCEPTION (most recent call first)

NullPointerException

Cannot invoke "nl.tudelft.queue.model.TimeSlot.getLab()" because the return value of "nl.tudelft.queue.dto.id.TimeSlotIdDTO.apply()" is null

nl.tudelft.queue.dto.create.requests.LabRequestCreateDTO in validate at line 94

Ownership Rules

Create Ownership Rule

All Environments

LAST 24 HOURS

LAST 30 DAYS

LAST SEEN

2 days ago in release a43ed84a3f6f

FIRST SEEN

a month ago in release 262264308d2d

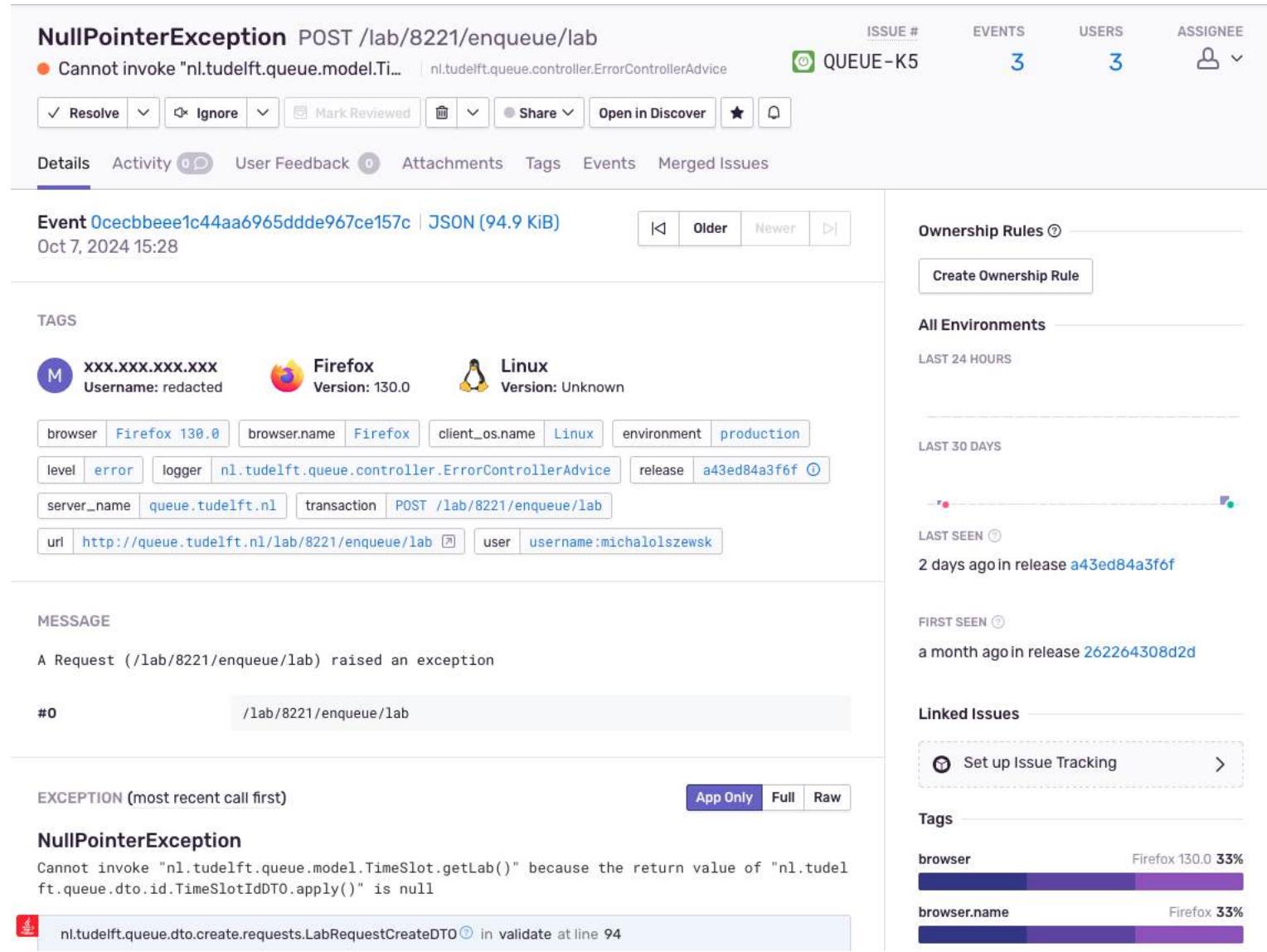
Linked Issues

Set up Issue Tracking >

Tags

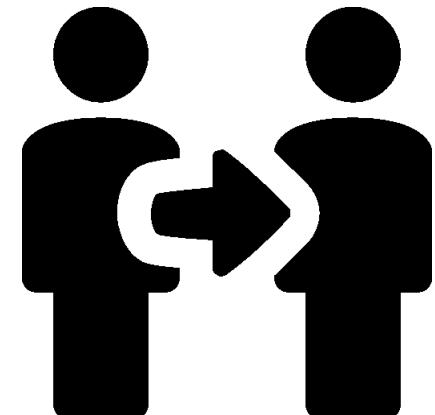
browser Firefox 130.0 33%

browser.name Firefox 33%



# Staging Environments

- Staging Environment matches Production Environment setup
  - Some issues only become apparent when the configuration matches
  - Trying things in an environment is very convenient

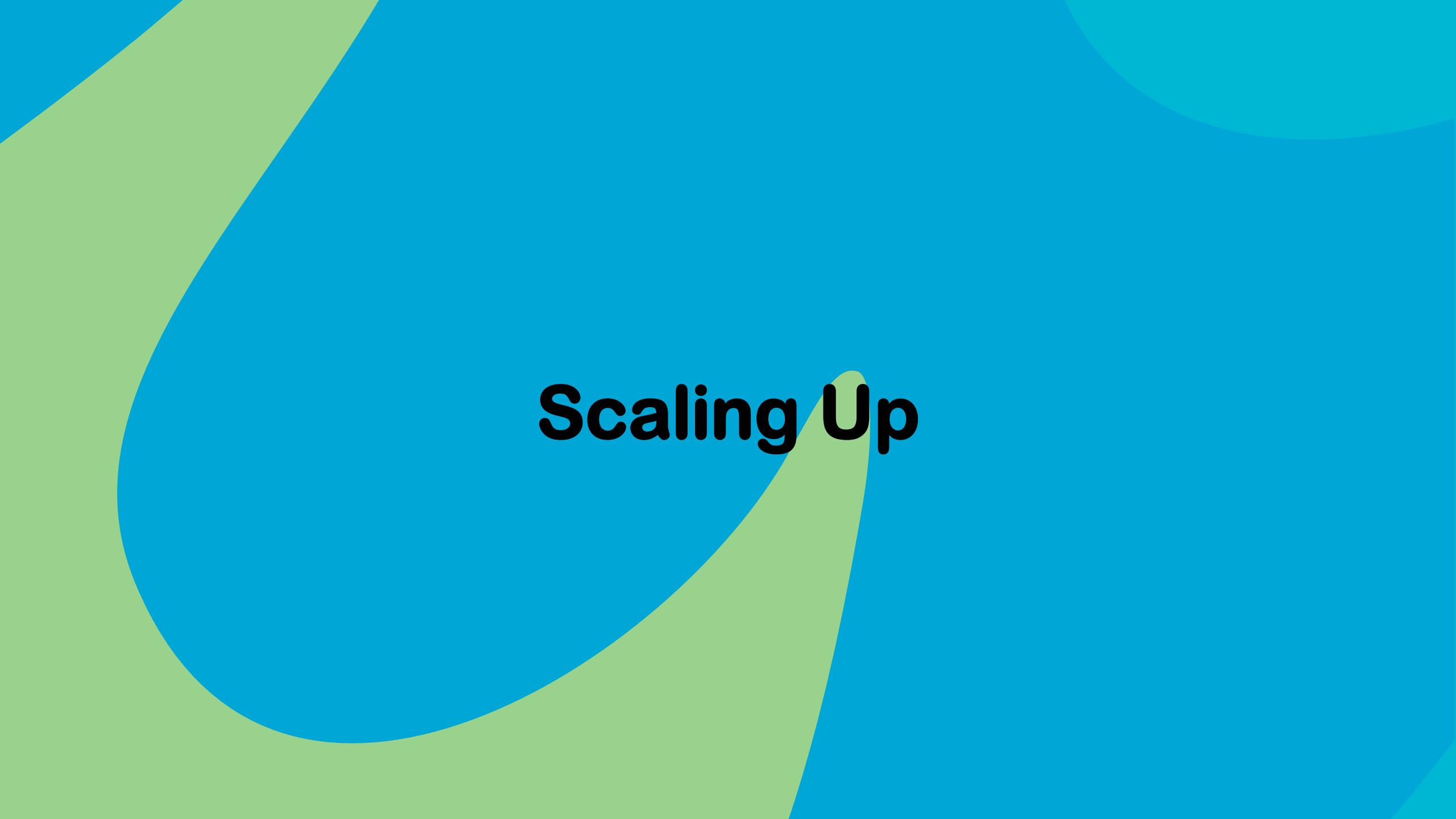


# How to reproduce the issue?

- Impersonation
  - Try to perform action as the user, more likely to run into the problem
- Test locally with production data
  - More representative of circumstances on production
  - Severe privacy concerns
  - Not feasible for large databases
- Extensive error reporting/tracking
  - E.g. Sentry

# Debugging Takeaways

- Reproducing issues of production software is hard
- Systems for tracking and error reporting exist to make this more doable



# Scaling Up

# DDoSing Queue

- Why so many requests, some multiple times?
- Maintainability / Small oversight
  - Code used RoleService.get() in different places
  - This simple oversight alone was enough to cause the whole system to go down!
- On development (local labracore with small database), this is near-instant and not noticeable

# How can we reduce Queue load?

# **Resolution 1: Caching**

# Caching Results

- Caching results in-application
  - In Queue: cache labracore responses
  - In ProjectForum: cache database query results per request (never run the same query twice for 1 request)
- When
  - Loading the same data from other services (or database) multiple times
  - Things that take a while to compute/load

# Caching Results

- Request-level caching is a common approach
  - First request for info goes to service/database/..., result is cached
  - All further requests for the same info go to cache
- Why separate cache per request vs just storing for a while?
  - Inconsistent data (some data newer, some data older)
  - Effects from other users don't matter (don't invalidate cache for updates from other users to data)

# Caching (Partial) Pages

- Caching (parts of) pages
  - In Answers-EWI: cache some shared page elements (e.g. sidebar)
- When
  - Same data is presented to a lot of users
  - Information changes are rare
- Examples?

# Caching

- Concerns
  - How long can you cache things?
  - Invalidation of items in cache after updates/deletion
  - Cache per request or for specific time?
  - Speed of cache lookup vs original request/calculation time.
  - How do we actually cache things?

# Caching – Redis

- Key-value store (map/dictionary)
- In-memory database
  - (Usually) not for permanent storage!
- Open-source
- Ideal for caching

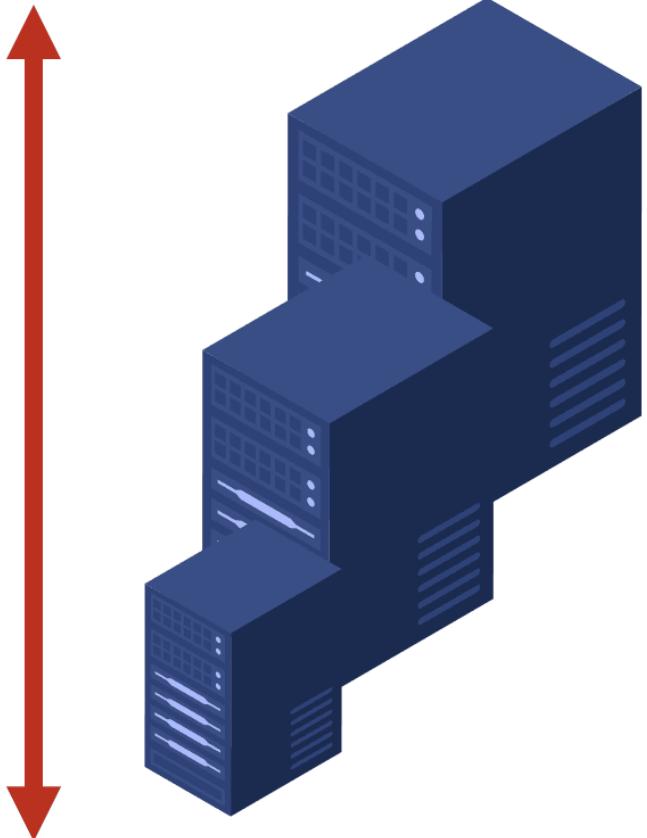


# **Resolution 2: Scaling**

# Scaling

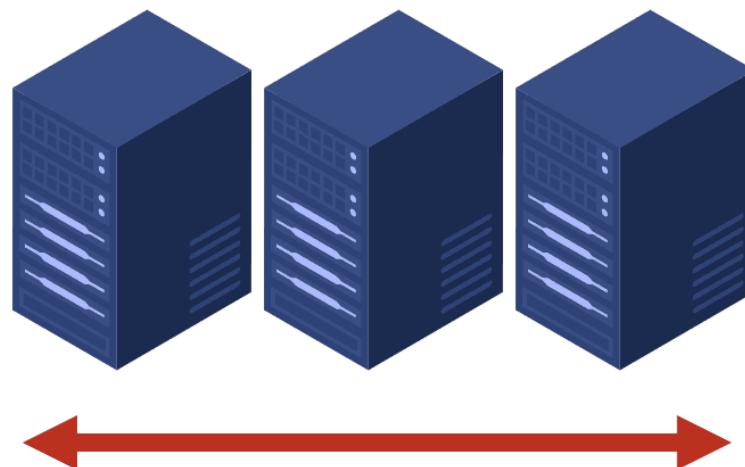
## Vertical Scaling

Increase or decrease the capacity of existing services/instances.



## Horizontal Scaling

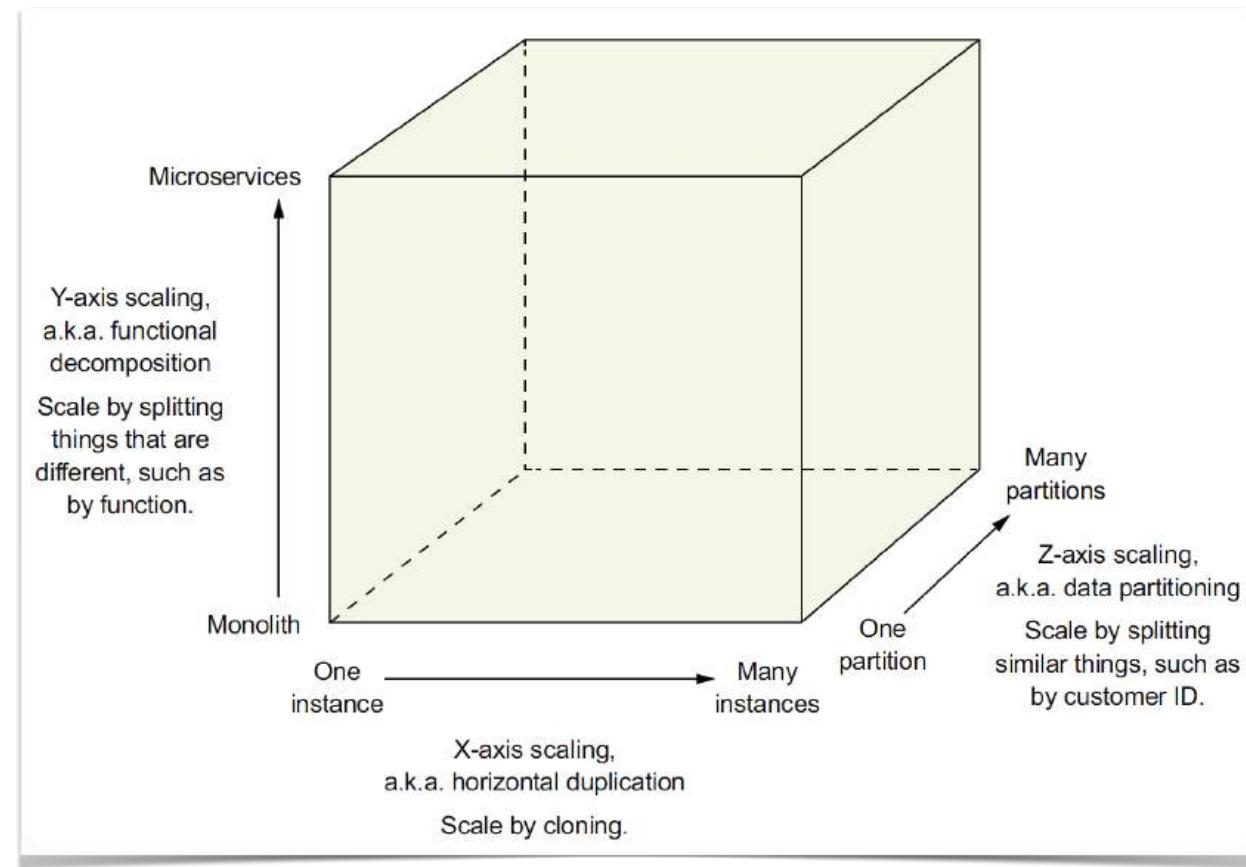
Add more resources like virtual machines to your system to spread out the workload across them.



# The Scale Cube

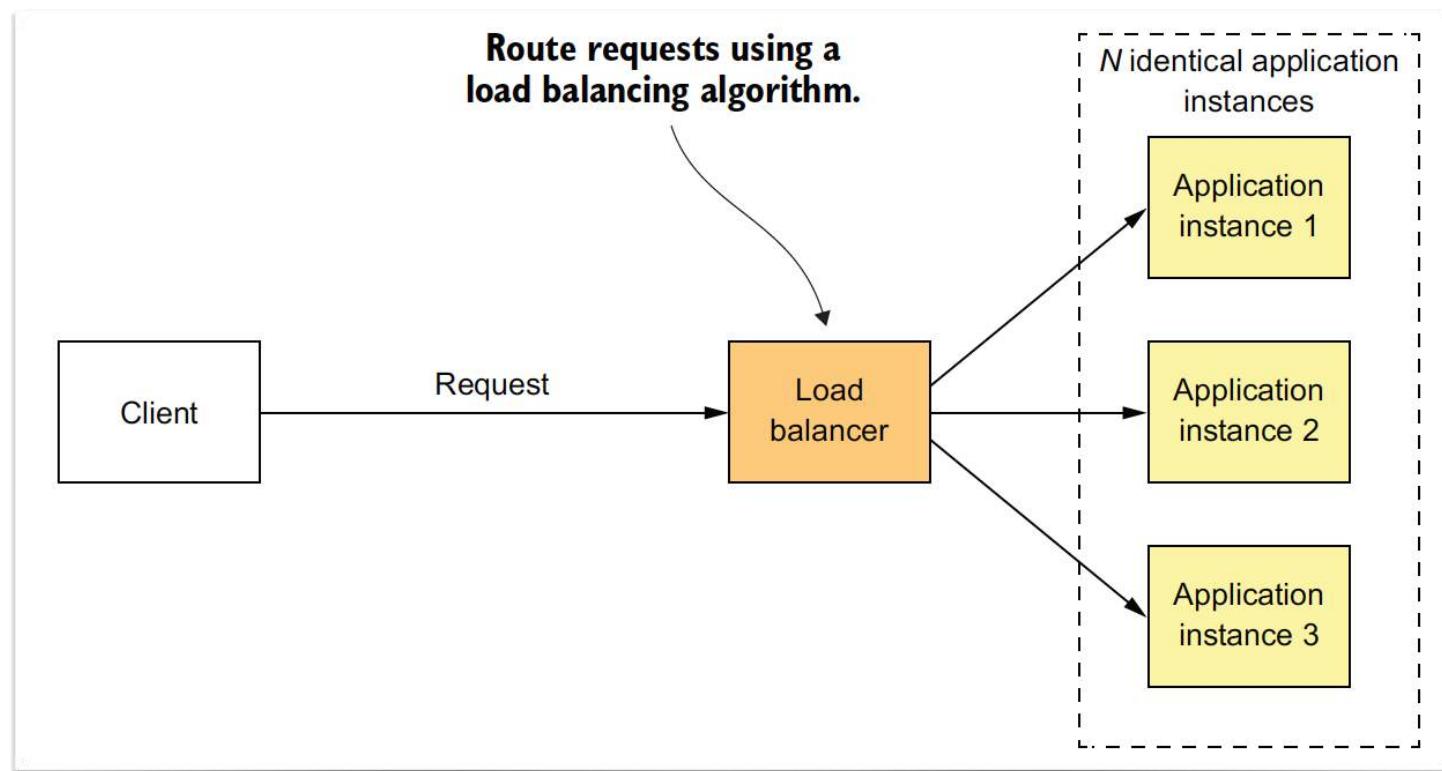
The scale cube defines **three** separate ways to **scale** an application:

- **X-axis scaling** load balances **requests** across multiple, identical instances (Scale by cloning)
- **Z-axis scaling** routes requests based on an attribute of the request (Scale by splitting similar things)
- **Y-axis functionally decomposes an application** into services (Scale by splitting things are different).



# X-Axis scaling

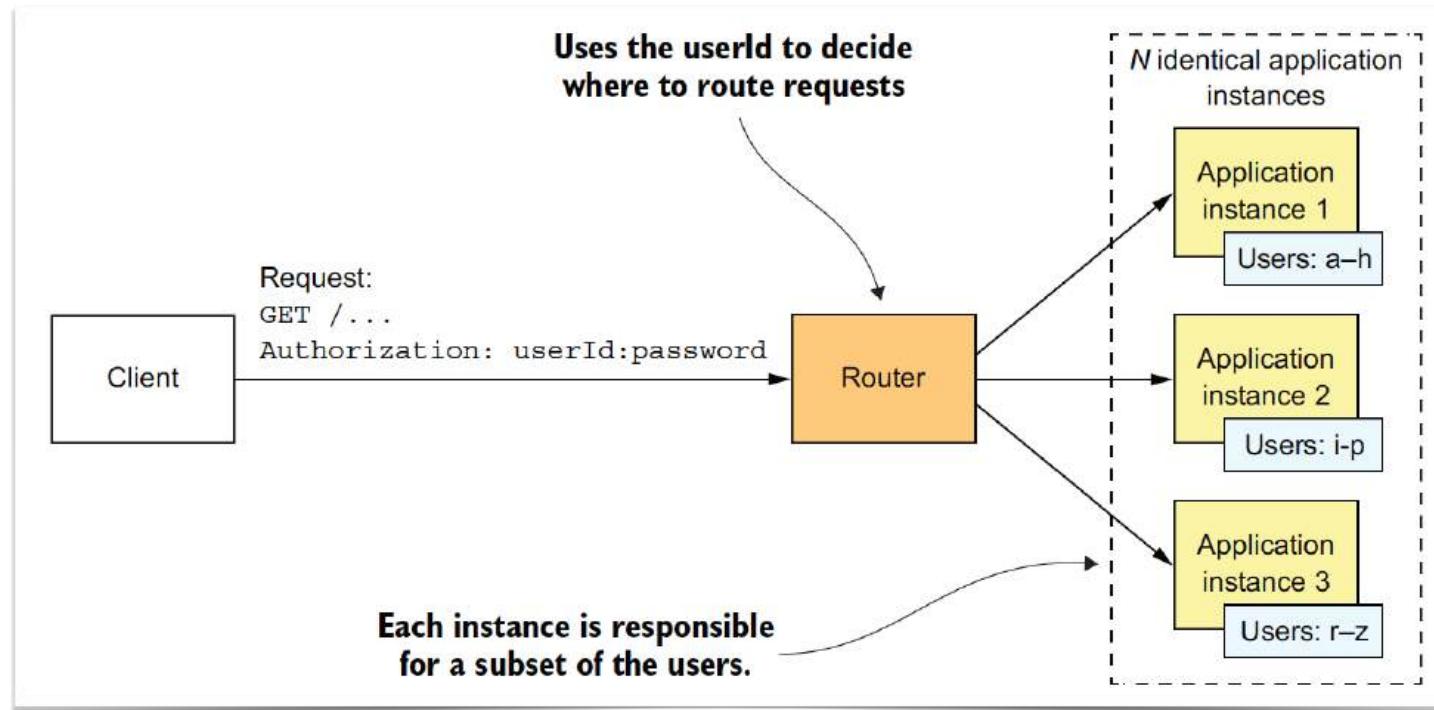
Multiple instances of the application run behind a load balancer. The load **balancer** distributes requests among the **N identical instances** of the application. This is a great way of **improving the capacity and availability** of an application.



# Z-axis scaling

**Z-axis scaling** also runs multiple instances of the monolith application, but unlike X-axis scaling, each instance is responsible for only a subset of the data.

The router in front of the instances uses a request attribute to route it to the appropriate instance.

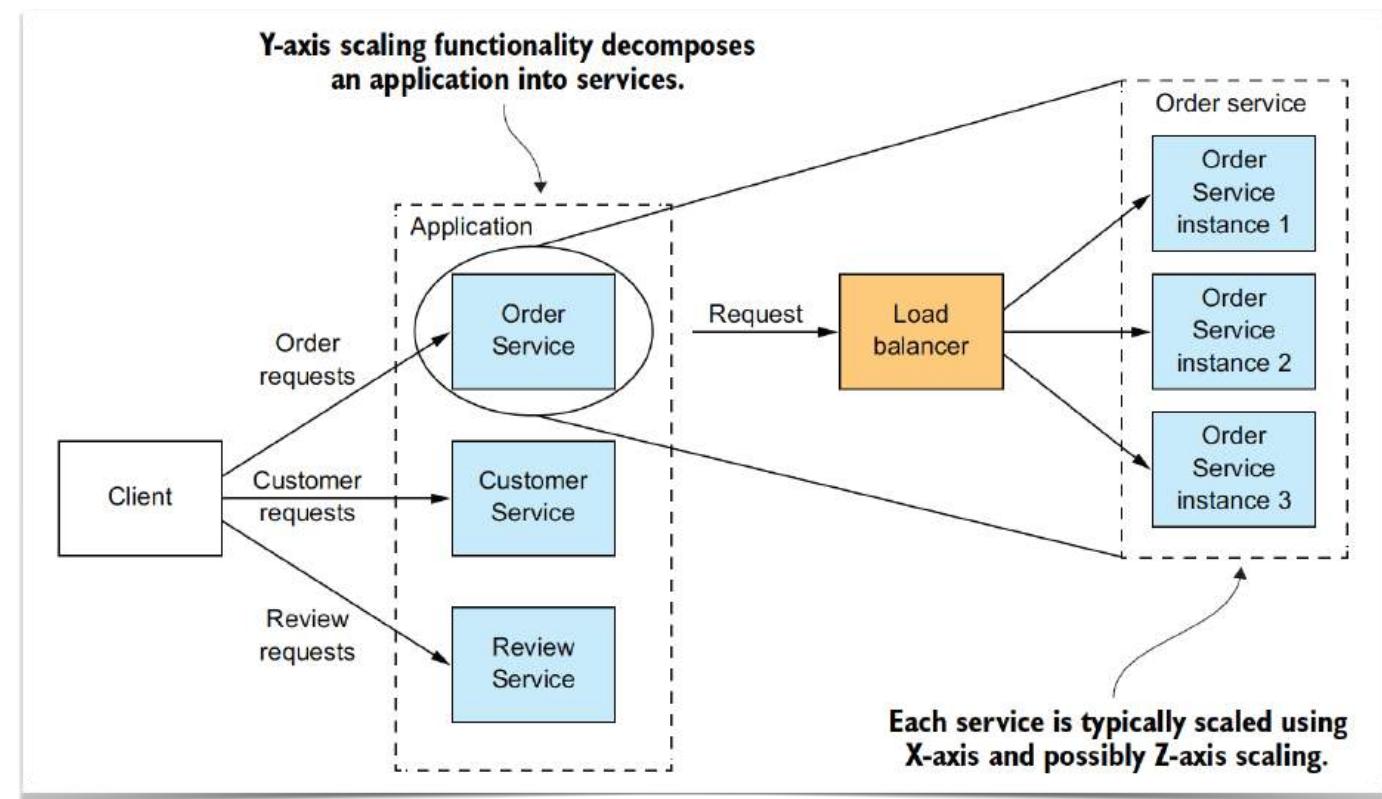


# Scaling Labracore

- How can we scale Labracore?
- Can we just duplicate the whole service?
  - X-axis scaling, yes
  - Z-axis scaling, yes
  - However, database becomes bottleneck unless also split!
  - How do we split our data?

# Y-axis scaling

- X- and Z-axis scaling
  - Improve **capacity** and **availability**
  - Neither solves the **problem of increasing development and application complexity.**
- Y-axis scaling (**functional decomposition**) does.



A **service** is a mini application that implements **narrowly focused functionality**, such as order management, customer management, etc.

# Scaling Labracore

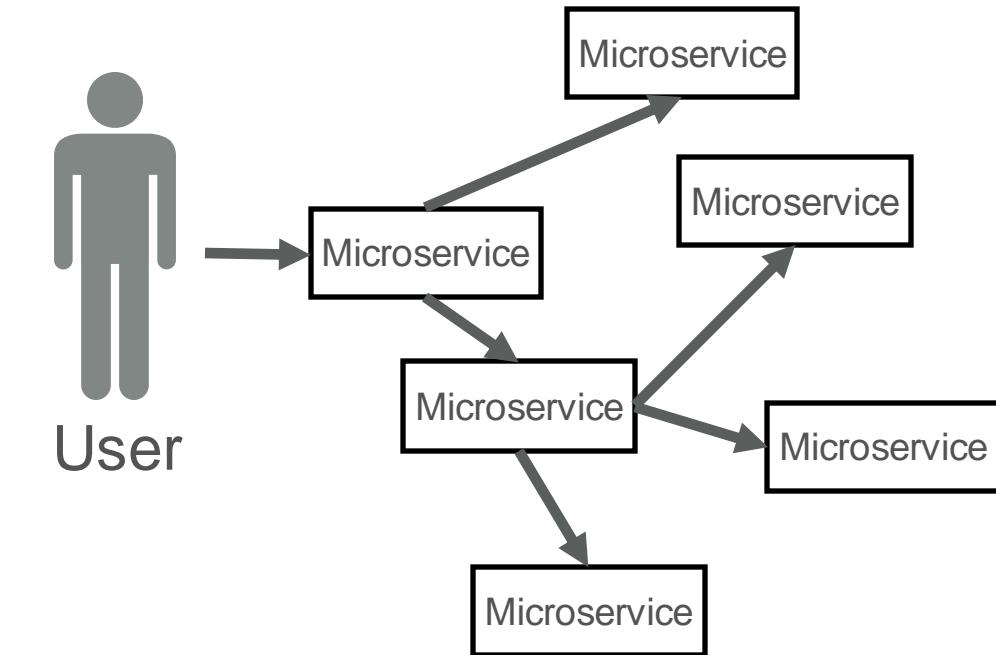
- Y-axis scaling allows us to also split the data!
- Implementing Y-axis scaling means we are switching to a **microservice architecture**
- Let's build Queue as a true microservice application!

# **Large Scale: Microservice Architecture**

# Microservices

Microservices re-arrange an application as collections of **loosely coupled services**.

Each microservice can be implemented in different programming languages, database, hardware and software environment, etc.



# Question from WebLab

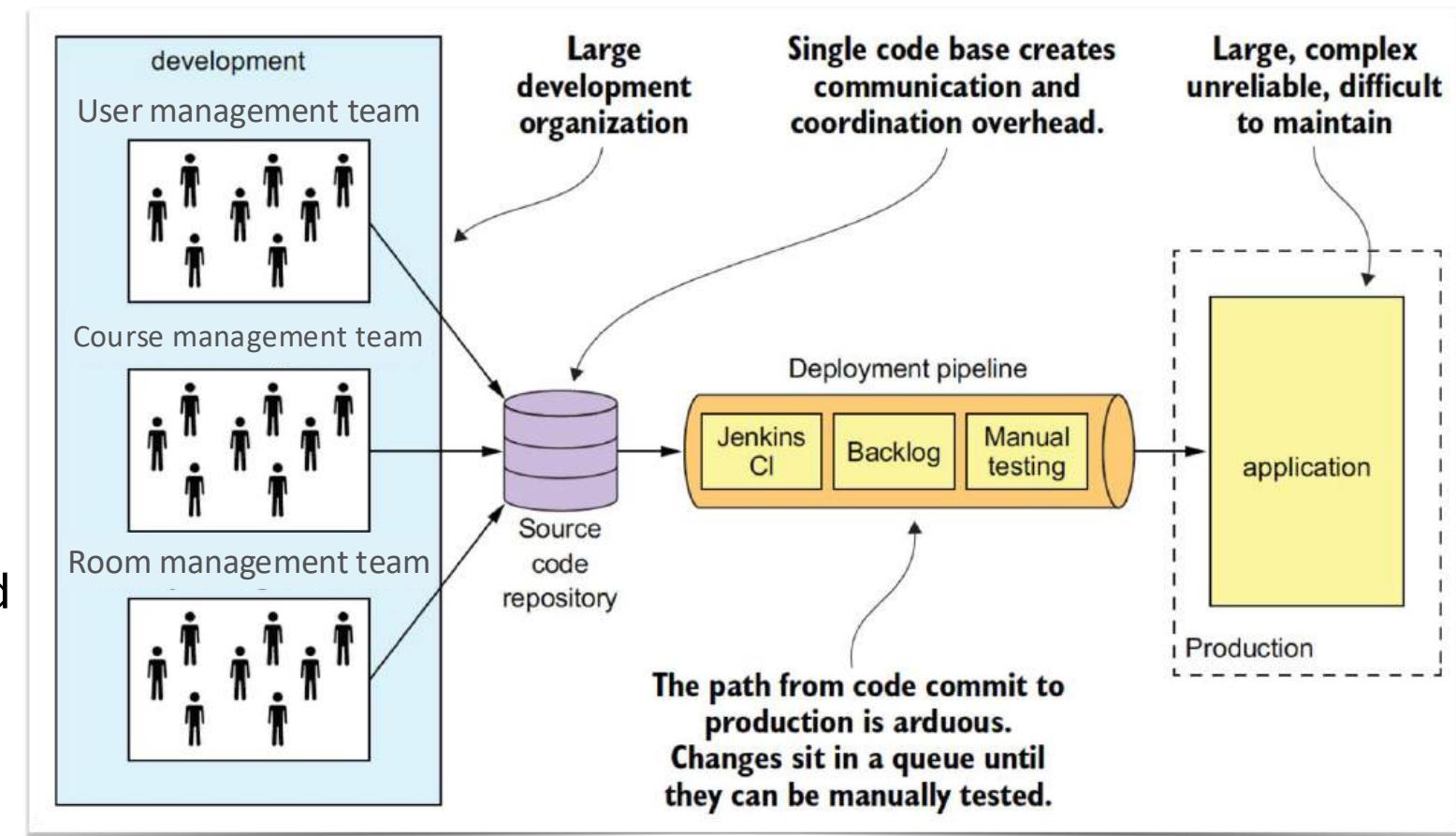
- “Isn’t a microservice layout just like any other system you build on your local computer, but then scattered across different computers?”
  - Yes
    - It’s like a normal application, but the parts are strictly separated (very well defined jobs)
  - No
    - Every microservice is a full application (webserver + database + api + ...)
    - Parts of your application communicate over the network instead of through e.g. local method calls

# What are the Limitations?

The large developer team commits their changes to a single source code repository.

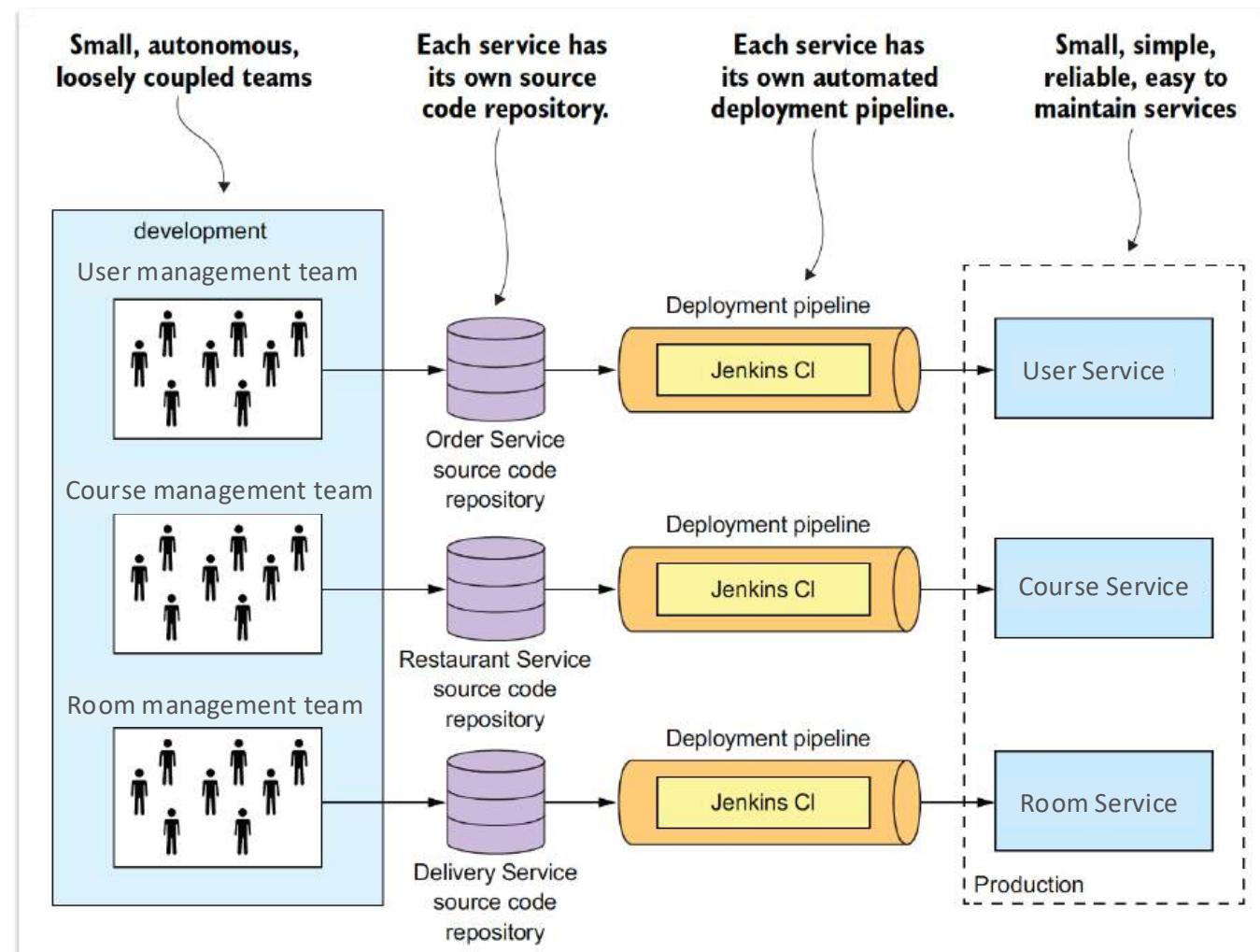
The path from code commit to production is long and arduous and involves manual testing.

The application is large, complex, unreliable, and difficult to maintain.

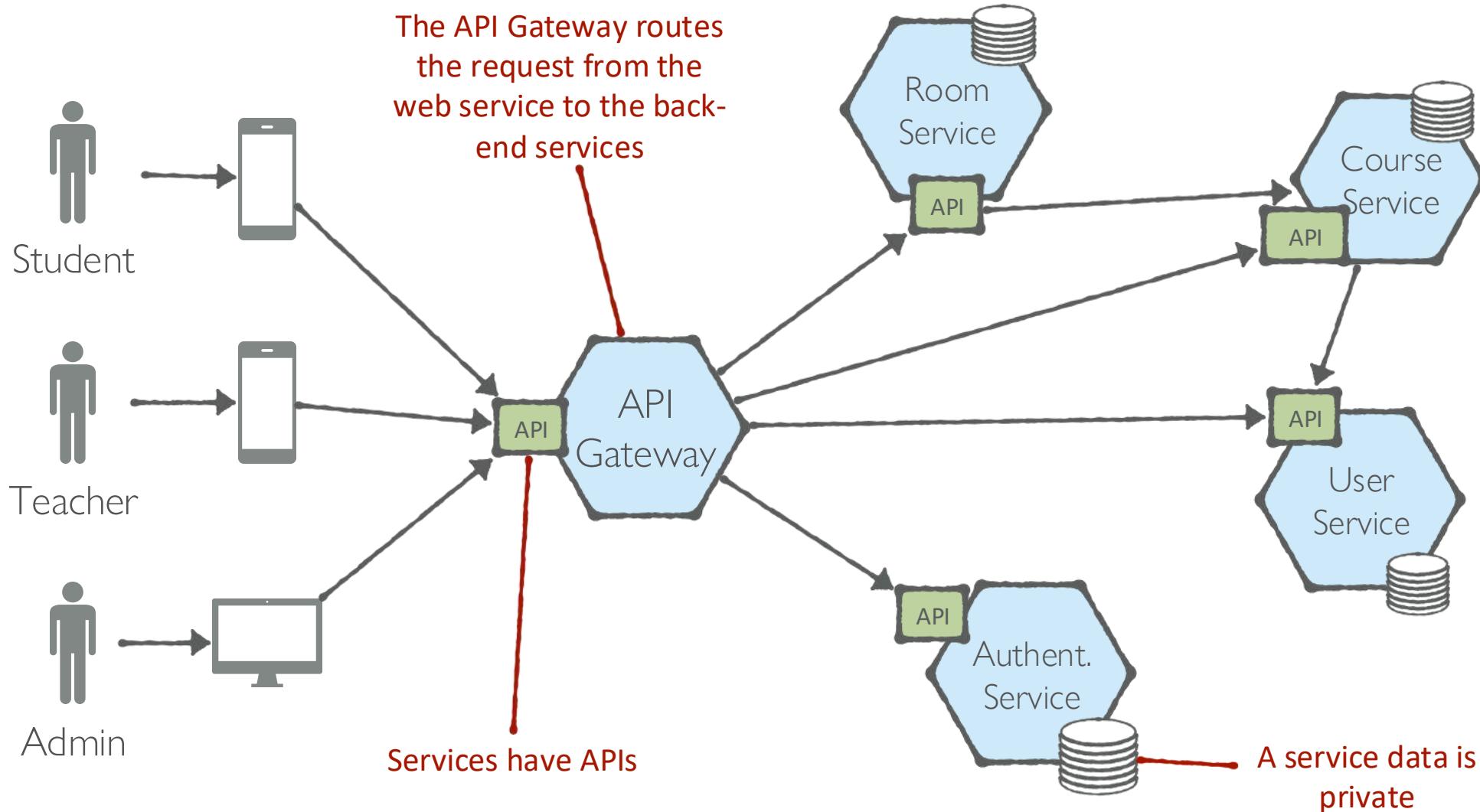


# Benefits of Microservices

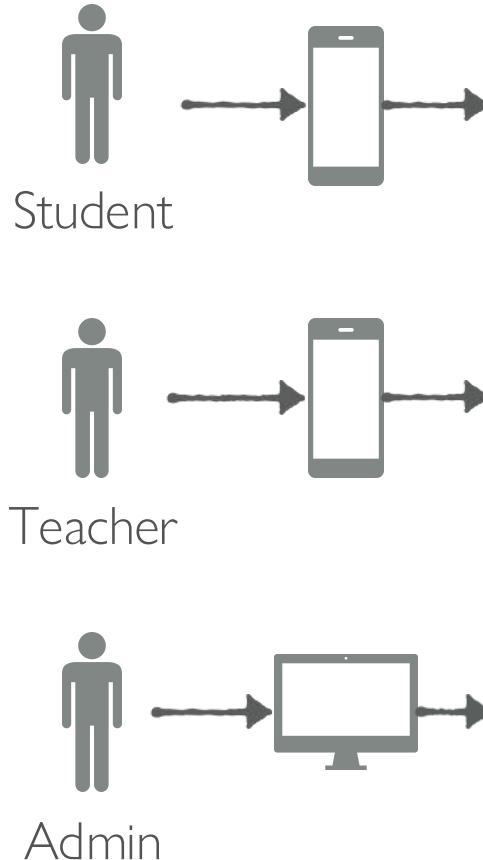
- Services are **small** and **easily maintained**.
- Services are **independently deployable**.
- Services are **independently scalable**.
- The microservice architecture enables teams to be **autonomous**.



# Course Management System: microservice architecture

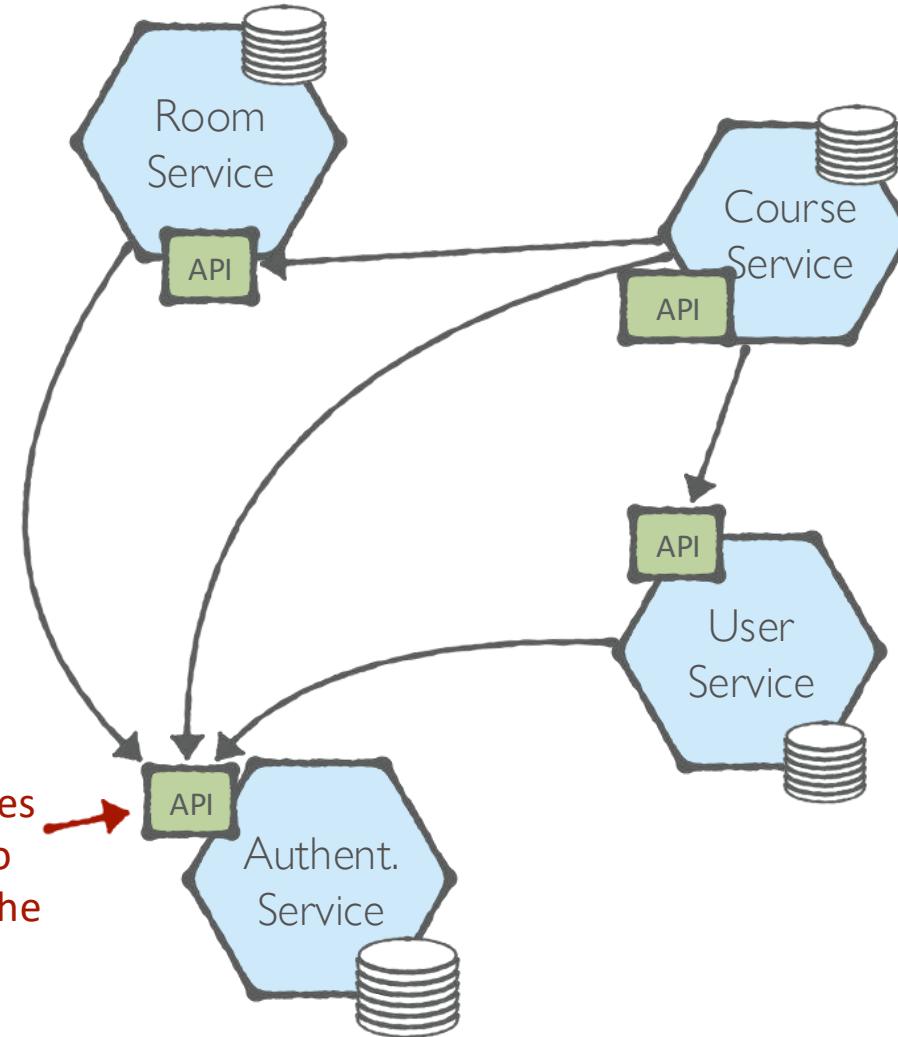


# Alternative microservice architecture



Users can send  
request to any  
microservice

All back-end microservices  
will require the users to  
“authenticate” through the  
authentication service



# Course Management System: microservice architecture

The Course Management System's business logic consists of numerous backend services. Each backend service has a REST API and its own private datastore. The backend services include the following:

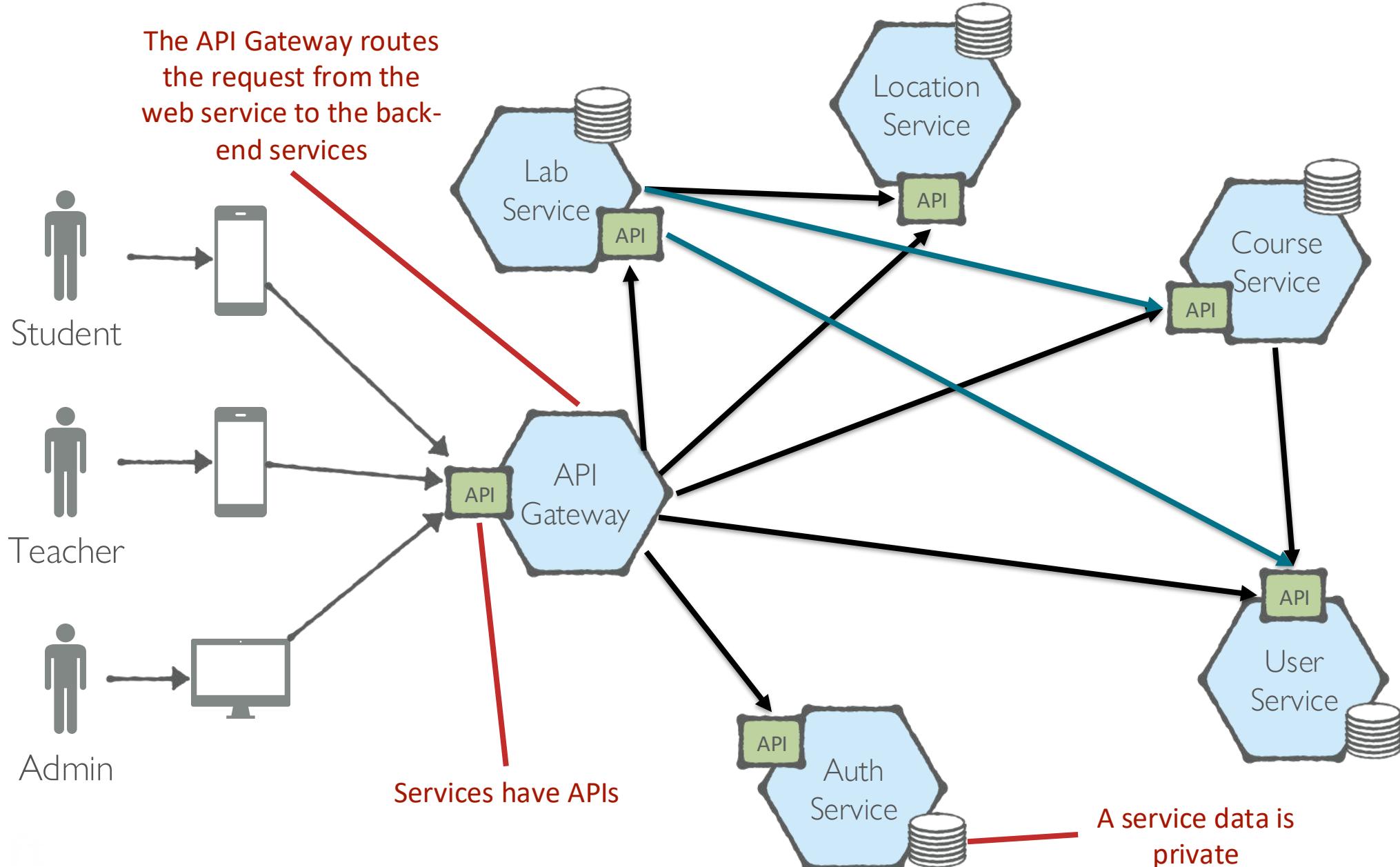
- **Room Service**: Maintain information about rooms
- **Course Service**: Manages courses, students enrollment, and course schedule
- **User Service**: Manage users (students and teachers) data
- **Authentication Service**: Manage the authentication and it is connected to the other TU Delft systems via Single-Sign-On

An API Gateway routes requests from the mobile applications to services. The services collaborate via APIs.

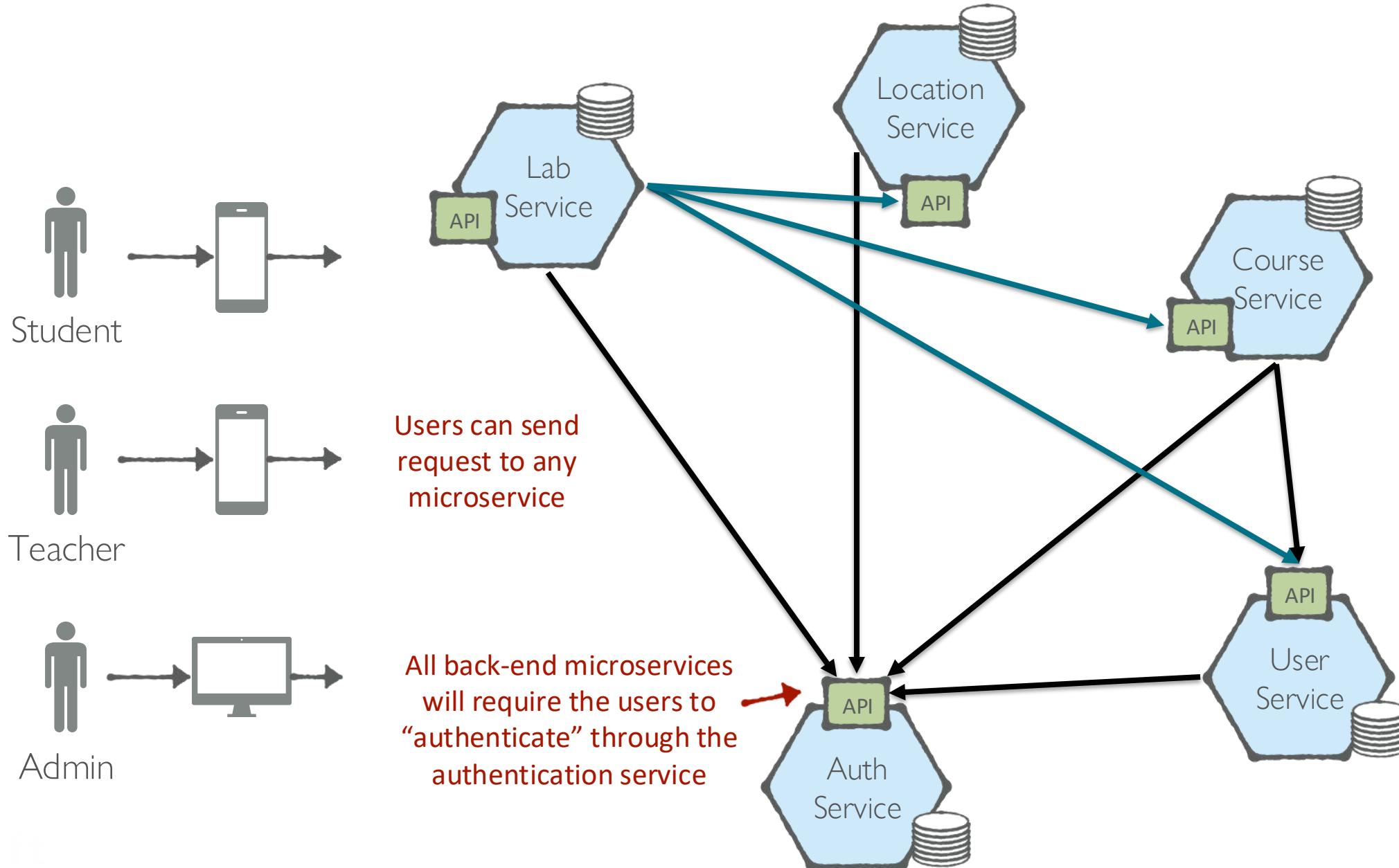
# Queue as Microservice Architecture

- Location Service
  - Information on rooms (and facilities)
  - Determine position of student in room
  - Route navigation for TA to student position
- Course Service
  - Staff (roles), Course information, Editions
  - Automatically update information from other systems (Study Guide, MyTimeTable)
- Authentication Service
  - Sign in through SSO
  - Permissions
- User Service
  - Basic information on all users
- (Lab) Session Service
  - Labs, enqueue/dequeue
  - Grab requests

# Queue as Microservice Architecture



# Alternative microservice architecture

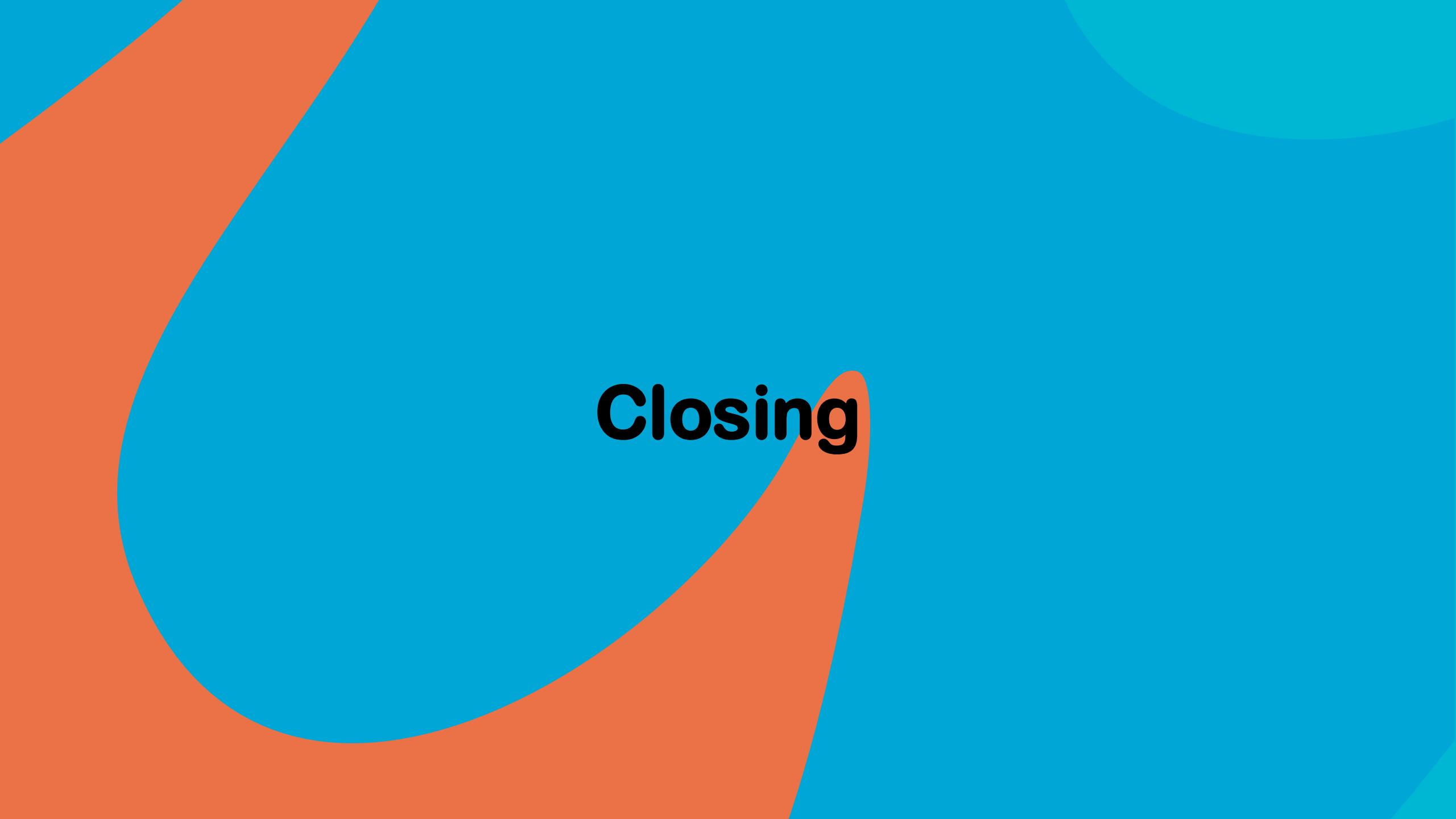


# Fault Tolerance

- Microservice Architecture can provide fault tolerance
  - Service down?
    - Go to one of the other identical ones
    - Spin up a new instance
  - Not all microservices need all other microservices to function
- Fallback functions
  - If a service doesn't respond, define a fallback
  - Allow large parts of the application to work with some parts broken.

# How do you test Fault Tolerance?

- Netflix: ChaosMonkey
  - Pick a random node in production, kill it
  - Does it still work?
  - Repeat
- Fault Injection
  - Block outgoing communication from Microservice

The background features a large, abstract shape composed of two main colors: orange and teal. The orange shape is located on the left side, starting from the top-left corner and curving down towards the bottom-right. The teal shape is on the right side, starting from the top-right corner and curving down towards the bottom-left. The two shapes overlap in the center.

# **Closing**

# Takeaways (Medium Scale)

- Even “simple” software gets really complex when it needs to be used in real-world settings
- Medium-Scale Software is already really hard to develop

*Note: EIP software is developed on a really tiny budget compared to usual software of its scale. Don't blame the developers!*

# Takeaways (Debugging)

- Analyzing issues of large-scale systems is hard.
- Reproducing production issues locally is a challenge. A lot of issues don't happen in development.
- Use application monitoring software to make your life easier.

# **CSE2115 Software Engineering Methods**

**Software at Scale (Part 2)**

**Taico Aerts & Arie van Deursen**

# Learning Objectives

- Use application monitoring software to analyse errors and performance of applications
- Know about different caching techniques to improve scalability
- Know different ways of scaling your application and infrastructure (Scale Cube)
- Convert existing systems to a microservice architecture
- **Know how to use Docker to create images as container blueprints**

# Outline

- Large-scale
  - Microservices
  - Deploying to the cloud
  - Serverless

# Last Time

# DDoSing Queue

- For every page load, queue used to do 20-40 requests to Labracore
  - DDoS amplification!
  - Labracore is hit ~30x as hard as Queue is



# Performance Metrics with Sentry



# How to reproduce the issue?

- Impersonation
  - Try to perform action as the user, more likely to run into the problem
- Test locally with production data
  - More representative of circumstances on production
  - Severe privacy concerns
  - Not feasible for large databases
- Extensive error reporting/tracking
  - E.g. Sentry

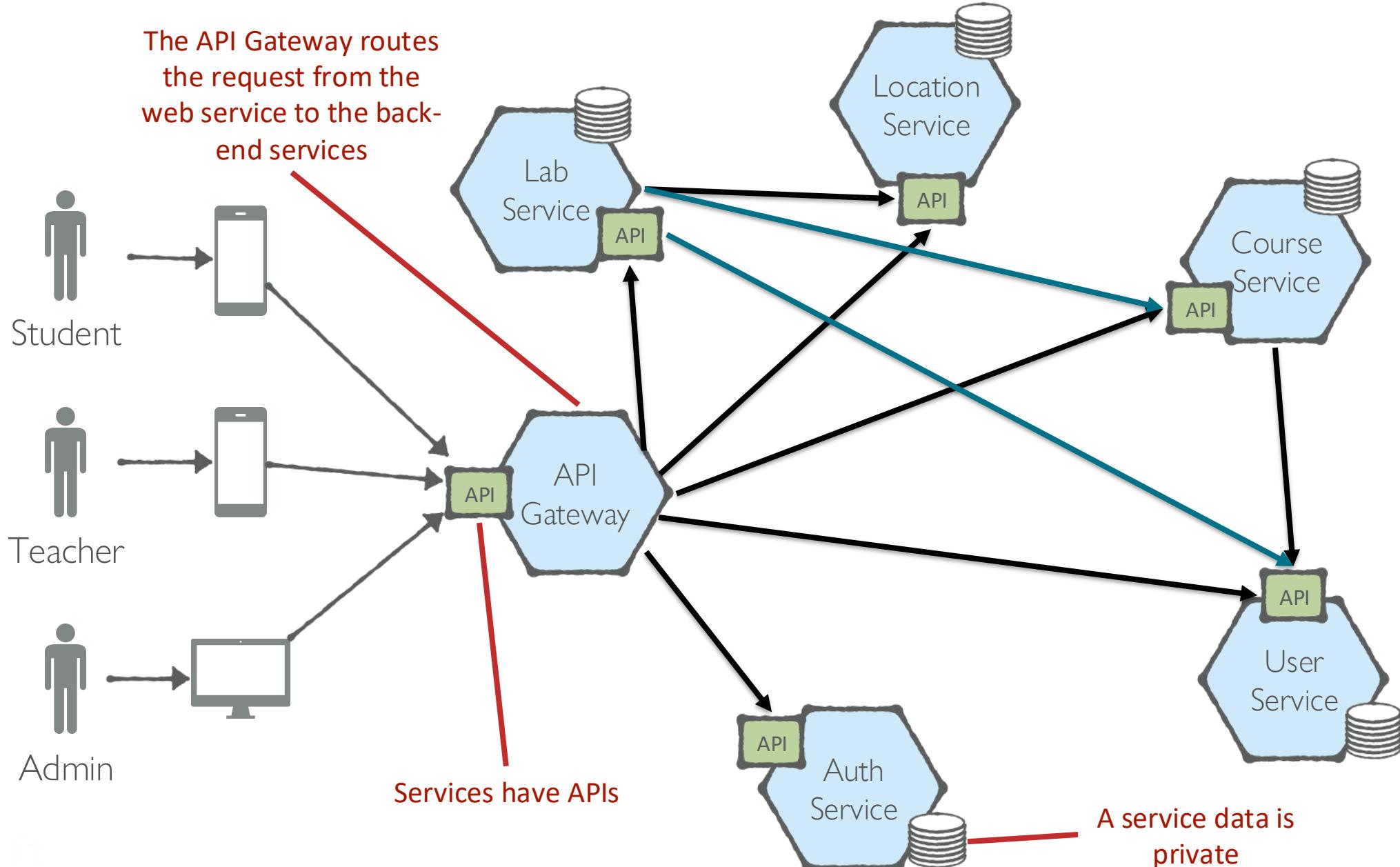
# Scaling Labracore

- Y-axis scaling allows us to also split the data!
- Implementing Y-axis scaling means we are switching to a **microservice architecture**
- Let's build Queue as a true microservice application!

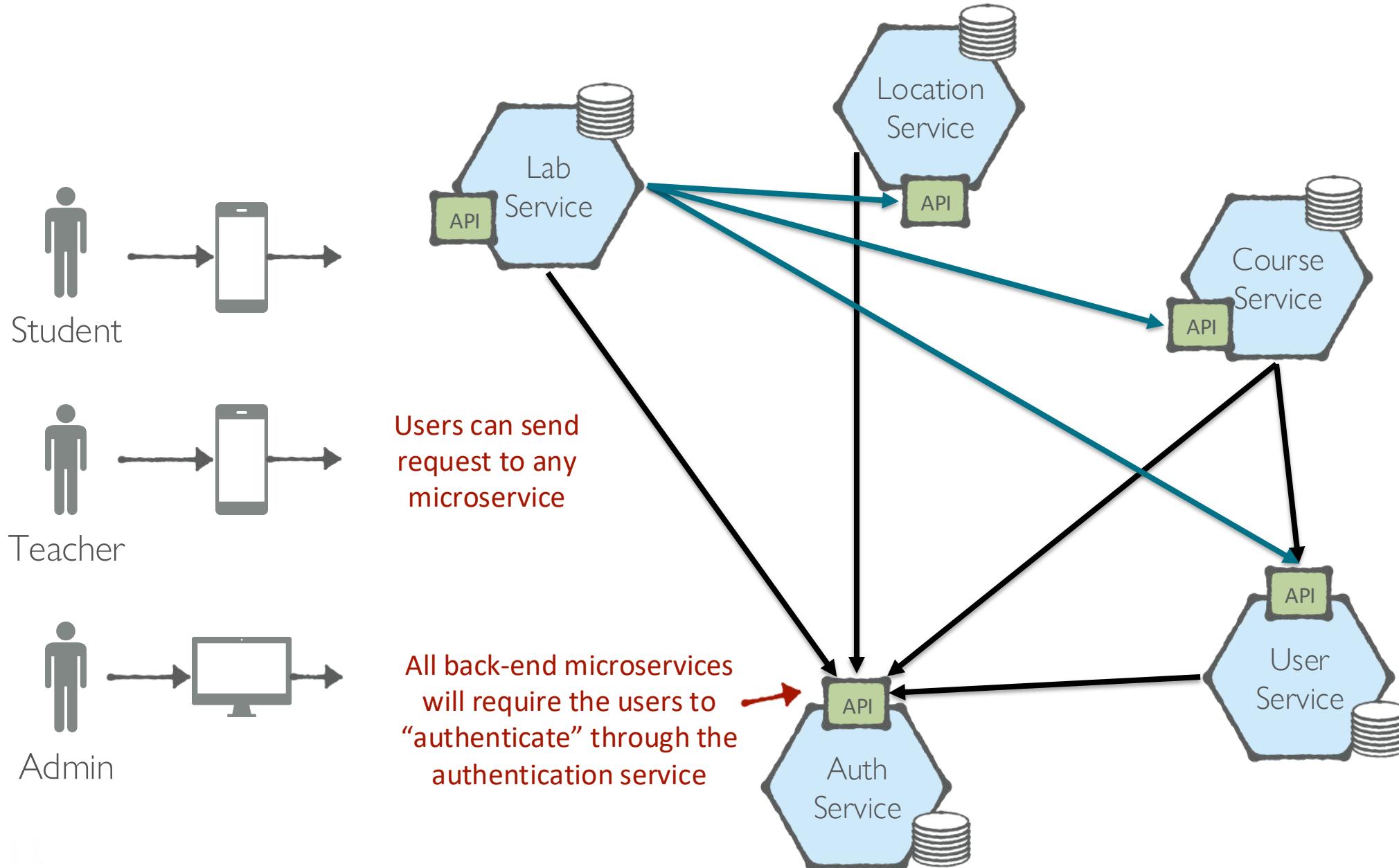
# Queue as Microservice Architecture

- Location Service
  - Information on rooms (and facilities)
  - Determine position of student in room
  - Route navigation for TA to student position
- Course Service
  - Staff (roles), Course information, Editions
  - Automatically update information from other systems (Study Guide, MyTimeTable)
- Authentication Service
  - Sign in through SSO
  - Permissions
- User Service
  - Basic information on all users
- (Lab) Session Service
  - Labs, enqueue/dequeue
  - Grab requests

# Queue as Microservice Architecture



# Alternative microservice architecture



# Fault Tolerance

# Fault Tolerance

- Microservice Architecture can provide fault tolerance
  - Service down?
    - Go to one of the other identical ones
    - Spin up a new instance
  - Not all microservices need all other microservices to function
- Fallback functions
  - If a service doesn't respond, define a fallback
  - Allow large parts of the application to work with some parts broken.

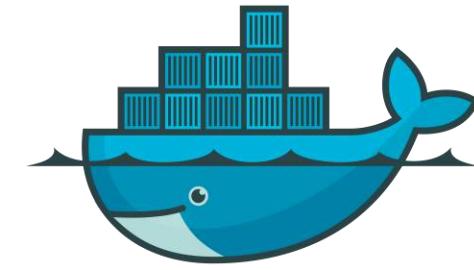
# How do you test Fault Tolerance?

- Netflix: ChaosMonkey
  - Pick a random node in production, kill it
  - Does it still work?
  - Repeat
- Fault Injection
  - Block outgoing communication from Microservice

# Deploying Large Scale Applications

# Docker

- Isolate your application/service
- Complete setup and configuration is captured
- You can reproduce the exact same application in the exact same environment across different machines!



docker



# Dockerfile

- Installation / Configuration instructions

```
FROM gradle:jdk21

RUN apt update && apt install mysql-client -y

EXPOSE 8081

RUN mkdir /var/www
RUN mkdir /var/www/queue

WORKDIR /var/www/queue

COPY build/libs/queue-*.jar queue.jar
COPY src/main/resources/application.docker.yml application.yml
COPY scripts/docker_entrypoint.sh docker_entrypoint

ENTRYPOINT ./docker_entrypoint
```

# Docker Images

- A dockerfile describes how to create a docker image
- The docker image is the **blueprint** for creating identical containers

# Example: Docker for WebLab

- Students provide code
- Code needs to be executed in some way
- Results need to be displayed by WebLab
- Ideally, this is efficient enough to do at a large scale with few resources.
- SEM: PlantUML WebLab Docker Image

# SEM WebLab Docker Image

```
FROM alpine/java:21-jdk

# Install packages
RUN apk add --no-cache bash graphviz wget

# Switch to Bash
SHELL ["/bin/bash", "-c"]

# Create student user
ARG UHOME=/home/student/
ENV HOME=/home/student
RUN set -o errexit -o nounset \
&& addgroup --gid 1000 "student" \
&& adduser --home "${UHOME}" --disabled-password --uid 1000 --ingroup "student" "student"
WORKDIR ${UHOME}

# Copy files into the container
COPY --chown=student home/ ${UHOME}
RUN ./download_runner.sh 0.2.10

# Switch to student user and create directories
USER student
RUN mkdir -p ${HOME}output/

# Run the runner
CMD [".run.sh"]
```

# (SEM PlantUML) WebLab Docker

- 1. Copy code files into container
- 2. Run predefined command (run.sh)
  - Write all output to files
- 3. Wait until done / kill after timeout
- 4. Copy output files
- 5. Show output on frontend

# Docker: Demo

# Example: Docker for GitLab

- Repositories with code
- Pipelines to test, lint, build, etc.
- **Every step in the pipeline runs in a docker container**
- Steps can define additional services: other docker containers that are accessible and run in parallel
  - E.g. a database service for your test step

# Example: Project Forum CI

Test:

```
stage: Test
```

```
image: $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
```

```
services:
```

```
- elasticsearch:7.17.1
```

```
- mariadb:10
```

```
variables:
```

```
RAILS_ENV: test
```

```
ELASTICSEARCH_URL: "https://elasticsearch:9200"
```

```
MYSQL_DATABASE: projectforum
```

```
MYSQL_ROOT_PASSWORD: root
```

```
script:
```

```
- bundle exec rails test
```

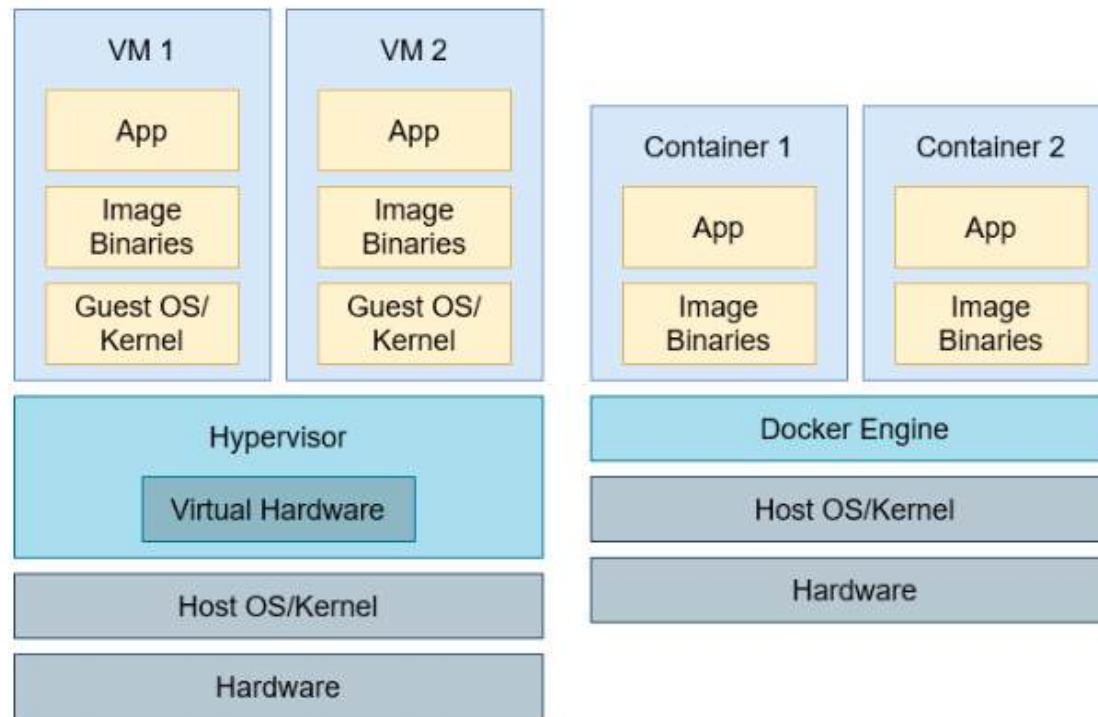
# Going Crazy with Docker

- DinD – Docker in Docker
- Actually has uses!
- Example: Project Forum GitLab CI step creating a docker image
  - Other steps use this image to avoid reinstalling necessary dependencies
- Remember, every gitlab job already runs in a docker container!



# Docker vs Virtual Machine

- VM emulates a whole machine + runs the whole client Operating System
- Docker does not run the whole OS



# Docker vs Virtual Machine

- Virtualization introduces performance overhead
- Containerization is more lightweight and efficient
- Any downsides to containerization?

# What is the Cloud?

- Kubernetes
  - Orchestrator for Docker Containers
  - Running on a cluster of systems
  - Potential to automatically scale
- Think GitLab
  - You have things that need to be run
  - It doesn't matter where it is run
  - You send the job to any machine in your cluster

*Containers, that'll fix it.*



Kubernetes  
for beginners.

*What could go wrong?*

O RLY?

*DevOps*

# What is the Cloud?



- Amazon Web Services (AWS)
- Google Cloud Platform
- Microsoft Azure
- Combination of various services
- Kubernetes/Docker is only one way of deploying applications

# Deploying in the Cloud

- Compute
  - Your actual service/application
  - Run of the mill (virtual) server
- Storage
  - Permanent/temporary storage for files
  - Replicated across regions
- Database
  - Optimized server running particular database software
  - Backups

# Deploying in the Cloud

- Cloud providers host containers and can perform autoscaling (elastic)
- Cloud providers encourage microservice-like architectures by separating all components

# **What is next?**

# Infrastructure as Code (IaC)

- Treat your infrastructure as code too
- Provision your cloud infrastructure (cloud or local) from the configuration
- Autoscaling (elastic)
- Terraform
  - Works with all cloud providers



# Infrastructure as Code (IaC)

- Many templates exist
- Integrate with your CI/CD to update services directly from your development platform (e.g. GitLab)



<https://memegenerator.net/X-All-The-Things>

# Serverless

- The next step after microservices:
- You scrap the services (!)
- You write single functions which can work without any application state
  - Every function is executed as if there is nothing else in your application
  - You can request information from your database
- They are executed “somewhere”

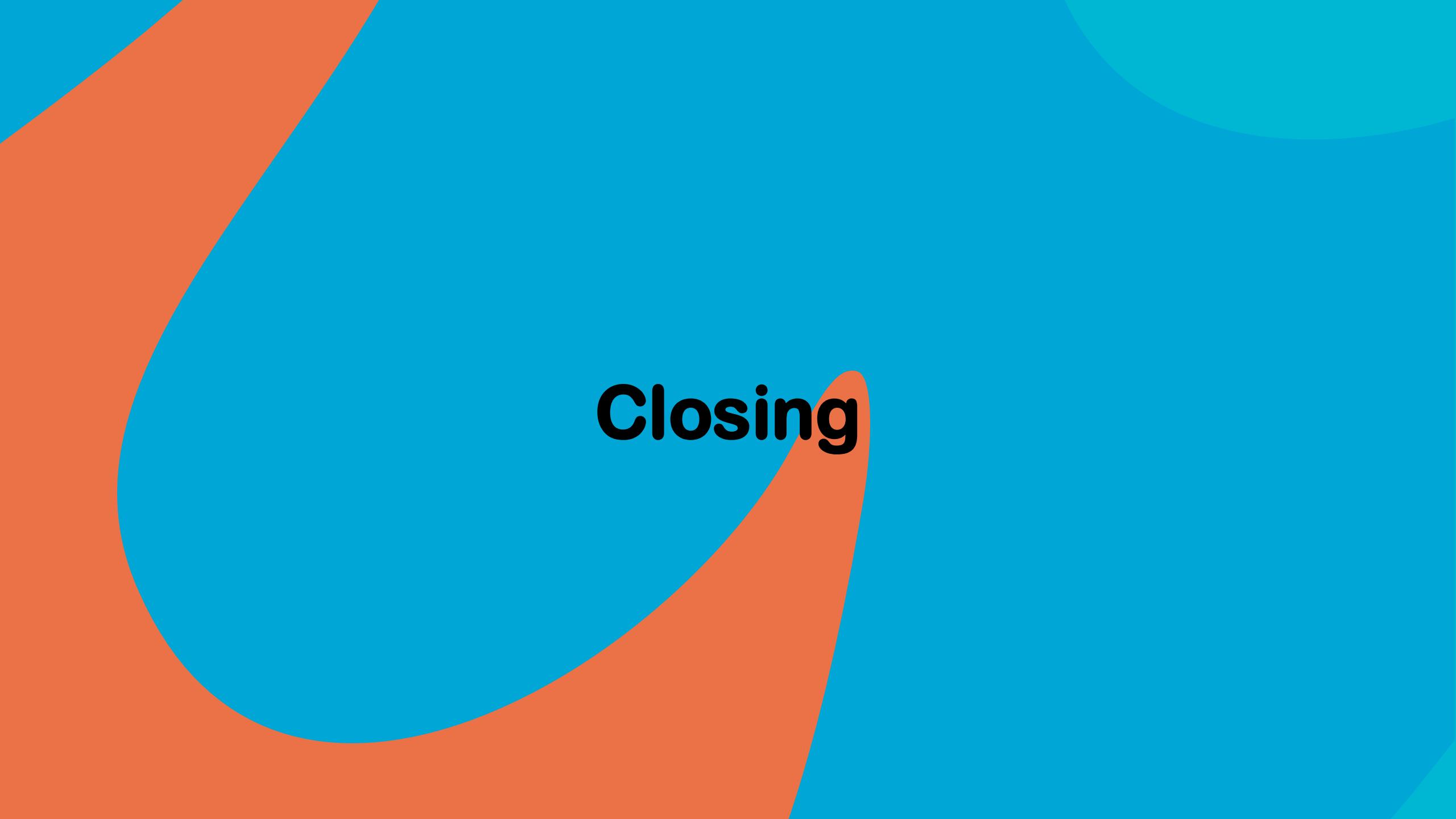
# Serverless

- Serverless still requires a server (!)
  - But you don't have to manage it
  - Which one doesn't matter (in theory)
- Advantages
  - Any machine can run any job (load distribution)
  - The pinnacle of isolated modules
- Disadvantages
  - No advantages from caching
  - Complexity can increase



# Serverless – Fun Fact

- Ethereum Smart Contracts are also serverless computations
- Miners execute contracts if paid the necessary money.
- Validators re-run the contract to verify the transaction was done correctly.
- Execution of contract (possibly) alters state of blockchain

The background features a large, abstract shape composed of two main colors: orange and teal. The orange shape is located on the left side, starting from the top-left corner and curving down towards the bottom-right. The teal shape is on the right side, starting from the top-right corner and curving down towards the bottom-left. The two shapes overlap in the center.

# **Closing**

# Takeaways (Medium Scale)

- Even “simple” software gets really complex when it needs to be used in real-world settings
- Medium-Scale Software is already really hard to develop

*Note: EIP software is developed on a really tiny budget compared to usual software of its scale. Don't blame the developers!*

# Takeaways (Scaling)

- Caching is essential
- Vertical scaling is finite, horizontal scaling is infinite
- Docker can be used to create identical setups across machines

# Takeaways (Debugging)

- Analyzing issues of large-scale systems is hard.
- Reproducing production issues locally is a challenge. A lot of issues don't happen in development.
- Use application monitoring software to make your life easier.

# Takeaways (Large Scale)

- Microservices organize your company, code and cloud.
- Manage infrastructure automatically on the cloud (with limits to protect your bank account) with Infrastructure as Code.

# Upcoming

- Next week: The future of Software Engineering

# ML4SE, Introduction

Egor Bogomolov  
Lead ML Researcher @ JetBrains Research



# About me

- Working as a lead ML researcher at JetBrains Research
- Team lead of the Code Modeling Research team at JetBrains
- Track lead at “LLM Adaptation for Coding Tasks” AI4SE track from the JetBrains side
- Pursuing PhD at TU Delft as a part of AI4SE

# Outline

- What is Machine Learning for Software Engineering (ML4SE)?
- Where does data come from?
- Is code different from natural language?
- How to measure quality?
- Which models do researchers use?

# What is ML4SE?

**ML4SE aims to help programmers in  
their work by leveraging ML methods**

# How much do developers *code*?

# Programming is not only coding

Activity Category		% of time over whole period	Duration per day (in hrs)			Time spent before switching (in mins)		
			Avg	Stdev	Max	Avg	Stdev	Max
Development Coding	reading/editing/navigating code (and other code related activities)	21.0%	1.5	±1.6	7.3	0.6	±2.6	135.7
Debugger Use	using the debugger inside the IDE	0.4%	0.1	±0.2	0.8	0.5	±0.8	13.4
Code Reviews	performing code reviews	1.3%	0.3	±0.4	2.1	1.3	±4.5	13.4
Version Control	reading/accepting/submitting changes	0.7%	0.1	±0.3	2.2	0.6	±1.0	12.9
Email	reading/writing emails	14.5%	1.1	±1.3	8.1	0.9	±4.8	89.6
Planning	editing work items/tasks/todos; creating/changing calendar entries	4.8%	0.5	±1.1	5.1	1.1	±2.5	67.5
Read/write documents	reading/editing documents and other artifacts, e.g., pictures	6.6%	0.5	±0.7	4.5	0.8	±3.3	114.7
Planned meeting	scheduled meeting/call	6.5%	1.0	±1.3	7.1	15.8	±35.3	203.1
Informal meeting	ad-hoc, informal communication; e.g., unscheduled phone call / IM, or colleague asks a question	3.4%	0.5	±0.6	4.2	2.0	±6.5	138.2
Work related browsing	Internet browsing related to code/work/task	11.4%	0.8	±1.3	12.8	0.5	±5.5	102.6
Work unrelated browsing	Internet browsing work unrelated	5.9%	0.5	±0.7	3.4	1.1	±4.3	91.8
Other	Anything else; aggregates several small sub-categories, such as changing music, updating software, using the file explorer or having a break	11.4%	0.8	±1.4	10.5	0.4	±5.6	112.5
Other RDP	Remotedesktop use which could not be mapped to another category	12.0%	1.5	±1.8	8.2	0.3	±2.6	85.4

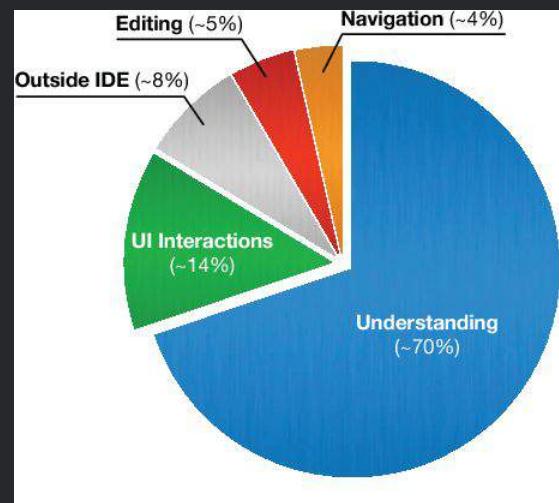
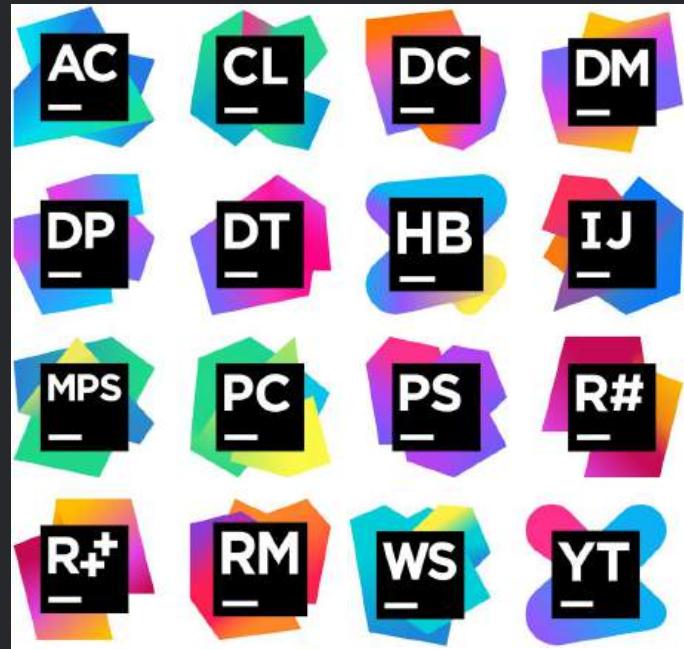


Fig. 3. How do developers spend their time?

# Programmers use a lot of tools

- IDEs
- CI/CD tooling
- Issue trackers
- Communication
- Design / prototyping



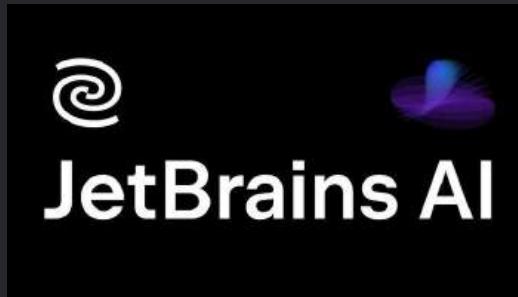
# Programming activities

- Writing code
- Writing documentation
- Editing existing code
- Testing
- Debugging
- Code comprehension
- Navigation and search
- ...

# ML4SE tasks

- Code generation / completion
- Code summarization / comprehension
- Issue deduplication
- Code search
- Program repair
- Test automation
- Code review
- Bug report analysis
- ...

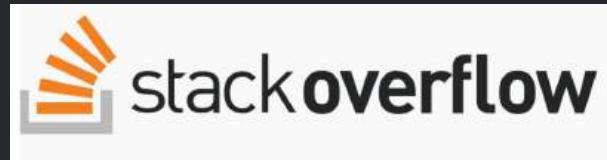
# The future is already here



Which data does  
ML4SE use?

# Where does data come from?

- Mining open-source repositories
  - GitHub, GitLab, ...
- Datasets based on them
  - The Stack, the Vault, ...
- Programming-related discussions
  - StackOverflow, GH discussions
- Issue trackers, code review tools, ...



# Data can be very different



## Contents

- Files with code
- Different languages

## Usage

- Code completion
- Documentation generation
- Method name prediction

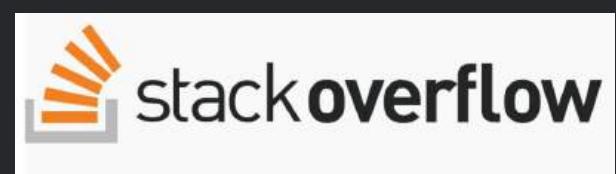


## Contents

- Merge requests
- Review comments
- Code edits

## Usage

- Code editing
- Bug repair
- Automated code review



## Contents

- Programming-related questions
- Answers to questions
- Code snippets

## Usage

- Question answering
- Code generation
- Instruction-tuning

# Should we treat code as a plain text?

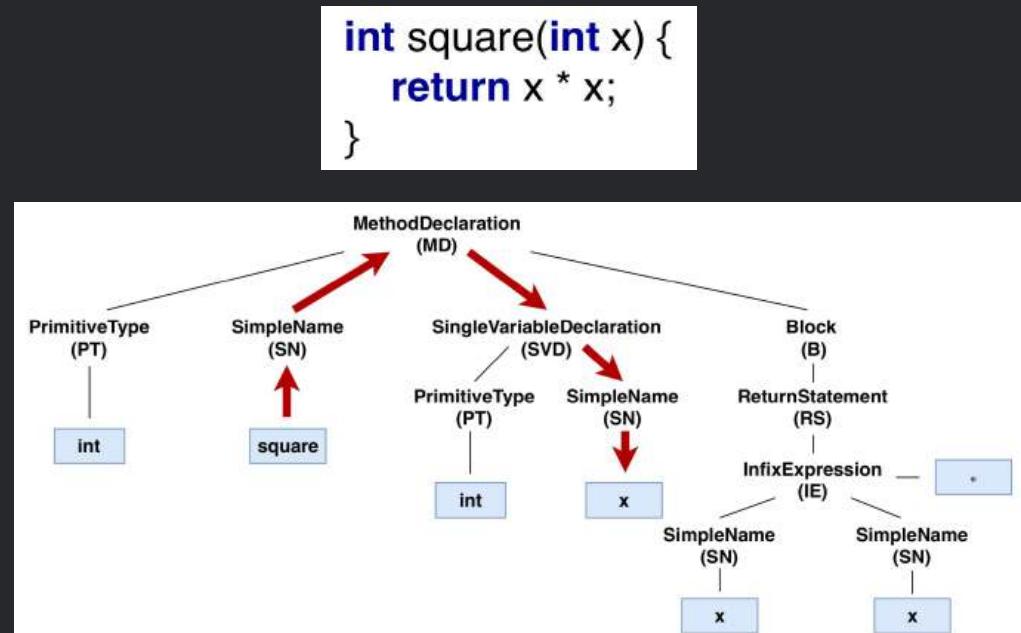
# Is code different from natural language?

- Looks like plain text

```
int square(int x) {  
    return x * x;  
}
```

# Is code different from natural language?

- Looks like plain text
- More structured – syntax tree
- Control- and Data-flow graphs
- Cross-file references
- Code can be executed!
- Development history (Git)
- Lots of related artifacts



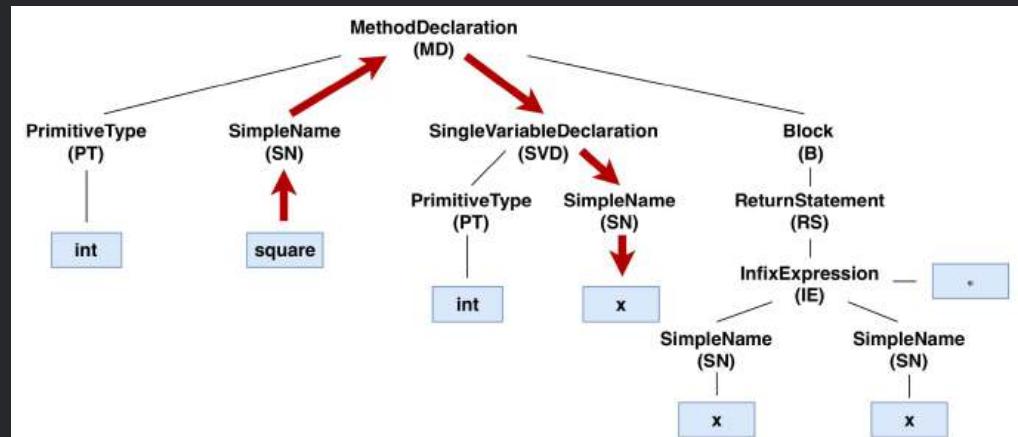
# Should we treat code as a plain text?

# Implications for model design

Architectures:

- Transformers, RNNs, ...
- Tree-based models
- Graph Neural Networks

```
int square(int x) {  
    return x * x;  
}
```



**We can even go multimodal!**

# ML4SE models

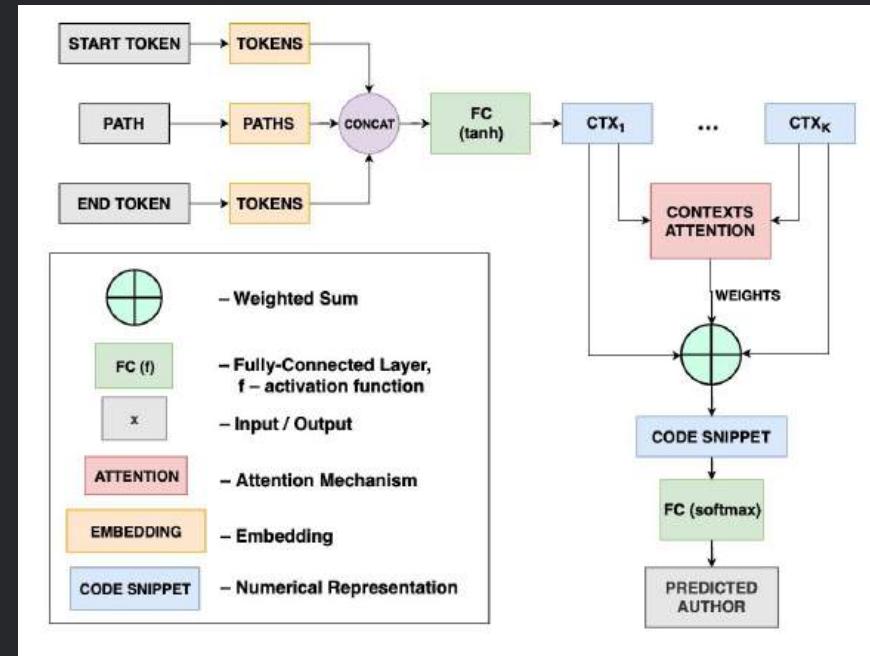
# A word about evaluation: HumanEval

- 164 algorithmic problems
- Test-based evaluation
- Python, then translated
- Probability of pass@K

EvalPlus Tests		
#	Model	pass@1
1	GPT-4-Turbo (April 2024) ✨	⚡ 86.6
2	DeepSeek-Coder-V2-Instruct ✨	⚡ 82.3
3	GPT-4-Turbo (Nov 2023) ✨	⚡ 81.7
4	GPT-4 (May 2023) ✨	⚡ 79.3
5	CodeQwen1.5-7B-Chat ✨	⚡ 78.7
6	claude-3-opus (Mar 2024) ✨	⚡ 77.4
7	DeepSeek-Coder-33B-instruct ✨	⚡ 75
8	OpenCodeInterpreter-DS-33B ✨ ❤️	⚡ 73.8
9	WizardCoder-33B-V1.1 ✨	⚡ 73.2
10	Artigenz-Coder-DS-6.7B ✨	⚡ 72.6

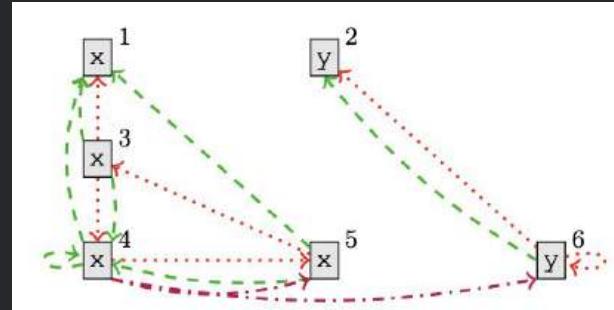
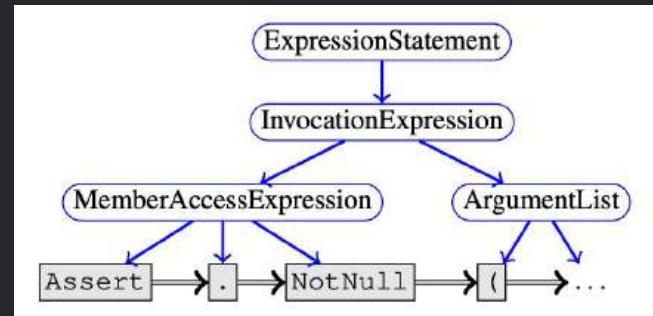
# Early code-specific architectures

- Explicitly leveraging code structure
- Require languages-specific code processing
- A wide variety of models:
  - Path-based architectures (code2seq)
  - Tree-based convolutions
  - TreeLSTMs
  - ...



# Usage of Graph Neural Networks

- Graphs are built as AST+CFG+DFG+PDG+...
- Good quality due to rich data representation
- Hard to mine large datasets and scale models
- Not suitable for code modeling
- Graph neural networks architectures
  - GCN
  - GGNN
  - GAT
  - ...



# Usage of graph data in Transformers

- Biases in self-attention based on edges
- Novel code-oriented tasks for pre-training
- “GNN-sandwiches”

## Models:

- CodeBERT
- GraphCodeBERT
- CodeT5
- UniXCoder

**Instruction:** Create a SQL query to get the list of employee names and ids with a monthly income greater than 4,000.

**Input:** n/a

**Output:**

```
SELECT id, name FROM Employees WHERE  
monthly_income > 4000;
```

**Instruction:** Write a code to add two numbers without using the "+" operator.

**Input:**

```
num1 = 2  
num2 = 7
```

**Output:**

```
num1 = 2  
num2 = 7  
# Without using "+" operator  
sum = num1 - (-num2)
```

**Instruction:** Optimize the given Python program to improve the speed of execution.

**Input:**

```
def calc( num1, num2):  
    result = 0  
    for i in range(num1, num2):  
        result += ((i+1) * (i+2))  
    return result
```

**Output:**

```
def calc(num1, num2):  
    result = (num1 + num2) * (num2  
- num1 + 1) // 2  
    return result
```

**Yet, data is too big of an advantage**

# Fine-tuning natural language LLMs

Models families fine-tuned on code data:

- StarCoder, StarCoder 2
- CodeLLaMA
- Refact-1.6
- DeepSeek-Coder, DeepSeek-Coder-v2
- Codestral
- Granite
- CodeQwen
- ...

Pre-training on  
NL and Code mix

Further tuning on  
code

Instruction tuning,  
DPO, PPO, ...

# Fine-tuning natural language LLMs

Models families fine-tuned on code data:

- StarCoder, StarCoder 2
- CodeLLaMA
- Refact-1.6
- DeepSeek-Coder, DeepSeek-Coder-v2
- Codestral
- Granite
- CodeQwen
- ...

EvalPlus Tests		
#	Model	pass@1
1	GPT-4-Turbo (April 2024) ✨	⚡ 86.6
2	DeepSeek-Coder-V2-Instruct ✨	⚡ 82.3
3	GPT-4-Turbo (Nov 2023) ✨	⚡ 81.7
4	GPT-4 (May 2023) ✨	⚡ 79.3
5	CodeQwen1.5-7B-Chat ✨	⚡ 78.7
6	claude-3-opus (Mar 2024) ✨	⚡ 77.4
7	DeepSeek-Coder-33B-instruct ✨	⚡ 75
8	OpenCodeInterpreter-DS-33B ✨ ❤️	⚡ 73.8
9	WizardCoder-33B-V1.1 ✨	⚡ 73.2
10	Artigenz-Coder-DS-6.7B ✨	⚡ 72.6

# Fine-tuning natural language LLMs

Models families fine-tuned on code data:

- StarCoder, StarCoder 2
- CodeLLaMA
- Refact-1.6
- DeepSeek-Coder, DeepSeek-Coder-v2
- Codestral
- Granite
- CodeQwen
- ...

Varying model size, context size, set of supported languages, tasks, performed tuning...

EvalPlus Tests		
#	Model	pass@1
1	GPT-4-Turbo (April 2024) ✨	⚡ 86.6
2	DeepSeek-Coder-V2-Instruct ✨	⚡ 82.3
3	GPT-4-Turbo (Nov 2023) ✨	⚡ 81.7
4	GPT-4 (May 2023) ✨	⚡ 79.3
5	CodeQwen1.5-7B-Chat ✨	⚡ 78.7
6	claude-3-opus (Mar 2024) ✨	⚡ 77.4
7	DeepSeek-Coder-33B-instruct ✨	⚡ 75
8	OpenCodeInterpreter-DS-33B ✨ ❤️	⚡ 73.8
9	WizardCoder-33B-V1.1 ✨	⚡ 73.2
10	Artigenz-Coder-DS-6.7B ✨	⚡ 72.6

# Challenges of going into production

- Take a big model (Llama-405B)
- Build a feature
- To the moon

# Challenges of going into production

- Take a ~~big model (Llama 40B)~~ - if latency is important, bigger is not always better!
- Build a feature
- To the moon

# Challenges of going into production

- ~~Take a big model (Llama 40B)~~ – if latency is important, bigger is not always better!
- ~~Build a feature~~ – UX and evaluation are crucial for a realistic setup
- To the moon

# Challenges of going into production

- ~~Take a big model (Llama 405B)~~ – if latency is important, bigger is not always better!
- ~~Build a feature~~ – UX and evaluation are crucial for a realistic setup
- ~~To the moon~~ – models experience degradation, you have to iterate on feedback

# That is it! Let's summarize

- What is Machine Learning for Software Engineering (ML4SE)?
  - ML methods aiming to help programmers in their day-to-day tasks
- Where does data come from?
  - Open-source repositories, code review systems, issue trackers, git history, ...
- Is code different from natural language?
  - It has strict structure, lots of internal dependencies and connection, it can be executed
- How to measure quality? – Use metrics or run tests
- Which models do researchers use? – Everything, from NLP models to Graph Neural Networks



# Generative AI in Development

Maliheh Izadi, PhD  
Software Engineering Methods course  
October 2024

# Introduction

Maliheh Izadi, PhD



Assistant Professor,  
Software Engineering Research Group

I work at the intersection of machine learning and software engineering to improve **software development tools** and **developer productivity**.

AISE research lab director [TU Delft]

AI4SE lab – scientific manager [TU Delft+JetBrains Research]

# **“Software is Eating the World”**

Marc Andreessen, Wall Street Journal, 2011

Software has been transforming the world, touching every aspect of our lives.

From smartphones to smart homes, software is the backbone of modern technology, driving innovation and efficiency.



# The Role of Developers in Software Creation



Developers have traditionally been the architects of software, responsible for:

- **Writing code:** Writing code involves creating detailed instructions (written in a programming language) that a computer can follow to perform specific tasks.

```
1 def factorial(n):  
2     if n == 0:  
3         return 1  
4     else:  
5         return n * factorial(n - 1)  
6  
7 # Example usage  
8 print(factorial(5)) # Output: 120
```

Writing code

# The Role of Developers in Software Creation



Developers have traditionally been the architects of software, responsible for:

- **Writing code:** Writing code involves creating detailed instructions (written in a programming language) that a computer can follow to perform specific tasks.
- **Debugging:** Debugging is the process of identifying and resolving errors or "bugs" in the code that cause software to behave unexpectedly or fail.

```
1 def factorial(n):  
2     if n == 0:  
3         return 1  
4     else:  
5         return n * factorial(n - 1)  
6  
7 # Example usage  
8 print(factorial(5)) # Output: 120
```

Writing code

```
1 def factorial(n):  
2     if n == 0:  
3         return 1  
4     else:  
5         return n + factorial(n - 1)  
6  
7 # BUG: Should use * instead of +  
8  
9 # Example usage  
10 print(factorial(5))  
11 # Wrong Output: 16 (should be 120)
```

Bug fixing

# The Role of Developers in Software Creation



Developers have traditionally been the architects of software, responsible for:

- **Writing code:** Writing code involves creating detailed instructions (written in a programming language) that a computer can follow to perform specific tasks.
- **Debugging:** Debugging is the process of identifying and resolving errors or "bugs" in the code that cause software to behave unexpectedly or fail.
- **Maintaining software:** Software maintenance refers to the ongoing process of updating and improving software after its initial release. This includes fixing bugs, adding new features, and ensuring compatibility with new systems or environments.

```
1 def factorial(n):
2     if n == 0:
3         return 1
4     else:
5         return n * factorial(n - 1)
6
7 # Example usage
8 print(factorial(5)) # Output: 120
```

Writing code

```
1 def factorial(n):
2     if n == 0:
3         return 1
4     else:
5         return n + factorial(n - 1)
6
7 # BUG: Should use * instead of +
8
9 # Example usage
10 print(factorial(5))
11 # Wrong Output: 16 (should be 120)
```

Bug fixing

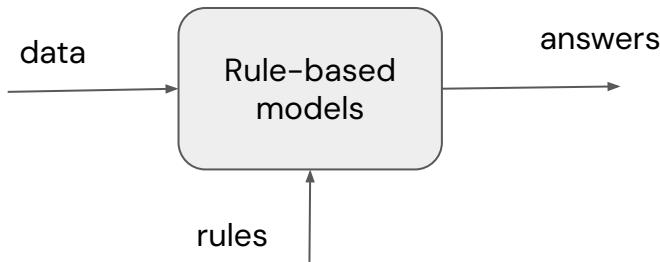
```
1 def factorial(n):
2     result = 1
3     for i in range(2, n + 1):
4         result *= i
5     return result
6
7 # Example usage
8 # Output: A very large number
9 print(factorial(100))
```

Maintenance

# The Shift!

## Explicit Rules in Software Development

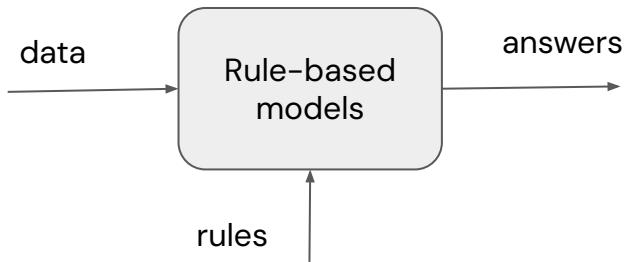
Initially, software development relied on **explicit rules** coded by developers, where detailed instructions were written for every task.



# The Shift!

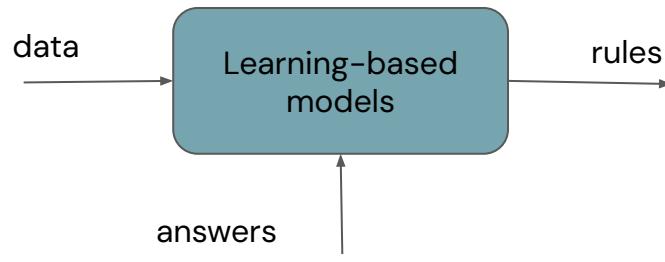
## Explicit Rules in Software Development

Initially, software development relied on **explicit rules** coded by developers, where detailed instructions were written for every task.



## The Rise of Machine Learning

With the advent of **machine learning**, computers began to learn from data, identifying patterns and making decisions without explicit programming. This transition allowed for more adaptive and intelligent software solutions.



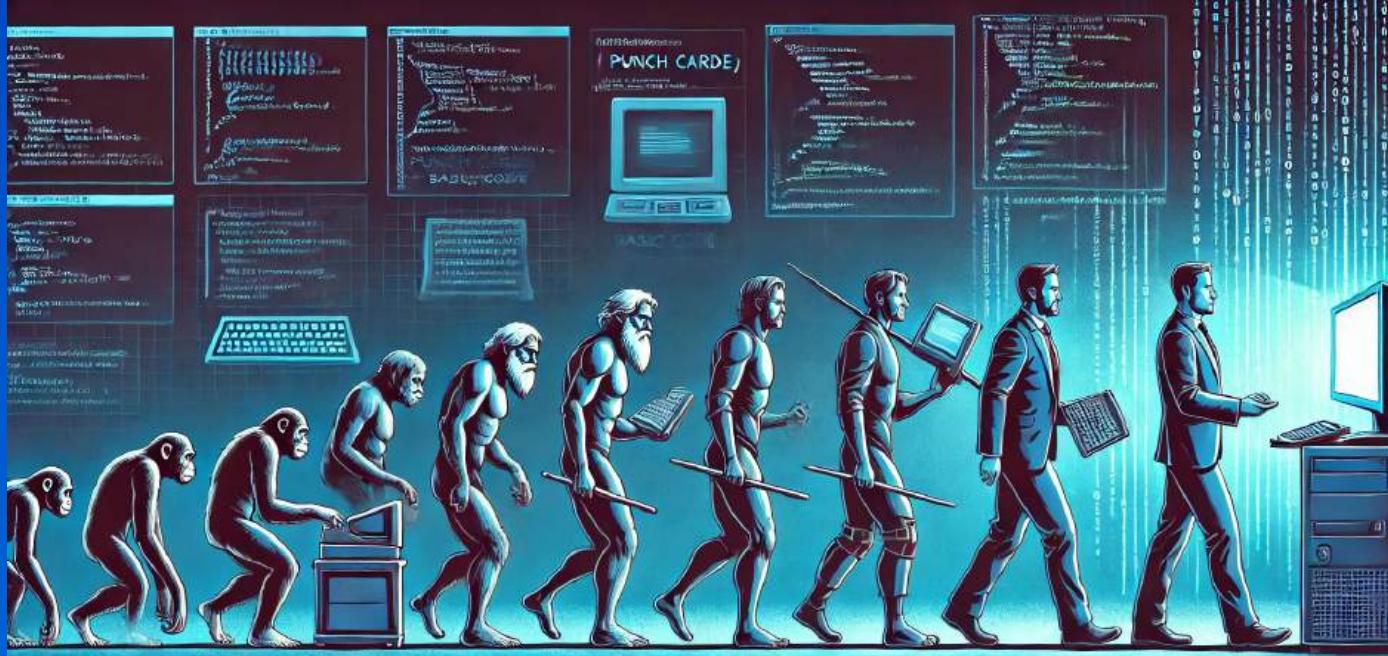
# Software Data

A huge wealth of various data exists in software lifecycle;

- source code,
- commits,
- bug reports,
- test cases,
- execution traces/logs,
- release notes,
- and real-world user feedback, etc.

# Evolution of Programming:

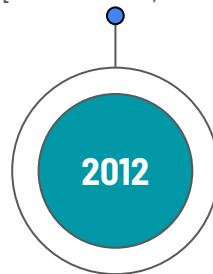
## How we learned form data?



# Development, Important Milestones

Repetitiveness and predictability of the software.

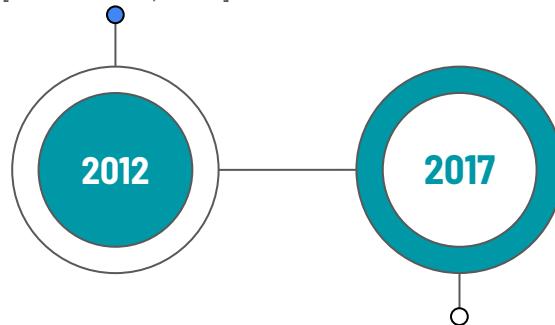
[Hindle et al., 2012]



# Development, Important Milestones

**Repetitiveness and predictability of the software.**

[Hindle et al., 2012]



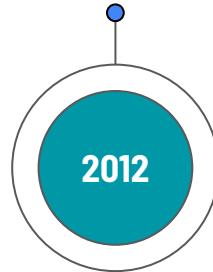
**Advanced statistical models (RNN, LSTM) are adapted to code modeling.**

[Hellendoorn et al., 2020]

# Development, Important Milestones

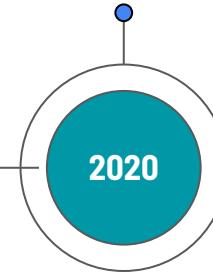
**Repetitiveness and predictability of the software.**

[Hindle et al., 2012]



**Transformer-based language models**

[Izadi et al., 2022, Kim et al., 2021]



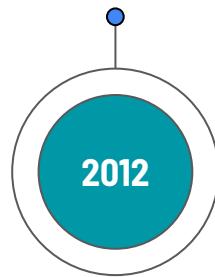
**Advanced statistical models (RNN, LSTM) are adapted to code modeling.**

[Hellendoorn et al., 2020]

# Development, Important Milestones

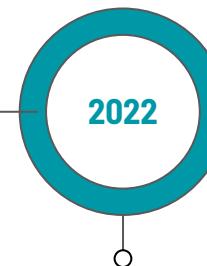
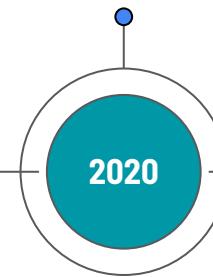
**Repetitiveness and predictability  
of the software.**

[Hindle et al., 2012, Allamanis et al., 2013]



**Transformer-based language  
models**

[Feng et al., 2020, Izadi et al., 2022, Kim et al., 2021]

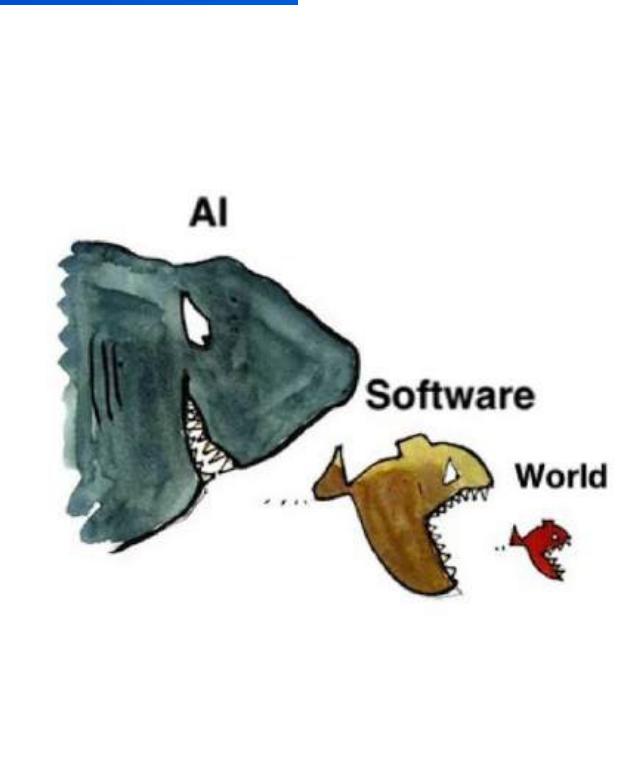


**Advanced statistical models  
(RNN, LSTM) are adapted to code  
modeling.**

[Yin and Neubig, 2017, Hellendoorn et al., 2020]

**Large Language Models (LLM) era.**  
[Copilot, Gemini, Code Whisperer, etc.]

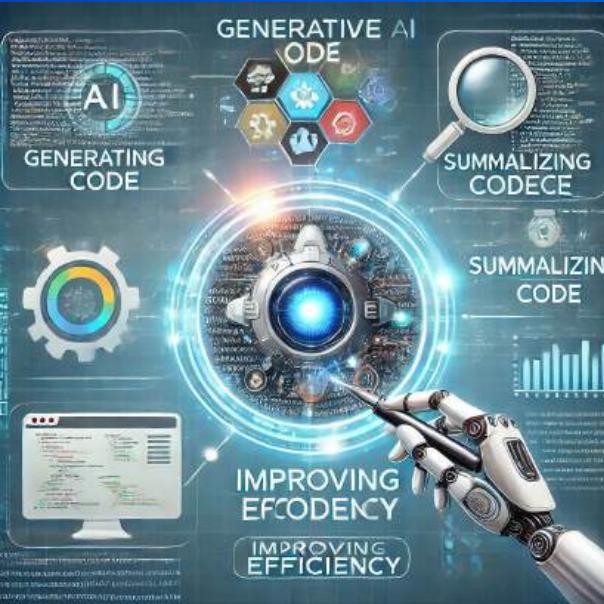
# The New Era: “AI is Eating Software”



We are now in an era where AI is taking over many aspects of software development. This phenomenon is often described as "AI eating software," highlighting AI's growing influence in:

- **Automating tasks:** Reducing the need for manual coding.
- **Enhancing capabilities:** Enabling more complex and intelligent software solutions.

# Generative AI: A Game Changer?



Generative AI is revolutionizing the development process by:

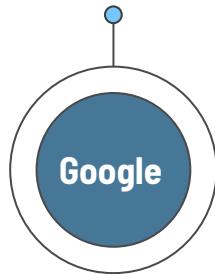
- **Generating code:** Automatically writing code based on given inputs.
- **Summarizing code:** Providing concise explanations of complex code.
- **Improving efficiency:** Reducing development time and effort.
- And many more tasks!

This technology is transforming how developers approach software creation.

# How good is Generative AI?

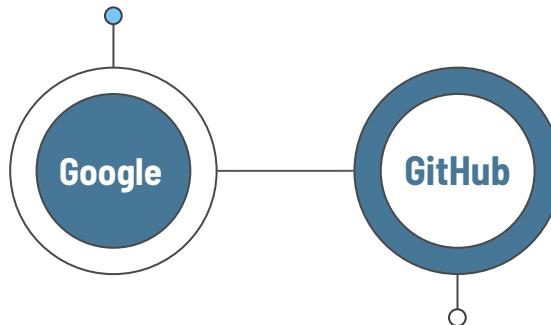
# How good are they?

**6% reduction in coding iteration time** with the new completion system! (July 2022)



# How good are they?

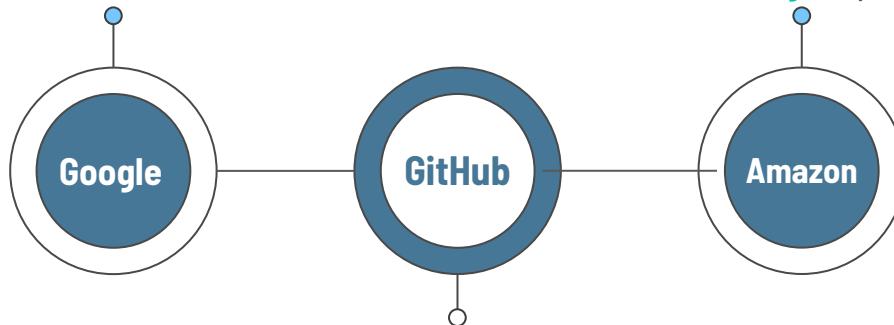
6% reduction in coding iteration time with the new completion system! (July 2022)



With Copilot, developers completed the task 55% faster! (September 2022)

# How good are they?

6% reduction in coding iteration time with the new completion system! (July 2022)

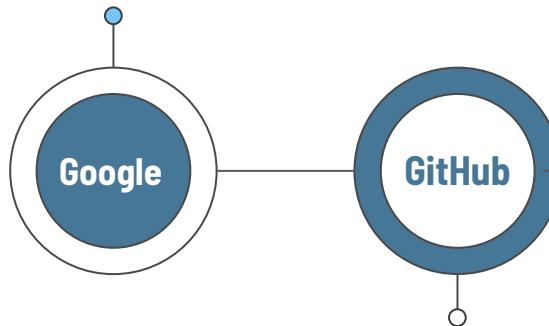


Participants with CodeWhisperer were 27% more likely to complete tasks successfully! (April 2023)

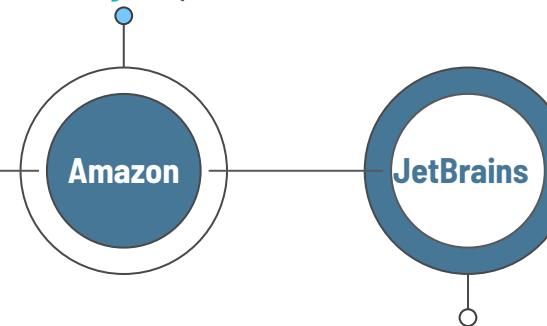
With Copilot, developers completed the task 55% faster! (September 2022)

# How good are they?

6% reduction in coding iteration time with the new completion system! (July 2022)



Participants with CodeWhisperer were 27% more likely to complete tasks successfully! (April 2023)



With Copilot, developers completed the task 55% faster! (September 2022)

77% of developers reported feeling more productive when using the AI assistant! (March 2024)

## Top Generative AI Stats for 2024

- The global generative market is worth **\$44.89 billion**

## Top Generative AI Stats for 2024

- The global generative market is worth **\$44.89 billion**
- **92% of Fortune 500 firms** have adopted generative AI

## Top Generative AI Stats for 2024

- The global generative market is worth **\$44.89 billion**
- **92% of Fortune 500 firms** have adopted generative AI
- **70% of Gen Z** have tried generative AI tools

## Top Generative AI Stats for 2024

- The global generative market is worth **\$44.89 billion**
- **92% of Fortune 500 firms** have adopted generative AI
- **70% of Gen Z** have tried generative AI tools
- **Nearly 9/10 of American jobs** could be impacted by generative AI

## Top Generative AI Stats for 2024

- The global generative market is worth **\$44.89 billion**
- **92% of Fortune 500 firms** have adopted generative AI
- **70% of Gen Z** have tried generative AI tools
- **Nearly 9/10 of American jobs** could be impacted by generative AI
- **95% of customer interactions** may involve AI by 2025

## Top Generative AI Stats for 2024

- The global generative market is worth **\$44.89 billion**
- **92% of Fortune 500 firms** have adopted generative AI
- **70% of Gen Z** have tried generative AI tools
- **Nearly 9/10 of American jobs** could be impacted by generative AI
- **95% of customer interactions** may involve AI by 2025
- **73% of marketing departments** use generative AI

# Top Generative AI Stats for 2024

- The global generative market is worth **\$44.89 billion**
- **92% of Fortune 500 firms** have adopted generative AI
- **70% of Gen Z** have tried generative AI tools
- **Nearly 9/10 of American jobs** could be impacted by generative AI
- **95% of customer interactions** may involve AI by 2025
- **73% of marketing departments** use generative AI
- AI could generate up to **97 million jobs by 2025**

# **When can we use it?**

# Generative AI in the Software Development Life Cycle (SDLC)

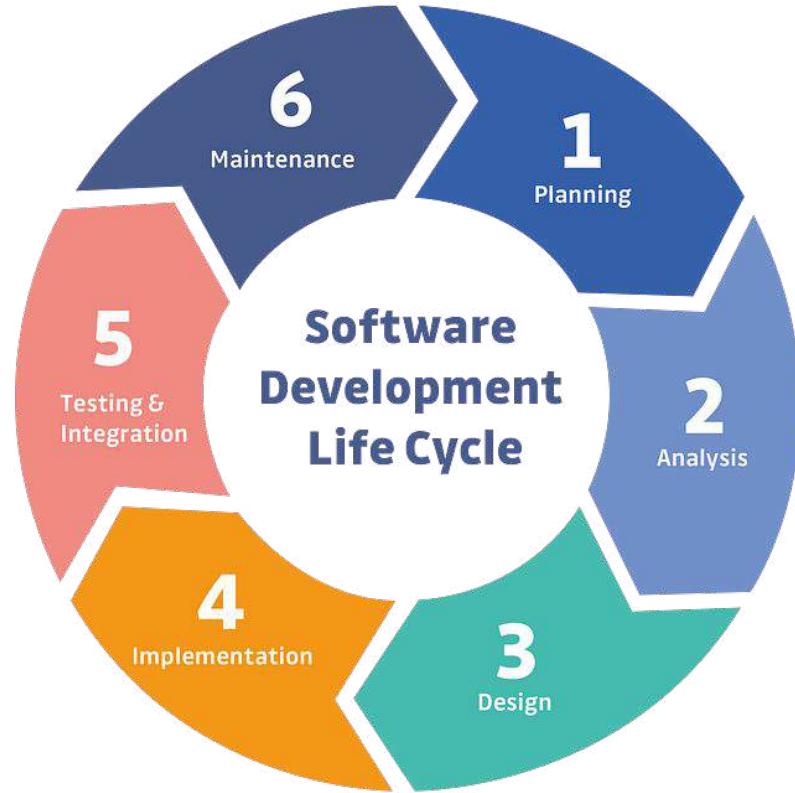


Image source:

<https://medium.com/agileinsider/how-does-the-sdlc-play-a-role-in-the-success-of-product-development-and-launch-a17baaac1054>

# Requirement Analysis

## Generating user stories and requirements

AI tools can analyze customer input, historical data, and market trends to automatically generate user stories or requirements, speeding up the process of gathering what the software needs to do.

**Example:** Imagine you're building a mobile banking app. Instead of manually writing user stories, AI could generate stories like:

- "As a user, I want to transfer money between accounts so that I can manage my funds easily."
- "As an admin, I want to review flagged transactions so that I can identify fraud."

## Creating design documents and architecture diagrams

During the design phase, AI can help architects and designers by suggesting system architecture diagrams, design patterns, or even generating documentation automatically.

**Example:** Let's say you're designing a cloud-based e-commerce platform. AI could generate an architecture diagram that includes a microservices design for scaling, using AWS Lambda for serverless functions and DynamoDB for database storage.

# Implementation

## Writing and optimizing code

AI can assist in writing, refactoring, or optimizing code. Instead of manually writing boilerplate code, you can leverage AI to handle repetitive tasks, suggest improvements, or even debug code.

For instance, if you need a function to handle payment processing, you could ask AI (e.g., GitHub Copilot) to write a Python function that interacts with an API:

```
def process_payment(amount, card_details):
    # Code for processing payment
    response = stripe.Charge.create(
        amount=amount,
        currency='usd',
        source=card_details
    )
    return response
```

It not only writes the function but can also suggest optimizations, such as error handling or performance improvements.

# Testing

## Generating test cases and identifying bugs

AI can automatically generate unit tests, integration tests, or even help with identifying bugs by analyzing the code and suggesting potential areas where errors might occur.

For instance, if you have a function for user registration, AI might generate test cases like:

- “Test with valid email and password.”
- “Test with an invalid email format.”
- “Test with a password that doesn’t meet security criteria.”

## Leveraging Large Language Models for Enhancing the Understandability of Generated Unit Tests

Amirhossein Deljouyi, Roham Kohestani, Maliheh Izadi, Andy Zaidman

*Delft University of Technology*

Delft, The Netherlands

a.deljouyi@tudelft.nl, r.kohestani@student.tudelft.nl, {m.izadi, a.e.zaidman}@tudelft.nl

## Suggesting updates and improvements

During the maintenance phase, AI can analyze system logs, code performance, and user feedback to suggest improvements or updates to the software. This makes it easier to keep systems optimized over time.

Let's say an AI tool detects that a certain part of the codebase is underperforming in terms of response time. It might suggest:

- "Refactor the database queries to improve load time."
- "Upgrade to a newer version of a framework for better security."

# Generative AI in the Software Development Life Cycle (SDLC)

## In summary:

In each phase of the SDLC, generative AI can act as an assistant

- reducing the time spent on repetitive tasks,
- improving accuracy, and
- suggesting intelligent improvements.

By integrating AI, developers can focus more on

**high-level decision-making** and creative **problem-solving**.

# AI Coding Assistants

# Current Tools: Empowering Developers

Several tools are empowering developers with generative AI capabilities:

These tools are making the development process more efficient and intuitive.



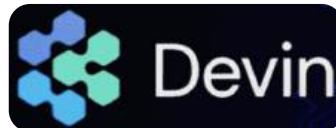
## JetBrains AI Assistant

Enhances code completion and debugging.



## Copilot

Assists in writing code by suggesting lines and functions.



## Devin

An AI agent that helps with project management and code reviews.

# Code Generation

# Code Generation Granularities

## Token prediction

This level of code completion involves predicting and suggesting the next token (such as a variable, keyword, or operator) in the code.

```
def add_two_numbers(a, b):  
    return a + # token suggestion -> 'b'
```

# Code Generation Granularities

## Token prediction

This level of code completion involves predicting and suggesting the next token (such as a variable, keyword, or operator) in the code.

## Full line completion

Full line completion predicts an entire line of code based on the current context. This can help speed up coding by suggesting a full logical step, often considering variable names or typical patterns.

```
def add_two_numbers(a, b):  
    return a + # token suggestion -> 'b'
```

```
if user.is_authenticated():  
    # Completion suggestion  
    return user.profile
```

# Code Generation Granularities

## Token prediction

This level of code completion involves predicting and suggesting the next token (such as a variable, keyword, or operator) in the code.

```
def add_two_numbers(a, b):  
    return a + # token suggestion -> 'b'
```

## Full line completion

Full line completion predicts an entire line of code based on the current context. This can help speed up coding by suggesting a full logical step, often considering variable names or typical patterns.

```
if user.is_authenticated():  
    # Completion suggestion  
    return user.profile
```

## Full block, function, or file generations

At this level, code completion suggests an entire block of code, such as a function definition, or even auto-generates files based on patterns.

```
public class Car {  
    private String make;  
    private String model;  
  
    // Full block suggestion  
    public Car(String make, String model) {  
        this.make = make;  
        this.model = model;  
    }  
  
    public String getMake() {  
        return make;  
    }  
  
    public String getModel() {  
        return model;  
    }
```

# Code4Me: Independent evaluation of open-weight LLMs4Code inside IDE



Maliheh Izadi



Jonathan Katzy



Tim Van Dam



Marc Otten



Răzvan Popescu



Arie van Deursen

# Code4Me: open-source line completion plugin



[Open-source plugins](#)

JetBrains

VSCode

# Code4Me: open-source line completion plugin



Open-source plugins

JetBrains

VSCode



Open weight LLMs

InCoder (Meta)

UniXcoder (Microsoft)

CodeGPT (Microsoft)

# Code4Me: open-source line completion plugin



Open-source plugins

JetBrains

VSCode



Open weight LLMs

InCoder (Meta)

UniXcoder (Microsoft)

CodeGPT (Microsoft)



Line Completion

Granularity level of the task

# Code4Me: open-source line completion plugin



## Open-source plugins

JetBrains

VSCode



## Multilingual

Diverse set of programming languages



## Open weight LLMs

InCoder (Meta)

UniXcoder (Microsoft)

CodeGPT (Microsoft)



## Line Completion

Granularity level of the task

# Code4Me: open-source line completion plugin



## Open-source plugins

JetBrains

VSCode



## Multilingual

Diverse set of programming languages



## Open weight LLMs

InCoder (Meta)

UniXcoder (Microsoft)

CodeGPT (Microsoft)



## Downloads/Users

~2700 downloads

~1500 users



## Line Completion

Granularity level of the task

# Code4Me: open-source line completion plugin



## Open-source plugins

JetBrains

VSCode



## Multilingual

Diverse set of programming languages



## Open weight LLMs

InCoder (Meta)

UniXcoder (Microsoft)

CodeGPT (Microsoft)



## Downloads/Users

~2700 downloads

~1500 users



## Line Completion

Granularity level of the task



## Telemetry data

Collect data from developers for evaluation

# Code4Me



code4me-me /  
code4me

<> Code Issues 3 Pull requests 1 Actions Projects ...

eye, fork, star icons

Two Automatic code completion IDE extensions for @JetBrains and @microsoft/vscode based on Transformer-based large language models for source code.

🔗 [code4me.me](#)  
Apache-2.0 license  
⭐ 54 stars ⚡ 4 forks 🏹 6 watching 🔔 6 Branches 🏷 11 Tags ⚡ Activity  
Custom properties  
🌐 Public repository · Generated from [JetBrains/intelliJ-platform-plugin-template](#)



```
public void save() {
    PasswordSafe.getInstance().
}
    ↗ save(this.settings);
    ↗ savePassword(credentials, settings);
    ↗ setPassword(settings.getUserToken(), true, false);
    ↗ setPassword(CredentialAttributes attributes, String password)
```

# How to use GenAI in development?

# Utilizing Generative AI: Settings

Generative AI can be used in different ways, depending on how much guidance or context you provide to the model. The three main techniques are **Zero-shot**, **Few-shot**, and **Prompt Engineering**.

# Zero-shot

In a zero-shot setting, you give the AI a task with no prior examples. The AI generates an output based solely on its training, without needing any additional context or guidance.

**Example:** Imagine you want to generate a function that converts Celsius to Fahrenheit in Python.

**Prompt:** "Write a Python function that converts Celsius to Fahrenheit."

```
def celsius_to_fahrenheit(celsius):
    return (celsius * 9/5) + 32
```

# Few-shot

In a few-shot setting, you provide a few examples to guide the AI. This helps it understand the task better and generate more accurate or customized responses.

**Example:** Let's say you need to create a function to format strings in different ways, like capitalizing or reversing them. You provide a couple of examples to show the AI what you expect:

**Prompt:** "Here are some examples of string formatting functions. Now, write one that capitalizes every word. Consider the following examples:"

```
def reverse_string(s):
    return s[::-1]

def to_lowercase(s):
    return s.lower()
```

# Few-shot

In a few-shot setting, you provide a few examples to guide the AI. This helps it understand the task better and generate more accurate or customized responses.

**Example:** Let's say you need to create a function to format strings in different ways, like capitalizing or reversing them. You provide a couple of examples to show the AI what you expect:

**Prompt:** "Here are some examples of string formatting functions. Now, write one that capitalizes every word. Consider the following examples:"

```
def reverse_string(s):
    return s[::-1]

def to_lowercase(s):
    return s.lower()
```

```
def capitalize_words(s):
    return s.capitalize_words()
```

# Prompt Engineering

Prompt engineering involves carefully crafting your prompt to guide the AI to produce the desired output. The quality of the prompt directly impacts the quality of the AI's response.

**Example:** Suppose you want the AI to generate a Python function that calculates the factorial of a number. A poorly designed prompt might lead to incorrect or inefficient code. A well-engineered prompt can be more specific and lead to better results.

**Poor Prompt:** "Write a function for factorial."

**AI Output** (which might be correct, but not optimal):

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

# Prompt Engineering

Prompt engineering involves carefully crafting your prompt to guide the AI to produce the desired output. The quality of the prompt directly impacts the quality of the AI's response.

**Example:** Suppose you want the AI to generate a Python function that calculates the factorial of a number. A poorly designed prompt might lead to incorrect or inefficient code. A well-engineered prompt can be more specific and lead to better results.

**Poor Prompt:** "Write a function for factorial."

**AI Output** (which might be correct, but not optimal):

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

**Well-Engineered Prompt:** "Write a Python function to calculate the factorial of a number using an iterative approach. The function should handle large inputs efficiently."

**AI Output**

```
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
```

# How to write better prompts?

# Crafting Effective Prompts

Good prompts can guide AI tools to produce better results, and there are a few key points to keep in mind:

- **Be specific:** Clearly describe what you want the AI to do. Vague or broad prompts can lead to undesired results.
- **Provide context:** Give enough background information so the AI understands the goal. The more relevant context, the better the output.
- **Use examples:** Show the AI what you expect. Providing examples clarifies the desired style, format, or type of output.
- **Iterate and refine:** Sometimes the first result isn't perfect. Adjust your prompt based on the output to improve the results. This iterative process helps tailor the output more precisely.

Now, let's look at examples using code to illustrate these tips for different AI tools.

# Be specific

Let's say you're asking an AI tool to generate Python code for reading a file.

A vague prompt like below.  
This can result in the AI  
generating simple code without  
error handling or customization.

```
1 # Vague prompt:  
2 "Write code to read a file."
```

# Be specific

Let's say you're asking an AI tool to generate Python code for reading a file.

A vague prompt like below.  
This can result in the AI  
generating simple code without  
error handling or customization.

```
1 # Vague prompt:  
2 "Write code to read a file."
```

Instead, be more specific:

```
# Specific prompt:  
"Write Python code to read a file, \  
handle exceptions if the file is missing, \  
and print each line."  
  
# Generated code:  
try:  
    with open('example.txt', 'r') as file:  
        for line in file:  
            print(line.strip())  
except FileNotFoundError:  
    print("The file was not found.")
```

# Provide Context

Without context, the AI might not know which programming language or framework you're working in. For example:

A vague prompt like:

```
# Context-free prompt:  
"Write a function to calculate the area of a circle."
```

# Provide Context

Without context, the AI might not know which programming language or framework you're working in. For example:

A vague prompt like:

```
# Context-free prompt:  
"Write a function to calculate the area of a circle."
```

But if you provide context about the programming language, it will generate the correct syntax:

```
1 # Prompt with context:  
2 "In Python, write a function to calculate \  
3 the area of a circle, \  
4 using a parameter for the radius."  
5  
6 # Generated Python code:  
7 import math  
8  
9 def area_of_circle(radius):  
10     return math.pi * radius**2
```

## Use Examples

Giving examples can help the AI better understand the format or structure you need. For instance, you want to generate test cases for a function:

Prompt without examples:

```
1 # Without examples:  
2 "Generate test cases for a \  
3 |function that adds two numbers."
```

# Use Examples

Prompt without examples:

```
1 # Without examples:  
2 "Generate test cases for a \  
3 |function that adds two numbers."
```

Giving examples can help the AI better understand the format or structure you need. For instance, you want to generate test cases for a function:

Instead, provide an example of how you want the test cases to look:

```
1 # With examples:  
2 "Generate test cases for \  
3 |a function that adds two numbers. \  
4 |Example format: test_add(1, 2, expected=3)."\  
5  
6 # Generated code:  
7 def test_add():  
8     assert add(1, 2) == 3  
9     assert add(-1, -1) == -2  
10    assert add(0, 0) == 0  
11    assert add(100, 200) == 300
```

## Iterate and Refine

Let's say you initially ask the AI for a solution to sorting a list in Python:

Initial prompt:

```
1 # First prompt:  
2 "Write Python code to \  
3 |sort a list of numbers."  
4  
5 # First result:  
6 numbers = [3, 1, 4, 1, 5, 9]  
7 numbers.sort()
```

## Iterate and Refine

Let's say you initially ask the AI for a solution to sorting a list in Python:

Initial prompt:

```
1 # First prompt:  
2 "Write Python code to \  
3 |sort a list of numbers."  
4  
5 # First result:  
6 numbers = [3, 1, 4, 1, 5, 9]  
7 numbers.sort()
```

You realize you also want the sorted list to be returned, not just sorted in place. So, you refine the prompt:

```
1 # Refined prompt:  
2 "Write Python code to sort \  
3 a list of numbers \  
4 and return the sorted list, \  
5 without modifying the original list."  
6  
7 # Refined result:  
8 numbers = [3, 1, 4, 1, 5, 9]  
9 sorted_numbers = sorted(numbers)
```

# Key Takeaways

**Zero-shot:** Useful for simple tasks where you don't need to provide examples.

**Few-shot:** Great for guiding the AI with a couple of examples to produce more relevant or customized results.

**Prompt Engineering:** Helps in refining the AI's output by carefully designing your prompt for more precise results.

# Resources for better prompt crafting

**The Prompt Report: A Systematic Survey of Prompting Techniques (paper)**  
<https://arxiv.org/pdf/2406.06608>

**The Prompt Report: A Systematic Survey of Prompting Techniques**

Sander Schulhoff<sup>2,3\*</sup>, Michael Ilic<sup>1\*</sup>, Nishant Balepur<sup>1</sup>, Konstantine Kahadze<sup>1</sup>, Amanda Liu<sup>1</sup>, Chenglei Si<sup>1</sup>, Yinheng Li<sup>2</sup>, Aayush Gupta<sup>1</sup>, HyoJung Han<sup>1</sup>, Sevien Schulhoff<sup>1</sup>, Pranav Sandeep Dulepet<sup>1</sup>, Saurav Vidyadhara<sup>1</sup>, Dayeon Kif<sup>1</sup>, Sweta Agrawal<sup>1,2</sup>, Chau Pham<sup>1,3</sup>, Gerson Krolo<sup>1</sup>, Felileen Li<sup>1</sup>, Hudson Tao<sup>1</sup>, Ashay Srivastava<sup>1</sup>, Hevander Da Costa<sup>1</sup>, Saloni Gupta<sup>1</sup>, Megan L. Rogers<sup>2</sup>, Inna Goncharenco<sup>3</sup>, Giuseppe Sarli<sup>1,10</sup>, Igor Galynker<sup>11</sup>, Denis Peskoff<sup>1</sup>, Marine Carpuat<sup>1</sup>, Jules White<sup>6</sup>, Shyamal Anadkat<sup>2</sup>, Alexander Hoyle<sup>1</sup>, Philip Resnik<sup>1</sup>, University of Maryland<sup>1</sup>, Learn Prompting<sup>1</sup>, OpenAI<sup>3</sup>, Stanford<sup>4</sup>, Microsoft<sup>5</sup>, Vanderbilt<sup>6</sup>, Princeton<sup>7</sup>, Texas State University<sup>8</sup>, Icahn School of Medicine<sup>9</sup>, ASST Brianza<sup>10</sup>, Mount Sinai Beth Israel<sup>11</sup>, Instituto de Telecomunicações<sup>12</sup>, University of Massachusetts Amherst<sup>13</sup>, sschulho@umd.edu, mili@umd.edu, resnik@umd.edu

**Abstract**

Generative Artificial Intelligence (GenAI) systems are being increasingly deployed across all parts of industry and research settings. Developers and end users interact with these systems through the use of prompting or prompt engineering. While prompting is a widespread and highly researched concept, there exists conflicting terminology and a poor ontological understanding of what constitutes a prompt due to the area's nascentcy. This paper establishes a structured understanding of prompts, by assembling a taxonomy of prompting techniques and analyzing their use. We present a comprehensive vocabulary of 33 vocabulary terms, a taxonomy of 38 text-only prompting techniques, and 40 techniques for other modalities. We further present a meta-analysis of the entire literature on natural language prefix-prompting.

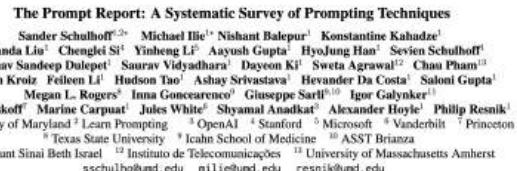
# Resources for better prompt crafting

## The Prompt Report: A Systematic Survey of Prompting Techniques (paper)

<https://arxiv.org/pdf/2406.06608>

## Anthropic's Prompt Engineering Interactive Tutorial (Repo)

[https://github.com/anthropics/courses/tree/master/prompt\\_engineering\\_interactive\\_tutorial](https://github.com/anthropics/courses/tree/master/prompt_engineering_interactive_tutorial)



# Resources for better prompt crafting

## The Prompt Report: A Systematic Survey of Prompting Techniques (paper)

<https://arxiv.org/pdf/2406.06608>

**The Prompt Report: A Systematic Survey of Prompting Techniques**

Sander Schulhoff<sup>2,3\*</sup>, Michael Ilic<sup>1\*</sup>, Nishant Balepur<sup>1</sup>, Konstantine Kahadze<sup>1</sup>, Amanda Liu<sup>1</sup>, Chenglei Si<sup>1</sup>, Yinheng Li<sup>2</sup>, Aayush Gupta<sup>1</sup>, HyoJung Han<sup>1</sup>, Sevien Schulhoff<sup>1</sup>, Pranav Sandeep Dulepet<sup>1</sup>, Saurav Vidyadhara<sup>1</sup>, Dayeon KF<sup>1</sup>, Sweta Agrawal<sup>1,2</sup>, Chau Pham<sup>1,3</sup>, Gerson Krolo<sup>1</sup>, Féleena Li<sup>1</sup>, Hudson Tao<sup>1</sup>, Ashay Srivastava<sup>1</sup>, Hevander Da Costa<sup>1</sup>, Saloni Gupta<sup>1</sup>, Megan L. Rogers<sup>2</sup>, Inna Goncharenco<sup>3</sup>, Giuseppe Sarli<sup>1,10</sup>, Igor Galynker<sup>11</sup>, Denis Peskoff<sup>2</sup>, Marine Carpuat<sup>1</sup>, Julie White<sup>6</sup>, Shyamal Anadkat<sup>2</sup>, Alexander Hoyle<sup>1</sup>, Philip Resnik<sup>1</sup>, University of Maryland<sup>1</sup>, Learn Prompting<sup>2</sup>, OpenAI<sup>3</sup>, Stanford<sup>4</sup>, Microsoft<sup>5</sup>, Vanderbilt<sup>6</sup>, Princeton<sup>7</sup>, Texas State University<sup>8</sup>, Icahn School of Medicine<sup>9</sup>, ASST Brianza<sup>10</sup>, Mount Sinai Beth Israel<sup>11</sup>, Instituto de Telecomunicações<sup>12</sup>, University of Massachusetts Amherst<sup>13</sup>, sschulho@umd.edu nilee@umd.edu resnik@umd.edu

**Abstract**

Generative Artificial Intelligence (GenAI) systems are being increasingly deployed across all parts of industry and research settings. Developers and end users interact with these systems through the use of prompting or prompt engineering. While prompting is a widespread and highly researched concept, there exists conflicting terminology and a poor ontological understanding of what constitutes a prompt due to the area's nascentcy. This paper establishes a structured understanding of prompts, by assembling a taxonomy of prompting techniques and analyzing their use. We present a comprehensive vocabulary of 33 vocabulary terms, a taxonomy of 38 text-only prompting techniques, and 40 techniques for other modalities. We further present a meta-analysis of the entire literature on natural language prefix-prompting.

15 Jul 2024

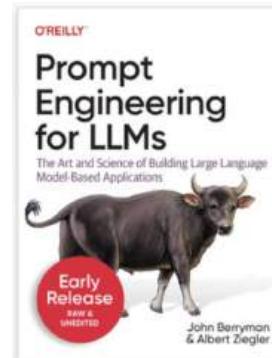
## Anthropic's Prompt Engineering Interactive Tutorial (Repo)

[https://github.com/anthropics/courses/tree/master/prompt\\_engineering\\_interactive\\_tutorial](https://github.com/anthropics/courses/tree/master/prompt_engineering_interactive_tutorial)

## Prompt Engineering for LLMs (Book)

By John Berryman, Albert Ziegler

<https://www.oreilly.com/library/view/prompt-engineering-for/9781098156145/>



# Implications

# Ethical, Legal, and Security Implications



## Ethical: Ensuring Fairness, Transparency, and Accountability

When integrating generative AI into software development, ethical concerns become a key issue. AI systems can unintentionally perpetuate biases or make decisions that are difficult to explain.

# Ethical, Legal, and Security Implications



## **Ethical: Ensuring Fairness, Transparency, and Accountability**

When integrating generative AI into software development, ethical concerns become a key issue. AI systems can unintentionally perpetuate biases or make decisions that are difficult to explain.

**Fairness:** AI models can unintentionally reflect biases present in the data they are trained on. For example, if an AI system used to automate job application screening is trained on historical data where certain groups were underrepresented or discriminated against, it might continue this pattern of bias.

# Ethical, Legal, and Security Implications



## Ethical: Ensuring Fairness, Transparency, and Accountability

When integrating generative AI into software development, ethical concerns become a key issue. AI systems can unintentionally perpetuate biases or make decisions that are difficult to explain.

**Fairness:** AI models can unintentionally reflect biases present in the data they are trained on. For example, if an AI system used to automate job application screening is trained on historical data where certain groups were underrepresented or discriminated against, it might continue this pattern of bias.

**Transparency:** Many AI models, especially deep learning models, are often referred to as "black boxes" because their decision-making process can be difficult to understand. Developers need to ensure that AI systems are designed in a way that their decisions can be explained to users.

# Ethical, Legal, and Security Implications



## Ethical: Ensuring Fairness, Transparency, and Accountability

When integrating generative AI into software development, ethical concerns become a key issue. AI systems can unintentionally perpetuate biases or make decisions that are difficult to explain.

**Fairness:** AI models can unintentionally reflect biases present in the data they are trained on. For example, if an AI system used to automate job application screening is trained on historical data where certain groups were underrepresented or discriminated against, it might continue this pattern of bias.

**Transparency:** Many AI models, especially deep learning models, are often referred to as "black boxes" because their decision-making process can be difficult to understand. Developers need to ensure that AI systems are designed in a way that their decisions can be explained to users.

**Accountability:** When an AI makes a decision, who is responsible if something goes wrong? In software development, this might be a concern when deploying AI systems in critical applications, such as healthcare or autonomous driving.

# Ethical, Legal, and Security Implications



## Legal: Navigating Intellectual Property Rights and Compliance

Generative AI systems are often trained on vast amounts of data, which can include code, designs, or other intellectual property (IP) that belongs to someone else.

# Ethical, Legal, and Security Implications



## Legal: Navigating Intellectual Property Rights and Compliance

Generative AI systems are often trained on vast amounts of data, which can include code, designs, or other intellectual property (IP) that belongs to someone else.

**Intellectual Property Rights:** When generative AI is used to create software or code, there may be concerns about who owns the generated code. If the AI is trained on copyrighted material, questions arise about whether the generated code is a derivative work of the original IP.

# Ethical, Legal, and Security Implications



## Legal: Navigating Intellectual Property Rights and Compliance

Generative AI systems are often trained on vast amounts of data, which can include code, designs, or other intellectual property (IP) that belongs to someone else.

**Intellectual Property Rights:** When generative AI is used to create software or code, there may be concerns about who owns the generated code. If the AI is trained on copyrighted material, questions arise about whether the generated code is a derivative work of the original IP.

**Compliance:** Different countries have various laws about data usage, and developers must ensure that their AI systems comply with these regulations. For example, the European Union's General Data Protection Regulation (GDPR) has strict rules about how personal data can be processed.

# Ethical, Legal, and Security Implications



## Legal: Navigating Intellectual Property Rights and Compliance

Generative AI systems are often trained on vast amounts of data, which can include code, designs, or other intellectual property (IP) that belongs to someone else.

**Intellectual Property Rights:** When generative AI is used to create software or code, there may be concerns about who owns the generated code. If the AI is trained on copyrighted material, questions arise about whether the generated code is a derivative work of the original IP.

**Compliance:** Different countries have various laws about data usage, and developers must ensure that their AI systems comply with these regulations. For example, the European Union's General Data Protection Regulation (GDPR) has strict rules about how personal data can be processed.

**Example:** A company uses an AI system to generate new software components. Later, it's discovered that the AI was trained on copyrighted open-source projects that have restrictive licenses (e.g., GPL). The company may face legal issues because the generated code could be considered a derivative work of the open-source projects, which they are not allowed to use in a proprietary product without following the open-source license's requirements.

# Ethical, Legal, and Security Implications



## Security: Protecting Against Vulnerabilities and Malicious Use

AI-generated code or models can introduce security vulnerabilities, either unintentionally or through malicious intent.

# Ethical, Legal, and Security Implications



## Security: Protecting Against Vulnerabilities and Malicious Use

AI-generated code or models can introduce security vulnerabilities, either unintentionally or through malicious intent.

**Vulnerabilities:** If an AI model generates code without proper security checks, it could introduce flaws like SQL injection vulnerabilities, buffer overflows, or other security weaknesses that attackers could exploit.

# Ethical, Legal, and Security Implications



## Security: Protecting Against Vulnerabilities and Malicious Use

AI-generated code or models can introduce security vulnerabilities, either unintentionally or through malicious intent.

**Vulnerabilities:** If an AI model generates code without proper security checks, it could introduce flaws like SQL injection vulnerabilities, buffer overflows, or other security weaknesses that attackers could exploit.

**Malicious Use:** Generative AI can also be used by malicious actors to create harmful code. For example, AI can be used to automate the creation of malware or phishing websites. Additionally, generative AI could be used to create sophisticated social engineering attacks.

# Ethical, Legal, and Security Implications



## Security: Protecting Against Vulnerabilities and Malicious Use

AI-generated code or models can introduce security vulnerabilities, either unintentionally or through malicious intent.

**Vulnerabilities:** If an AI model generates code without proper security checks, it could introduce flaws like SQL injection vulnerabilities, buffer overflows, or other security weaknesses that attackers could exploit.

**Malicious Use:** Generative AI can also be used by malicious actors to create harmful code. For example, AI can be used to automate the creation of malware or phishing websites. Additionally, generative AI could be used to create sophisticated social engineering attacks.

**Example:** An AI system generates code for a login form. However, it overlooks proper input validation, leading to a vulnerability where users could input malicious SQL code and gain unauthorized access to a system. In this case, the AI's generated code created a security hole that could be exploited.

# Interested to know more?

## The (ab)use of Open Source Code to Train Large Language Models

Ali Al-Kaswan  
Delft University of Technology  
Delft, The Netherlands  
ali.kaswan@tudelft.nl

Malibeh Izadi  
Delft University of Technology  
Delft, The Netherlands  
m.izadi@tudelft.nl

**Abstract.** In recent years, Large Language Models (LLMs) have gained significant popularity due to their ability to generate human-like text and their potential applications in various fields, such as Software Engineering. LLMs for Code are currently trusted on large unauthenticated corpora of source code scraped from the Internet. In this paper, we discuss the security, privacy, and licensing implications of memorisation. In this work, we will discuss the security, privacy, and licensing implications of memorisation. We argue why the use of open-source code to train LLMs for Code is problematic and how we can provide four sensible recommendations to address this issue.

### I. LANGUAGE MODELS FOR CODE

Large Language Models (LLMs) have gained significant attention in the field of Natural Language Processing (NLP) in recent years due to their ability to perform a wide range of NLP tasks. LLMs for Code are currently trained on large amounts of open-source code, which is often unauthenticated and contains many amounts of data, improve in accuracy as they grow from millions to billions of parameters. LLMs for code are trained on massive amounts of data and can learn the structure and syntax of programming languages, making them well-suited for tasks such as code summarization, generation, and completion [1–3]. LLMs are even making their way into commercial products, like GitHub's Copilot, Replicated's ChatCopilot, and others. Moreover, it has been shown that LLMs memorize large amounts of training data [4]. Memorisation enables the extraction of the data using Data Extraction Attacks. Some attacks have even been able to extract addresses and other personal information from public models [5]. Memorisation also impacts LLMs for code, with all its associated consequences. We will discuss these consequences in three categories: security, privacy, and licensing.

### II. SECURITY IMPLICATIONS

Test memorisation has strong security implications. Firstly, memory-mind code datasets are not curated or manually curated; the datasets could therefore contain many biases,<sup>1</sup> and instances of badly written or buggy and insecure code. A recent study found that around 40% of GitHub Copilot's code generations for MITRE's top 25 Common Weakness Enumerations, a list of the most dangerous software weaknesses were found to be vulnerable [4]. If these models become more prevalent and trusted, they can introduce more vulnerable code into software.

<sup>1</sup>Open GFIT-2 Know Your Place Number? - https://github.com/layA/zippyfuzz/Treeview. http://arxiv.org/abs/1908.03881  
<sup>2</sup>Not license Square: https://checkformln1992.github.io/  
<sup>3</sup>GitHub Copilot: https://github.com/MLPMLP

### III. PRIVACY IMPLICATIONS

Memorisation enables adversaries to access training data, and everything contained within, simply by accessing the model. This has major privacy implications since code can contain sensitive information, such as credit card API keys, and other sensitive information, such as logins and words discussed by developers. Code can also contain personal information like emails or contact information. If personal data is published on the internet, the data could be retracted and deleted from the source. But once it is mined and used to train an LLM, the information is forever embedded in a compact representation, which is queryable at scale. With query access to these models, an adversary can easily extract [6] and then then publish sensitive private. There are many reasons why one could publicly share private information: (1) simply by accident, or (2) a malicious actor could share this information in a doxing campaign [1]. Even if the data is published willingly, the owner has a certain use and audience in mind and might not wish to share this information with the entire world. This is referred to as the re-purposed data problem.<sup>2</sup>

### IV. LICENSING

Publicly available source code is also subject to licenses, some of which heavily regulate the use of the material. Initially, developers raised concerns about licensed code on social media. GitHub Copilot could be prompted to produce verbatim copies of copyrighted code, without providing the required attribution or license terms.<sup>3</sup> Similarly, Copilot was producing copyrighted code while attributing the wrong author and providing a wrong license.<sup>4</sup> Later, a lawsuit was filed against GitHub, Microsoft, and others, claiming that Copilot is violating the license of open-source code.<sup>5</sup>

Broadly, open-source code is licensed under two types of licenses. **Permissive licenses**, allow users to use, modify, and distribute the software for any purpose, without requiring that the user share their work. **Non-permissive licenses**, also known as "copyleft" licenses, require that users freely share their own software under the same license if they distribute

## Traces of Memorisation in Large Language Models for Code

Ali Al-Kaswan  
Delft University of Technology  
Delft, The Netherlands  
ali.kaswan@tudelft.nl  
Malibeh Izadi  
Delft University of Technology  
Delft, The Netherlands  
m.izadi@tudelft.nl  
Arie van Deursen  
Delft University of Technology  
Delft, The Netherlands  
arie.vandeursen@tudelft.nl

### ABSTRACT

Large language models have gained significant popularity because of their ability to generate human-like text and potential applications in various fields, such as Software Engineering. Large language models for code are currently trained on large unauthenticated corpora of source code scraped from the Internet. In this paper, we explore memorisation in large language models for code and compare the size of the traces of memorisation in large language models trained on natural language. We adopt an existing technique for natural language and construct a baseline for code. We show that by doing so, we can detect attacks. We run both benchmarks against a variety of models, and perform a data extraction attack. We find that large language models for code are vulnerable to data extraction attacks, like their natural language counterparts. From the training data that was identified to be potentially extractable, we were able to extract 47% from a large LLM for code, and 100% from a smaller one. We also observe that models memorise more, as their parameter count grows, and that their pre-training data are also vulnerable to attack. We also find that data owners are more likely to be higher than regular code or documentation and that different model architectures memorise different samples. Data leakage has seven outcomes, as we urge the research community to further investigate the extent of this problem using a wider range of models and detection techniques to build safeguards to mitigate this issue.

### CCS CONCEPTS

• Security and privacy • Software and its engineering • Computing methodologies → Machine learning

### KEYWORDS

Large Language Models, Privacy, Memorisation, Data Leakage

### ACM Reference Format:

Ali Al-Kaswan, Malibeh Izadi, and Arie van Deursen. 2024. Traces of Memorisation in Large Language Models for Code. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24)*, April 14–28, 2024, Lisbon, Portugal. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3597920.3601013



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.  
CC BY-NC 4.0  
https://creativecommons.org/licenses/by-nc/4.0/  
http://creativecommons.org/licenses/by-nc/4.0/legalcode

<sup>4</sup>StackOverflow License: https://stackexchange.com/help/licensing

# Ethical, Legal, and Security Implications



## Addressing These Implications is Essential for Responsible AI Integration

It's important for developers and organizations to actively address these concerns when integrating generative AI into their workflows. They must ensure the AI systems are designed, trained, and deployed in a way that promotes fairness, transparency, and security, while also staying within legal boundaries.

### Strategies to Address These Concerns:

**Ethical:** Regularly audit AI systems for bias, provide clear documentation of how AI decisions are made, and establish clear accountability frameworks.

**Legal:** Ensure the AI is trained on data that is legally allowed for use, and consult legal experts on intellectual property matters related to AI-generated outputs.

**Security:** Implement robust security checks on AI-generated code, and educate developers on the potential security risks of using AI-generated content.

# Will AI replace developers?

The future of software development is being shaped by AI and generative AI.

By embracing these technologies **responsibly**, developers can drive innovation and create more intelligent and efficient software solutions.

Engage with these tools, stay informed, and contribute to the evolving landscape of software development.



# AI4SE; a collaboration between TU Delft and JetBrains

- The 10th ICAI lab at TU Delft started in Oct 2023

## Management Team



TU Delft



Jetbrains

# AI4SE; a collaboration between TU Delft and JetBrains

- The 10th ICAI lab at TU Delft started in Oct 2023
- 5 research tracks

## Management Team



TU Delft



Jetbrains

# AI4SE; a collaboration between TU Delft and JetBrains

- The 10th ICAI lab at TU Delft started in Oct 2023
- 5 research tracks
  - **Track 1:** validation of generated code (Led by Annibale Panichella)

## Management Team



TU Delft



Jetbrains

# AI4SE; a collaboration between TU Delft and JetBrains

- The 10th ICAI lab at TU Delft started in Oct 2023
- 5 research tracks
  - **Track 1:** validation of generated code (Led by Annibale Panichella)
  - **Track 2:** Adapting LLMs to Code (Led by Maliheh Izadi)

## Management Team



TU Delft



Jetbrains

# AI4SE; a collaboration between TU Delft and JetBrains

- The 10th ICAI lab at TU Delft started in Oct 2023
- 5 research tracks
  - **Track 1:** validation of generated code (Led by Annibale Panichella)
  - **Track 2:** Adapting LLMs to Code (Led by Maliheh Izadi)
  - **Track 3:** Interactive and Aligned IDEs (Led by Maliheh Izadi)

## Management Team



TU Delft



Jetbrains

# AI4SE; a collaboration between TU Delft and JetBrains

- The 10th ICAI lab at TU Delft started in Oct 2023
- 5 research tracks
  - **Track 1:** validation of generated code (Led by Annibale Panichella)
  - **Track 2:** Adapting LLMs to Code (Led by Maliheh Izadi)
  - **Track 3:** Interactive and Aligned IDEs (Led by Maliheh Izadi)
  - **Track 4:** Utilizing runtime info (Led by Burcu Ozkan)

## Management Team



TU Delft



Jetbrains

# AI4SE; a collaboration between TU Delft and JetBrains

- The 10th ICAI lab at TU Delft started in Oct 2023
- 5 research tracks
  - **Track 1:** validation of generated code (Led by Annibale Panichella)
  - **Track 2:** Adapting LLMs to Code (Led by Maliheh Izadi)
  - **Track 3:** Interactive and Aligned IDEs (Led by Maliheh Izadi)
  - **Track 4:** Utilizing runtime info (Led by Burcu Ozkan)
  - **Track 5:** Improving programming education (Led by Gosia Migut)

## Management Team



TU Delft



Jetbrains

# AI4SE; a collaboration between TU Delft and JetBrains

- The 10th ICAI lab at TU Delft started in Oct 2023
- 5 research tracks
  - **Track 1:** validation of generated code (Led by Annibale Panichella)
  - **Track 2:** Adapting LLMs to Code (Led by Maliheh Izadi)
  - **Track 3:** Interactive and Aligned IDEs (Led by Maliheh Izadi)
  - **Track 4:** Utilizing runtime info (Led by Burcu Ozkan)
  - **Track 5:** Improving programming education (Led by Gosia Migut)
- **10 PhD students**

## Management Team



TU Delft



Jetbrains

# AI4SE; a collaboration between TU Delft and JetBrains

- The 10th ICAI lab at TU Delft started in Oct 2023
- 5 research tracks
  - **Track 1:** validation of generated code (Led by Annibale Panichella)
  - **Track 2:** Adapting LLMs to Code (Led by Maliheh Izadi)
  - **Track 3:** Interactive and Aligned IDEs (Led by Maliheh Izadi)
  - **Track 4:** Utilizing runtime info (Led by Burcu Ozkan)
  - **Track 5:** Improving programming education (Led by Gosia Migut)
- **10 PhD students**
- **Multitude of Bsc and MSc students for thesis and internships**

## Management Team



TU Delft



Jetbrains

# AI4SE; a collaboration between TU Delft and JetBrains

- The 10th ICAI lab at TU Delft started in Oct 2023
- 5 research tracks
  - **Track 1:** validation of generated code (Led by Annibale Panichella)
  - **Track 2:** Adapting LLMs to Code (Led by Maliheh Izadi)
  - **Track 3:** Interactive and Aligned IDEs (Led by Maliheh Izadi)
  - **Track 4:** Utilizing runtime info (Led by Burcu Ozkan)
  - **Track 5:** Improving programming education (Led by Gosia Migut)
- **10 PhD students**
- **Multitude of Bsc and MSc students for thesis and internships**

**Website:** <https://se.ewi.tudelft.nl/ai4se/>

**Reach out** to me or Mitchell Olsthoorn to know more.

**Contact:** m.izadi@tudelft.nl

## Management Team



TU Delft



Jetbrains



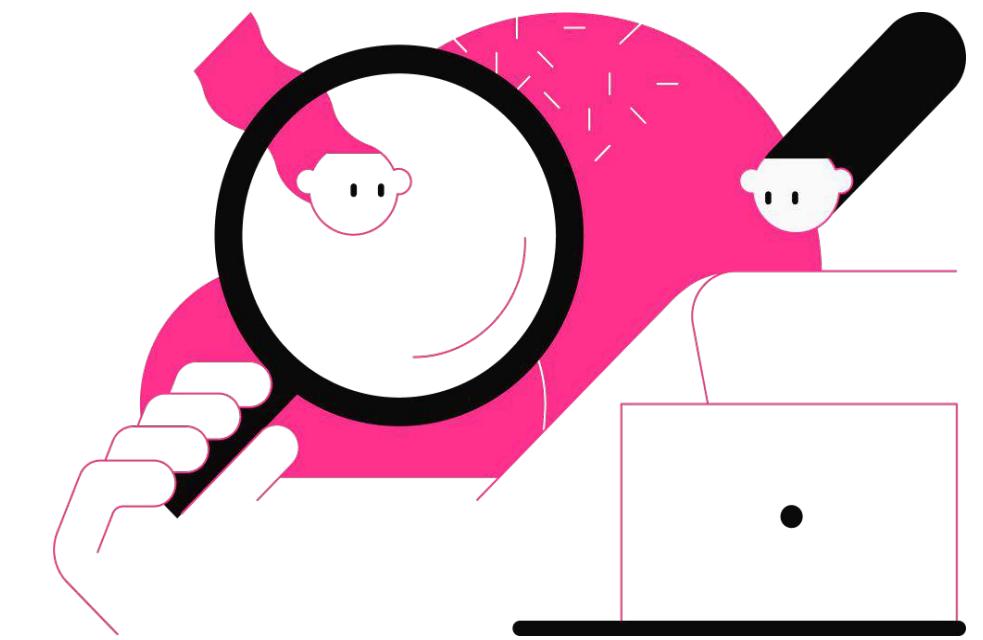
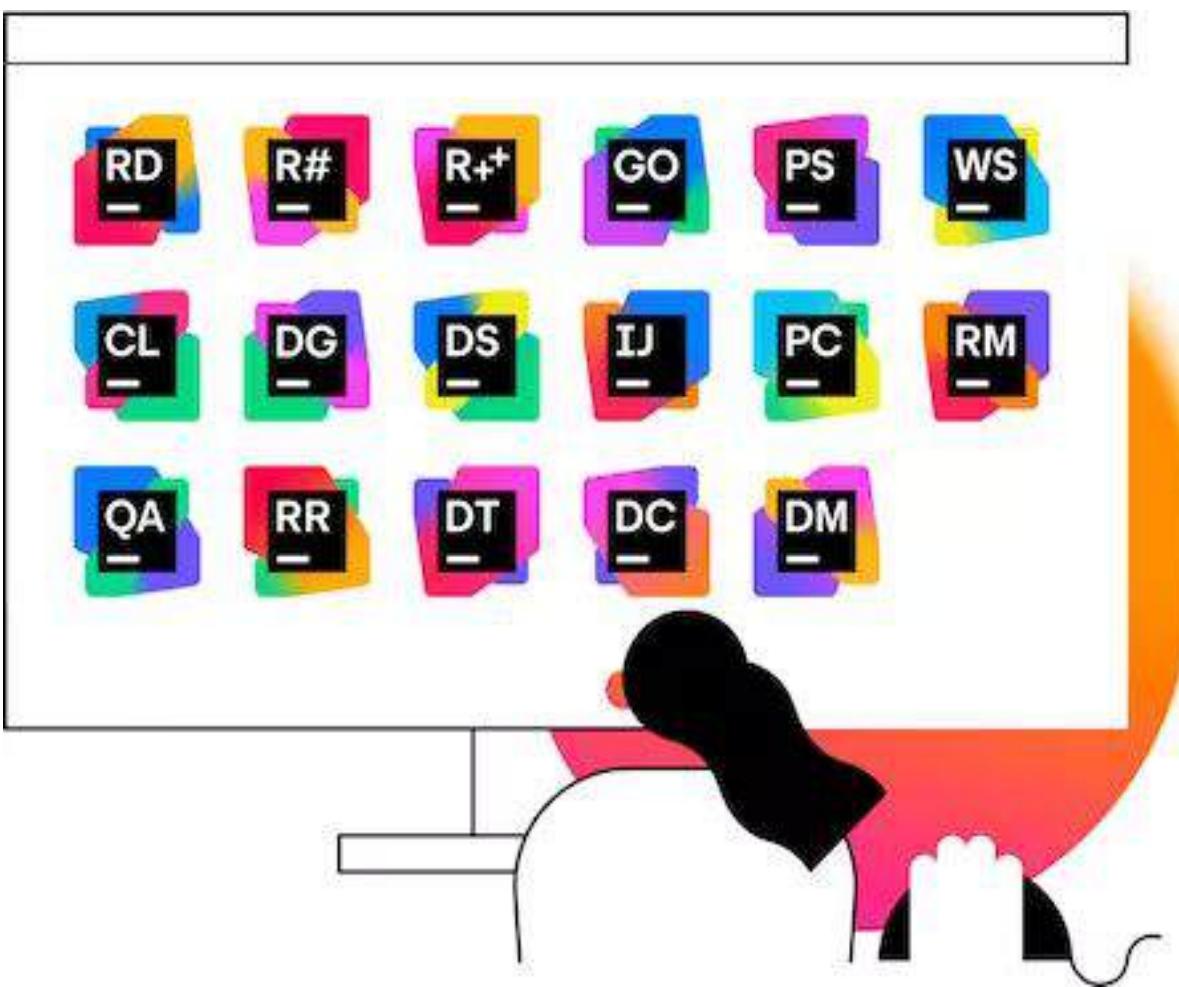
# Human - AI eXperience

**Agnia Sergeyuk**

Team Lead @ HAX Research,  
PhD @ TU Delft



# About me



**Human - AI  
eXperience**

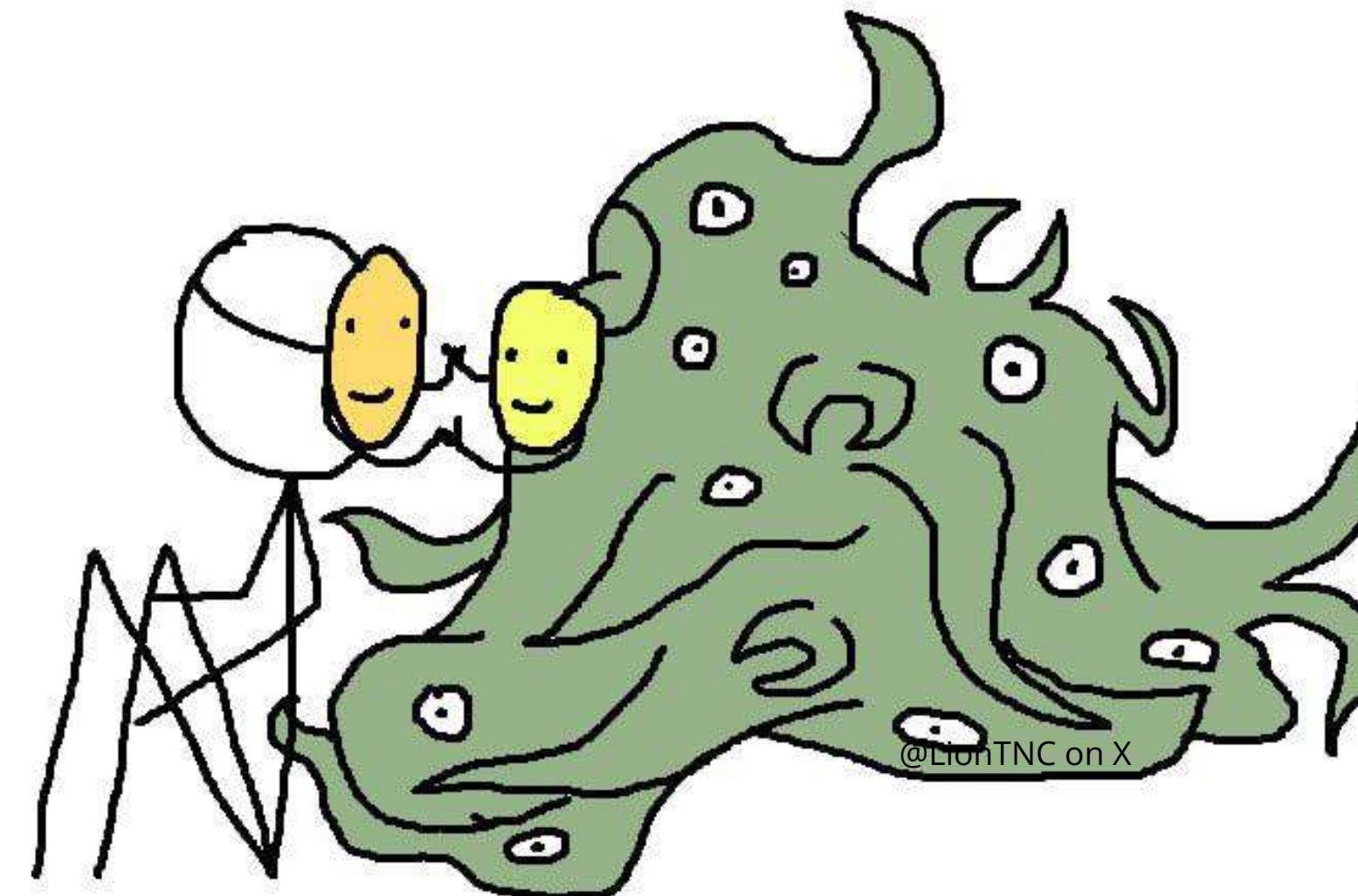
# **Human-Computer Interaction (HCI)**

**+**

## **LLMs & AI**

**=**

# **Human-AI eXperience (HAX)**



@LionTNC on X



# in-IDE HAX

How can Human-AI interactions within IDEs be optimized to improve developers' productivity and satisfaction?

**DESIGN**

What is the most effective HAX modality in an IDE?

**IMPACT**

How does HAX influence IDE users?

**QUALITY**

What are the critical attributes of AI output?  
How can they be enhanced to support IDE users?

# Design

- Systematizing methodologies and proposing principles for designing HAX
- Prototyping integration of AI assistants into programming environments

# Design principles

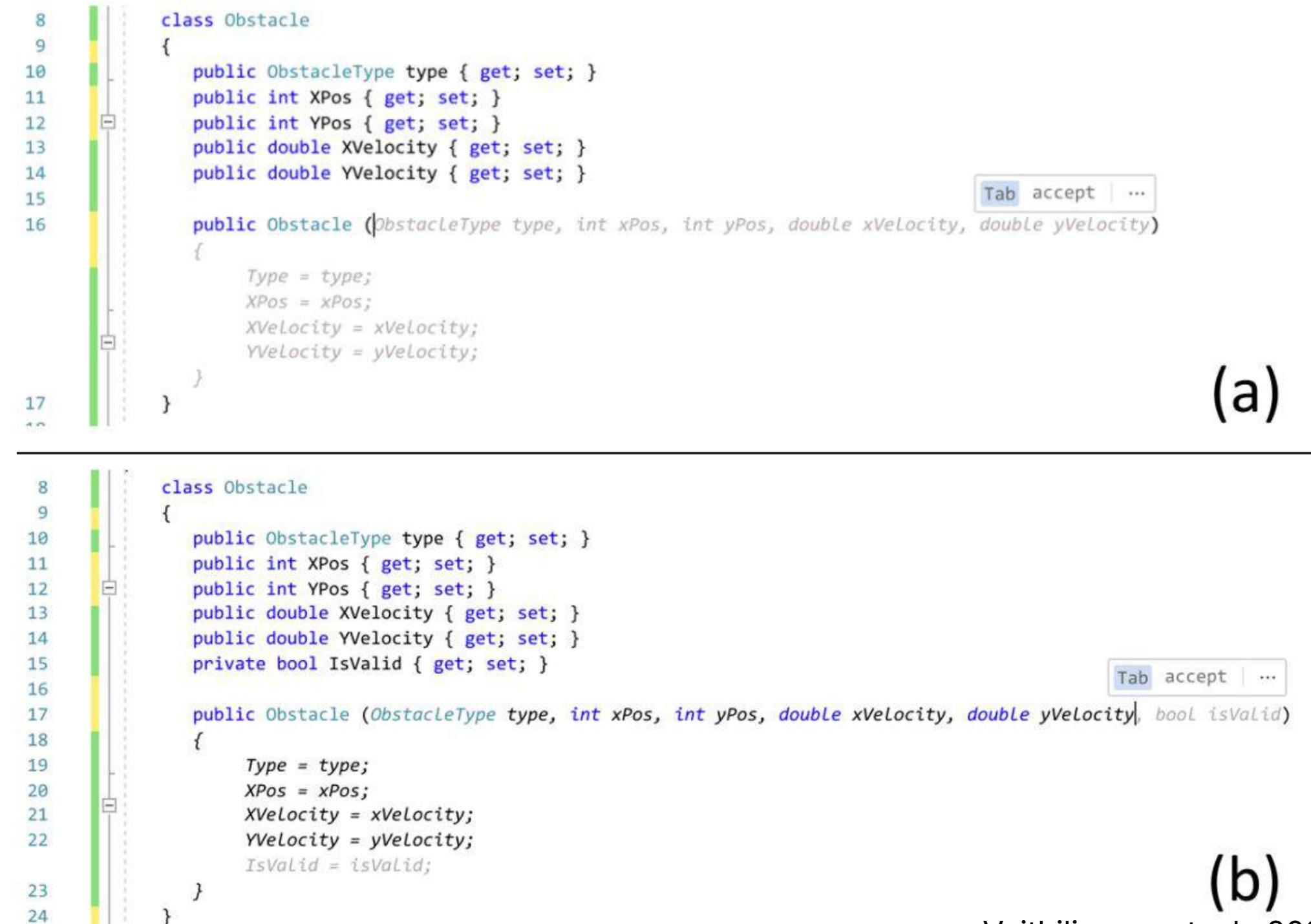
adaptability, clarity, and user-friendly interactions

- Communicating **probabilistic nature** and variable outputs — facilitating user annotation and visualization of differences.
- Accommodating **imperfection** — implementing user-driven controls, providing sandbox environments, communicating capabilities and limitations, with a focus on preventing risks and harms.

# Design principles

## for autocomplete

- glanceable suggestions,
- juxtaposition for clarity,
- simplicity through familiarity,
- sufficient visibility for validation,
- snoozability of suggestions.



The image shows two screenshots of an IDE interface illustrating design principles for autocomplete. Both screenshots feature a vertical color-coded bar on the left (green for class members, yellow for methods, and grey for comments) and a code editor with line numbers (8 to 17). A cursor is positioned at the end of the constructor call in both cases. A tooltip window labeled 'Tab accept | ...' is open, providing additional context or options for the current action.

**(a)**

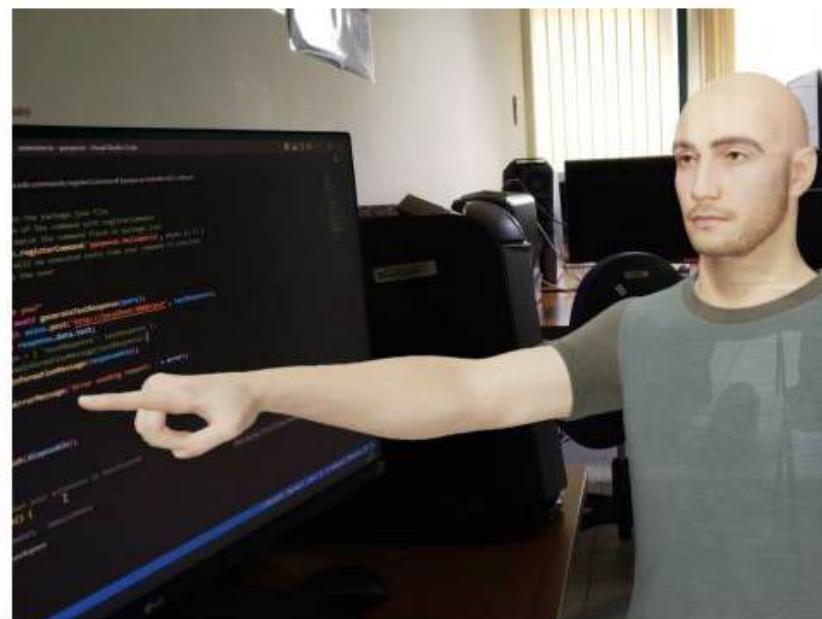
```
8   class Obstacle
9
10  {
11      public ObstacleType type { get; set; }
12      public int XPos { get; set; }
13      public int YPos { get; set; }
14      public double XVelocity { get; set; }
15      public double YVelocity { get; set; }
16
17      public Obstacle (ObstacleType type, int xPos, int yPos, double xVelocity, double yVelocity)
18      {
19          Type = type;
20          XPos = xPos;
21          XVelocity = xVelocity;
22          YVelocity = yVelocity;
23      }
24  }
```

**(b)**

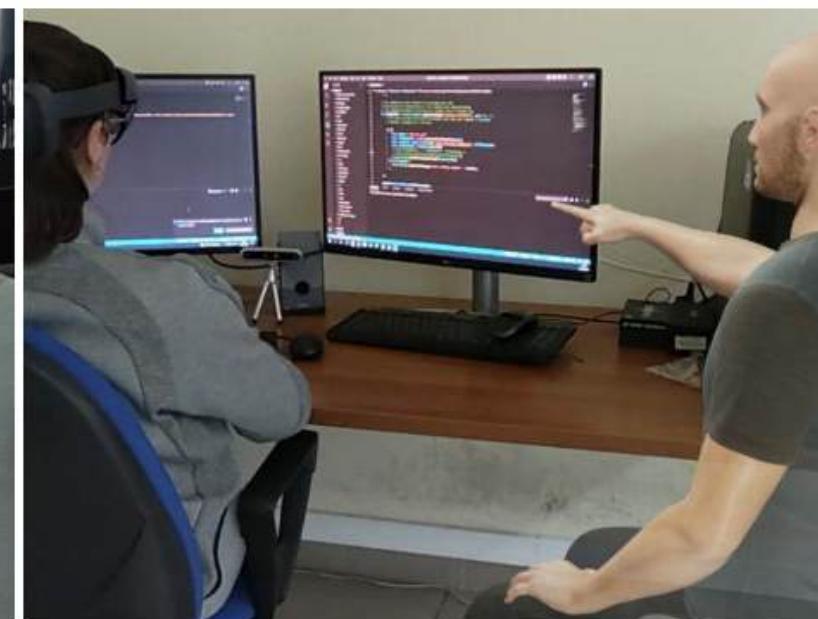
```
8   class Obstacle
9
10  {
11      public ObstacleType type { get; set; }
12      public int XPos { get; set; }
13      public int YPos { get; set; }
14      public double XVelocity { get; set; }
15      public double YVelocity { get; set; }
16      private bool IsValid { get; set; }
17
18      public Obstacle (ObstacleType type, int xPos, int yPos, double xVelocity, double yVelocity, bool isValid)
19      {
20          Type = type;
21          XPos = xPos;
22          XVelocity = xVelocity;
23          YVelocity = yVelocity;
24          IsValid = isValid;
25      }
26  }
```

# AI as a pair programmer

to assist novices in their education, revealing a generally high level of acceptance and usefulness



(a)



(b)

Manfredi et.al., 2023

A screenshot of a software interface for Eclipse Application. On the left, there is an IDE window showing Java code for a Tic Tac Toe game. The code includes imports, class definitions, and test methods. On the right, there is a "Session Chat" window showing a conversation between a user named "Wizard" and an AI named "Robe". The AI is responding to the user's comments about testing and accommodating multiple players. There are also two small avatars of women displayed above the chat window.

Robe and Kuttal, 2022

**Conversational and explainable**  
AI might support user productivity and  
trust while, in some cases,  
causing overreliance

# In the Design of HAX, we propose

chat-based interaction with LLMs may not always be the most effective approach, and different tasks may require alternative methods.

- Non-chat format of features
- Predict actions and suggest them proactively
- Text-to-project
- Multiagency

# Interactive & Aligned IDE

AI4SE @ TU Delft — Track 3.

Ultimate Goal: Increase usability and accessibility of AI tools in IDEs

Key Results: Human-AI alignment inside IDE.

Novel LLM features tailored for developer workflow.

Explainable and trustworthy agent.



Sergey Titov



Maliheh Izadi

# Impact

- Understanding how programmers use and perceive AI assistants
- Identifying challenges and creating a taxonomy of activities
- Assessing users' productivity and trust

# Impact of AI tools

## ↑ Productivity ↑

Only with explainable output and readable code.

## ↑ Education ↑

AI tools can positively influence programming education by facilitating learning and boosting motivation.

## ⬇️ Code quality ⬇️

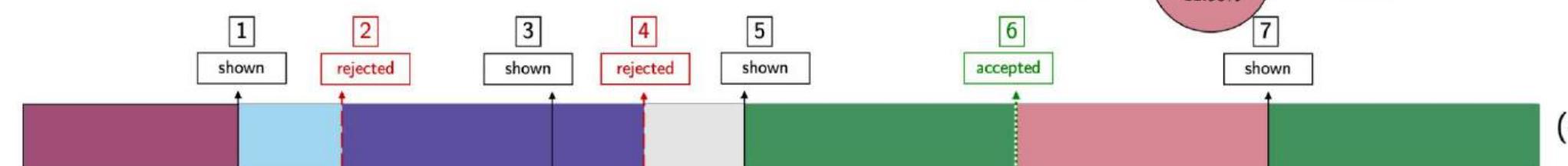
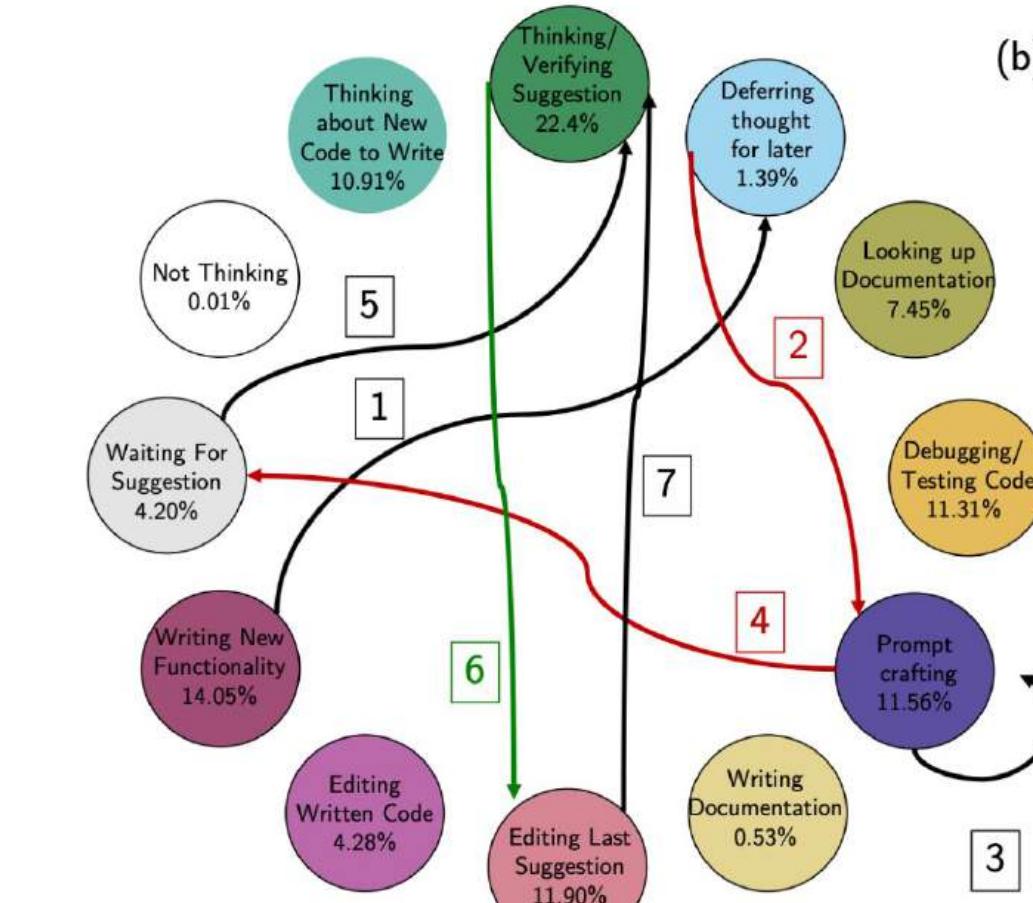
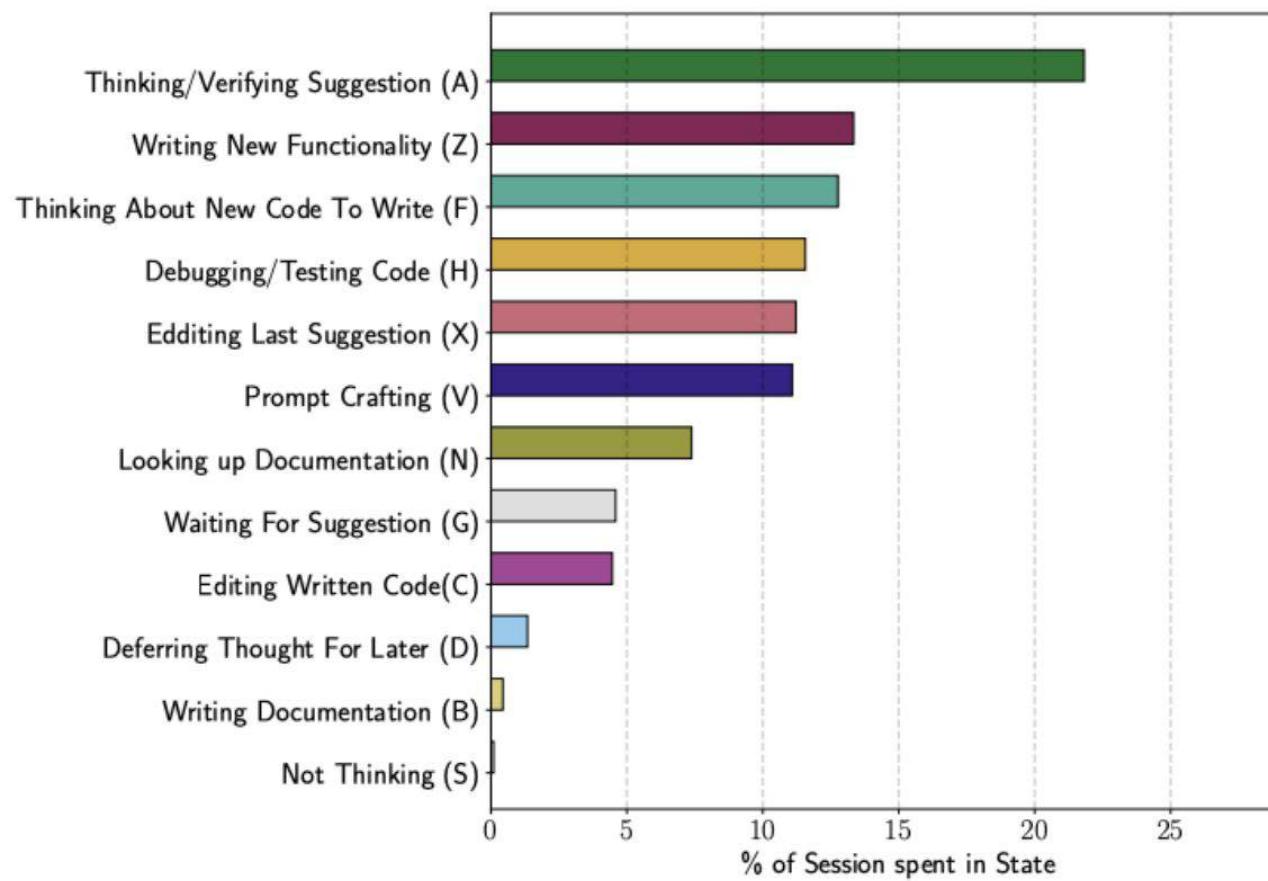
Developers sometimes struggle to receive from AI the outputs that would align with their requirements and expectations.

## ⬇️ Discretion ⬇️

Challenges such as over-reliance and the need for caution.

# Impact of AI tools

Alter the traditional workflow



**Successful AI integration involves balancing the benefits of acceleration with the need for careful exploration and validation.**

# In the Impact of HAX, we propose

to focus on trust and explainability.

- Inspiration from NLP domain:
  - Highlighting impactful tokens
  - Approximating model uncertainty
  - Providing clear and transparent context of model's output
- Form accurate expectations of AI performance
- Control AI by setting and adjusting preferences

# Benefits of AI Assistant

Ultimate Goal:

Assess how our AI assistant impacts developers' satisfaction and productivity.

Key results:

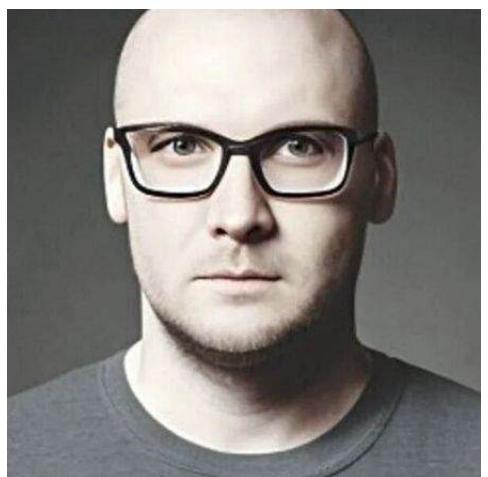
Data for marketing campaign and decision making.



Nadezhda  
Novikovskaya



Natalia  
Pritykovskaya



Anton Arhipov

# AI in SDLC

With the University of California, Irvine

Ultimate Goal: Investigate developers' patterns of use of AI tools at different software development stages.



Yaroslav Golubev

Key results: Reasons for not using AI in certain steps of SDLC,  
Users' suggestions for improvements in those areas.



Iftekhar Ahmed

# Longitudinal Analysis of AI Impact

With the University of California, Irvine

Ultimate Goal: Foresight and innovation support

Key results: Derived trends and changes in AI-affected metrics of  
in-IDE activity



Iftekhar Ahmed

# Quality

Evaluating various aspects of AI assistants' functionality:

- Correctness
- Understandability
- Security

# Quality findings

- **Correctness** – while AI can provide relevant solutions and suggestions for trivial tasks, they still might be erroneous and require the user's ability to notice and correct them.
- **Comprehensibility** – AI generally produce understandable code that might be even less complex than one written by human, but there is room for improvement.
- **Security** – risks might depend on the model used for the assistance foundation. For instance, 40% of generated programs on C appeared to be vulnerable.

**Variations in the core model affect AI assistance effectiveness and utility, influencing user productivity by enhancing suggestion accuracy and timing.**

# In the Quality of HAX, we propose to focus on readability.

- How do people discriminate between readable and unreadable code?
- General properties of code that people prefer
- Groups of programmers with a common view

# Readability of AI-generated Code

Ultimate Goal: Optimize LLMs for generating more readable code, enabling developers to quickly understand outputs and save time.

Key Results: Definition of Code Readability and its features  
Model of Code Readability,  
Code Readability Adjusted LLM.



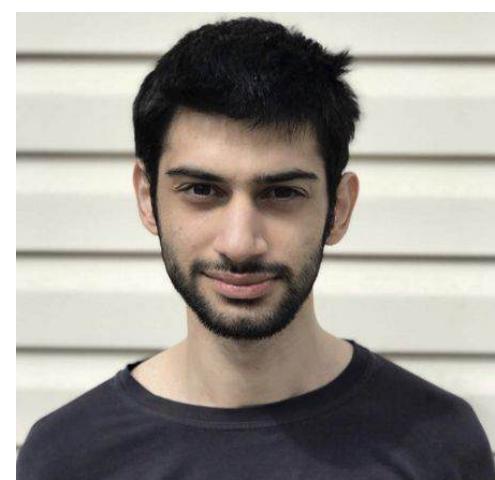
Olga Lvova



Anastasiia Serova



Sergey Titov



Farid Bagirov

# The Design Space

Key areas of HAX where improvements can be made derived from 35 interviews with different groups of users, including adopters, churned users, and non-users of AI in IDEs

[Paper](#)

Technology Improvements									
Accessible Technology	Proactive Technology	Reliable Technology	Up-to-date Technology	Ethical Technology	Private Technology	Secure Technology	Aligned Technology	Context-aware Technology	
Craft versatile technologies that are both affordable and adaptable to various constraints	Enhance efficiency and foresight by reducing users' manual effort and preemptively solving issues	Ensure the robustness and accuracy of AI systems and increase users' trust in AI solutions	Ensure AI systems remain current and competitive by integrating the latest advancements and updates	Balance innovation with conscientious practices, fostering technology that is both effective and morally sound	Focus on safeguarding user privacy and enhancing data security	Fortify AI systems against vulnerabilities and ensuring their integrity	Ensure that AI-generated solutions are coherent, non-redundant and consistent with codebase	Enchase AI's ability to understand and integrate various levels of context	
Separate feature purchase, Offline assistance	Proactive optimization, AI performing actions	Output clarifications, Providing sources	SOTA tooling, Up-to-date data	Rules and restrictions	On-premise AI	Annotating AI code	Library suggestions, Better commit messages	Manually set context, Project-wide context awareness	

Technology Interaction		Technology Alignment		Streamlining Skills Building			
Non-Interruptive Technology	Internal System Integration	Adjustable Technology	Streamlining Workflow	User Education	Learning Support		
Enhance user experience by reducing interruptions, and addressing interaction issues to maintain a smooth and efficient workflow	Enhance the cohesion and interaction of AI systems within development environments	Enhance functionality through configurable AI solutions that adapt to diverse requirements	Create a cohesive work environment that supports organization and facilitates daily tasks	Equip users with the knowledge and tools needed to effectively interact with AI systems	Support the acquisition of new concepts and skills, making the learning process more effective and empowering for developers		
Stable AI functionality, Fixing frequent logouts	Anywhere-in-IDE AI access, In-line code explanations	Configurable AI, AI action customization	Daily ecosystem summarizer, Automation of repetitive tasks	Shareable Prompt Library	Step-by-step output revealing		

Streamlining Programming Tasks							
AI Support in SDLC	Navigating the Codebase	AI in Ideation	AI in Implementation	AI in Testing	AI in Maintenance	Code Optimization	External System Integration
Aim for universal adoption to provide consistent programming assistance, handle tedious and well-known tasks, and improve the end product	Facilitate a deeper comprehension of code and project structure to aid in navigating and understanding codebase	Enhance the creative and brainstorming process by providing more effective ways to generate and explore ideas	Support development workflows streamlining the process from concept to implementation	Enhance test generation and ensure robust, thorough evaluation of software systems	Streamline and enhance maintenance tasks to optimize quality to ensure high-performance and well-maintained codebases	Enhance code efficiency and quality to ensure high-performance and well-maintained codebases	Incorporate AI into broader development workflows and external systems, enhancing overall development efficiency and connectivity
Universal programming assistance	Onboarding support, Database analytics	Creative AI	Multi-file generation, Project set-up assistance	Coverage report, UI testing	Repository management, Dependency management	Code evaluation, Code review support	AI in CI/CD Pipelines, Out-IDE AI access



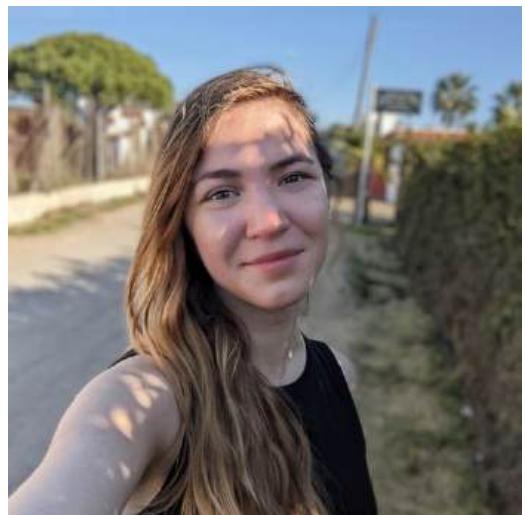
# Come collaborate



**ilia.zaharov**  
**@jetbrains.com**



**dariia.karaeva**  
**@jetbrains.com**



**ekaterina.koshchenko**  
**@jetbrains.com**



**agnia.sergeyuk**  
**@jetbrains.com**



**linkedin**

# Reading

Something I would recommend to read:

- In-IDE Human-AI Experience in the Era of Large Language Models; A Literature Review, 2024
- Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming, 2024
- Toward General Design Principles for Generative AI Applications, 2023
- Grounded Copilot: How Programmers Interact with Code-Generating Models, 2023
- Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models, 2022
- Keep an eye on CHI conference, Microsoft Research Blog, GitHub Blog and JetBrains Blog

# **CSE2115 Software Engineering Methods**

**Recap and Q&A**

**Taico Aerts & Arie van Deursen**

# What did we (attempt to) teach you?

- How to be a software engineer
- How to create the right software
- How to maintain and measure quality of software, and it's importance at larger scales
- How to work together in software development teams
- How to make use of new and emerging technologies to become a better software engineer
- How to make sure your software product is going to be a success

# Vevox



vevox

- [vevox.app](http://vevox.app)
- 129-948-017

# **W1 - Requirements Engineering**



# Questions?

- Why is it important to identify all stakeholders?
- What different ways exist to extract requirements from stakeholders?
- What quality attributes should requirements adhere to?
- How should we prioritize requirements?
- How can we show/discuss requirements with stakeholders?



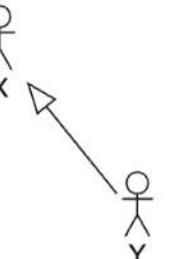
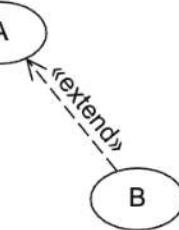
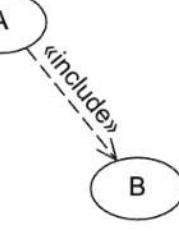
# **w1/2 – UML and C4**



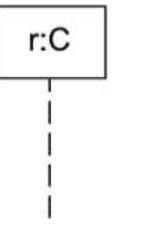
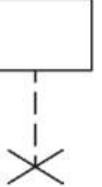
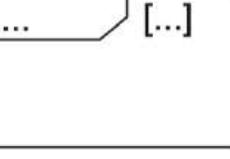
# Which diagrams? How detailed?

- Use Case // Sequence // Class // Component Diagram
  - Read, Create
- Activity Diagram
  - Read

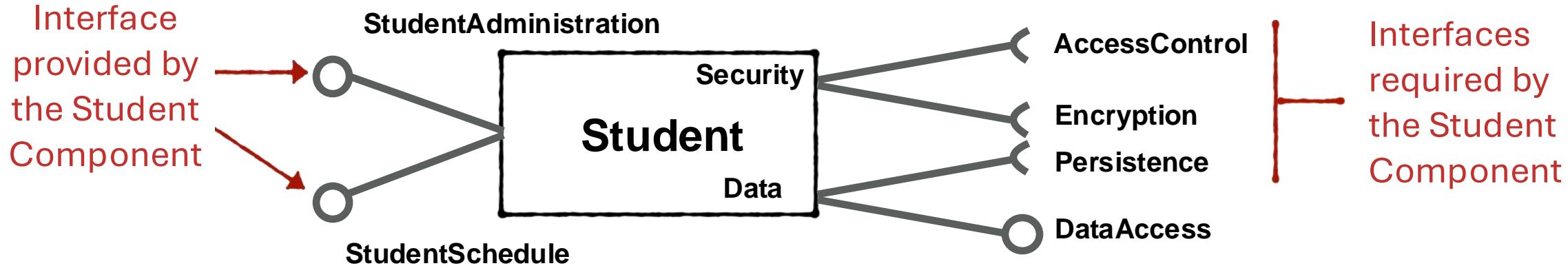
# Use Case Diagrams

Use case		Unit of functionality of the system
Actor	 or 	Role of the users of the system
Association		X participates in the execution of A
Generalization (actor)		Y inherits from X; Y participates in all use cases in which X participates
Extend relationship		B extends A: optional incorporation of use case B into use case A
Include relationship		A includes B: required incorporation of use case B into use case A

# Sequence Diagrams

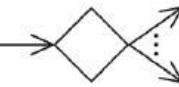
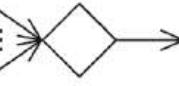
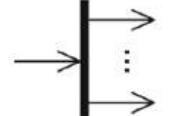
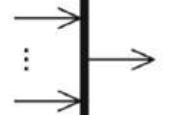
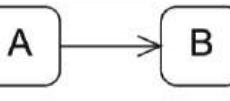
Name	Notation	Description
Lifeline		Interaction partners involved in the communication
Destruction event		Time at which an interaction partner ceases to exist
Combined fragment		Control constructs
Synchronous message		Sender waits for a response message
Response message		Response to a synchronous message

# Component Diagram

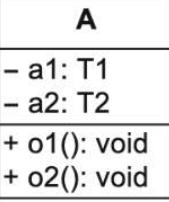
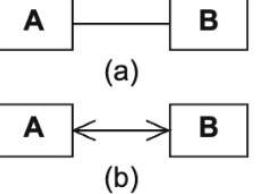
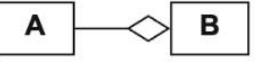
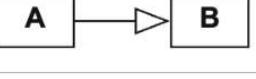


- The **lollipop notation** denotes the main interface of each component (rectangle)
  - The **sockets** (-c) denote the interfaces the component requires (**input**)
  - The **lollipops** (-o) denote the interfaces a given component provides (**output**)

# Activity Diagrams

Name	Notation	Description
Action node	Action	Actions are atomic, i.e., they cannot be broken down further
Initial node	●	Start of the execution of an activity
Activity final node	○	End of ALL execution paths of an activity
Decision node		Splitting of one execution path into two or more alternative execution paths
Merge node		Merging of two or more alternative execution paths into one execution path
Parallelization node		Splitting of one execution path into two or more concurrent execution paths
Synchronization node		Merging of two or more concurrent execution paths into one execution path
Edge		Connection between the nodes of an activity
Object node	Object	Contains data and objects that are created, changed, and read

# Class Diagrams

Name	Notation	Description
Class		Description of the structure and behavior of a set of objects
Association		Relationship between classes: navigability unspecified (a), navigable in both directions (b)
Strong aggregation = composition		Existence-dependent parts-whole relationship (A is part of B; if B is deleted, related instances of A are also deleted)
Shared aggregation		Parts-whole relationship (A is part of B; if B is deleted, related instances of A need not be deleted)
Generalization		Extends relationship (A extends B)
Implements		Implements relationship (A implements B)

--|> and ..|> in PlantUML!

## Visibility

- + public
- private
- # protected
- ~ package

## Attributes

visibility name : Type

## Methods

visibility name() : Type  
Visibility name(Type, Type) : Type  
visibility name(param: Type) : Type

# Example Questions

- Given this diagram, explain ...
- Given this diagram, is it possible for ...?
- Given this diagram, which of the following are true?
- Given this context, create the corresponding ... diagram

# Composition vs Shared Aggregation

- Composition (UML)
  - A is part of B. If B is deleted, A must also be deleted
  - Example: Faculty - University
- Share Aggregation (UML)
  - A is part of B. If B is deleted, A continues to exist
  - Example: Student – University
- Aggregation (DDD)
  - Completely different, later in this lecture
- Normal Association (UML)
  - Used to indicate many associations
    - A sends messages to B
    - A creates instances of B
    - A has attribute of type B (or collection)
    - A receives messages with B in them
- Usually composition/aggregation implies normal association

# Spec Tests on UML Diagrams

- Checked during the exam
  - Basic properties (block/class/... with correct name)
- Used by us to grade the question
  - Arrows between blocks of particular types
  - Fields/methods with visibility, name, type
  - Fuzzy matching on names or annotations
  - ....
  - → Instead of human grading, the computer grades like a human would (as much as possible without AI)
- We inspect submissions, and create spec tests for grading after the exam

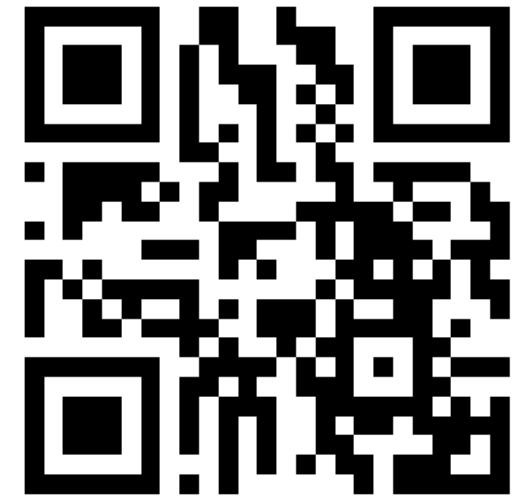
# PlantUML Notation

- Follow “UML @ Classroom” arrows
- PlantUML
  - PlantUML Class Diagram
    - Use class, abstract class, interface, enum, ...
    - **Don't** use extends or implements, it gives incorrect arrows
  - PlantUML Sequence Diagram
    - Use default arrows for sequence diagram (-> and <--)
    - Activate/deactivate to indicate lifetime
    - Destroy to destroy objects

# Quick Questions

- Example Sequence/Component Diagram in WL?
  - No promises, but I will ask the TAs to work on one if lab is quiet
- Sequence diagram, difference between deactivated and destroyed?
  - Is the component actively waiting (active) or will it receive an event at a later time (deactivated). Destroyed is for explicit destruction.

# Questions?



# **W2 - Software Architecture**



# Event-driven vs Publish-Subscribe

- Very similar
- Publish-Subscribe pattern is a (the) way to create an event-driver architecture

# Publish-Subscribe vs Observer

- Publish-Subscribe is about the architecture of (a part of) the system (consisting of multiple independent components/services!)
- Observer is a design pattern you can use within a single component/service

# Quick Questions

- How in-depth should we know architectural patterns?
  - Decently: identify patterns, advise on which pattern(s) to use, explain properties of pattern

# Questions?

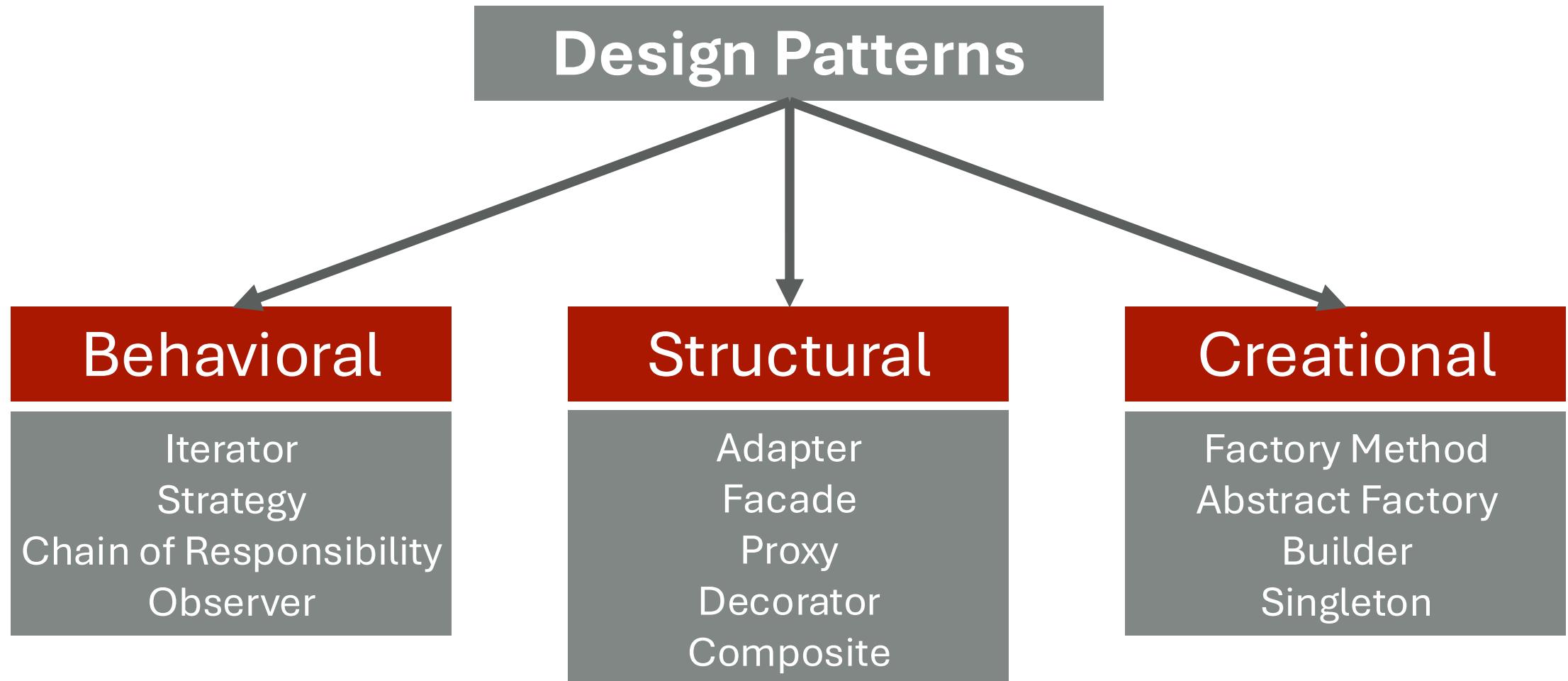
- Why would one consider using the C4 model over UML?
- Which architectural pattern(s) is/are best suited for a particular context?



# **W2 – Design Patterns**



# Which design patterns should I know?



# Which Design Pattern to Apply?

## Scenario:

Design an online **investment** system for a large international bank allowing users to create investments.

Users can create various **types of investments** (e.g., bonds) across **different markets** (e.g., US, EU), each with specific attributes like interest rate, risk level, and duration.



The system should accommodate multiple combinations of markets and investment types while minimizing code repetition. Additionally, the system should remain open for future expansion to include more markets and investment types.



# Answer

- Most applicable: **abstract factory**
  - Family of objects per market
    - EU Bond, EU Stock
    - US Bond, US Stock
  - The market serves as an instantiation of the abstract factory
    - EUMarket can create EU Bonds, Stocks, etc.
    - USMarket can create US Bonds, Stocks, etc.

# Reverse Exercise

- Let's consider the **Strategy pattern**.
- Describe a real-world application or system where the **Strategy pattern** can be employed. Illustrate a scenario highlighting the benefits and advantages of using the Strategy pattern in that context.



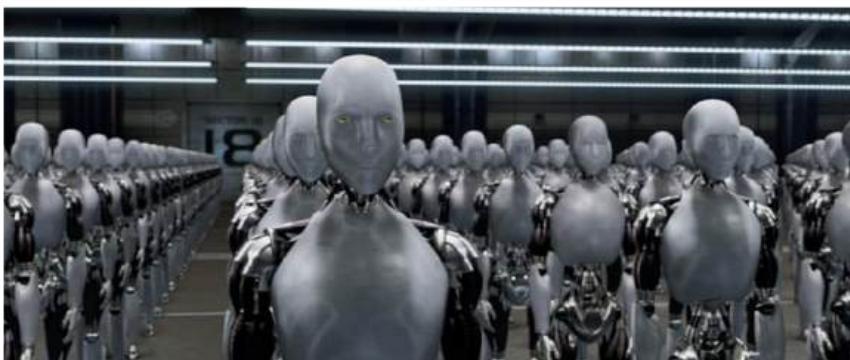
# Answer

- (Example from lecture): Google Maps
  - Strategy on route planning
  - Choose between Walking, Bike, Car, Public Transport
  - Each is a Route Planning Strategy, which influences how roads/routes are considered

# Which Design Pattern to Apply?

## Scenario:

Develop an online system for building **robots** with customizable hardware (**sensors**) and software (**AI engines**). Customers can incrementally select different types of sensors (e.g., cameras, tactile sensors, microphones) and their corresponding AI engines (e.g., deep-learning recognition, regression models, text-to-speech) to build customized robots.



“Io Robot” from Wikipedia

The system should allow step-by-step **assembly of robots** with **selectable sensor and AI engine combinations** while accommodating future expansions for additional sensor and AI options.

# Answer

- Most applicable: Builder Pattern
  - Step-by-step construction
  - Many combinations
  - Ability to add additional attributes later (with default values) which will not require code changes

# Quick Questions

- Will we need to implement design patterns on the exam
  - No, but given a scenario you will need to be able to draw a class diagram of your code
- How in-depth should we know design patterns
  - Quite: Advice on which pattern, show how to apply pattern to scenario, indicate pros/cons of pattern, etc.
- Why are some patterns underlined in overview?
  - Underlined = not in book, see slides/refactoring.guru
- Flyweight?
  - You don't need to know it



# **W3 - Open Source**



# Questions?

- What are the trade-offs of different licenses (free vs open source)?
- Is it possible to set up a business model around selling open source software? How?



# **W3 - Domain Driven Design**



# Value Objects vs Entities

- Difference is determined by identity!
- Value Object
  - Immutable
  - Distinguished by attributes, not by identity
  - Example: address - consists of multiple attributes, identical attributes = same address
- Entity
  - Can be mutable
  - Has a unique identity, even if all attributes are equal (besides id)
  - Example: person – people with same name exist, but are still different people

# Aggregates

- Bubble
  - Every bubble can be worked on independently
  - Every bubble functions as a single unit
- Do **not** think about it from a database perspective

# Quick Questions

- Can something be neither value object nor entity?
  - ~Yes, e.g. a service or repository



# **W4 – Agility at Scale**



# How detailed should I know things?

- Check practice exam for good examples.
- Decently detailed, know the differences between each framework to be able to advise others on what to use and why
- You don't need to know nitty gritty details like from Lecture Preparation questions.
  - Those questions specifically were to have you experience how some frameworks go too far

# Questions?



**W4 – SE at Bol.com**



# What types of questions about this?

- Example: Large companies (like Bol.com) do various checks before deploying new code into production. Mention one such check and explain briefly how it is used.

# What types of questions about this?

- Large companies (like Bol.com) do various checks before deploying new code into production. Mention one such check and explain briefly how it is used.
  - A/B testing: first roll out to small sample of customers, measure impact on products sold and only deploy further if positive

# Questions?



# **CSE2115 Software Engineering Methods**

**Recap and Q&A**

**Taico Aerts & Arie van Deursen**

# **Guessing correction**

# Guessing Correction

- Single-answer MCQ
  - Correct → 10
  - Blank or Incorrect →  $- \frac{10}{\# \text{options}-1}$
- Example: Ten 4-option single-answer MCQs
  - 7 correct, 3 incorrect/unanswered
  - Grade =  $\frac{10 \cdot 7 + (-3.3) \cdot 3}{10} = 6$

# Guessing Correction

- We are adjusting your expected score when you know **NOTHING** about SEM from 2.5 to 0.
- E.g. 40 MC questions, 4 options
- $P = 25\%$  to guess question right
  - $\rightarrow 10 \text{ right}, 30 \text{ wrong}$
- $$\frac{10 \cdot 10 + \left(-3\frac{1}{3}\right) \cdot 30}{40} = 0$$

WebLab applies guess correction at the question level. For both unanswered and incorrectly answered questions, it subtracts points to account for random guessing, ensuring that blind guessing would result in an expected score of zero. The deduction is based on the probability of guessing a correct answer, depending on the number of choices and correct answers.

**WebLab does not distinguish between unanswered and wrongly answered questions.**

This is done for several reasons:

- Students would spend considerable time before, during, and after the exam making tactical decisions about whether to answer questions;
- Students with the same level of knowledge may differ significantly in their willingness to attempt an answer;
- The decision to answer or skip a question is not always made rationally;
- Differences in guessing behavior can also affect students' final grades.

By treating unanswered and wrongly answered questions equally, students can focus on answering the questions without worrying about whether they should leave a question unanswered.

# Winning Strategy

- Choose the answer you believe is best
- Answer all questions, always
- Score will reflect your knowledge
  - Eliminating options is a benefit, your odds of guessing correct go up

# **W5 – Software Metrics & Technical Debt**



# Cohesion vs Coupling

- Cohesion
  - How well do methods in a module belong together
  - You want this to be high (relevant code in one place)
- Coupling
  - How many other classes do we depend on (e.g. through method calls)
  - You want this to be low (single, isolated task)

# Cyclomatic Complexity

- Without Graph?
  - Possible, count decision points + 1
- What are decision points?
  - All “checks” (if, for, while)
- >> *Code Example in IDE* <<

# Cyclomatic Complexity

- Control flow graph
  - Node is a sequence of code ending in a decision (split) or a point that other places go (back) to
  - >> *Example in IDE* <<

# CC

```
2 public class PrimeNumber {  
3     /**  
4      * This method computes the number of prime numbers smaller than  
5      * or equal to a given positive number  
6      *  
7      * @param number upper bound  
8      * @return number of prime numbers <= {@code number}  
9     */  
10    public static int countPrimes(int number) {  
11        int results = 0;  
12        if (number > 1) {  
13            if (number < 4)  
14                results = number - 1;  
15            else {  
16                BitSet set = new BitSet();  
17                int s = (int) Math.sqrt(number);  
18                int counter = number;  
19                for (int p = 2; p <= s; p++) {  
20                    if (!set.get(p)) {  
21                        for (int q = 2; (p * q) <= number; q++) {  
22                            if (!set.get(p * q)) {  
23                                counter--;  
24                                set.set(p * q);  
25                            }  
26                        }  
27                    }  
28                }  
29                results = counter - 1;  
30            }  
31        }  
32        return results;  
33    }  
34 }
```

- 6 decision points
- CC = 6 + 1 = 7

What is the Cyclomatic Complexity?



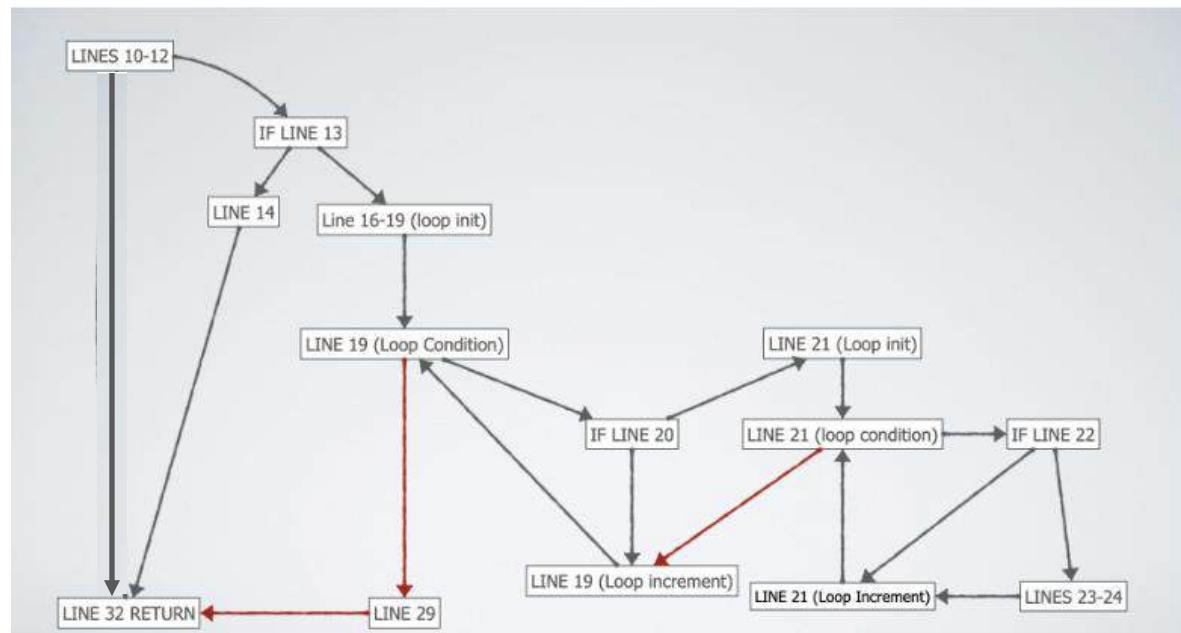
```

2 public class PrimeNumber {
3     /**
4      * This method computes the number of prime numbers smaller than
5      * or equal to a given positive number
6      *
7      * @param number upper bound
8      * @return number of prime numbers <= {@code number}
9     */
10    public static int countPrimes(int number) {
11        int results = 0;
12        if (number > 1) {
13            if (number < 4)
14                results = number - 1;
15            else {
16                BitSet set = new BitSet();
17                int s = (int) Math.sqrt(number);
18                int counter = number;
19                for (int p = 2; p <= s; p++) {
20                    if (!set.get(p)) {
21                        for (int q = 2; (p * q) <= number; q++) {
22                            if (!set.get(p * q)) {
23                                counter--;
24                                set.set(p * q);
25                            }
26                        }
27                    }
28                    results = counter - 1;
29                }
30            }
31        }
32        return results;
33    }
34 }

```

# CFG

- 14 nodes
- 19 edges
- CC =  $19 - 14 + 2 = 7$



# LCOM (Lack of Cohesion of Methods)

- LCOM = Pairs of methods with no shared attributes – pairs of methods with shared attributes
- Minimum value is zero

# Quick Questions

- Know formulas?
  - Yes: CC, LCOM, CM



# **W5 – Sustainable Software Engineering**



# Quick Questions

- Python Programming on Exam?
  - No
- Manifesto from BS or from WL?
  - BS reading material is leading



# **W6 – Software at Scale**



# Docker vs VM

- Docker
  - you run a normal application on your system, but isolate it more than normal from other applications
- Virtual Machine
  - you emulate an entire computer (i.e. you emulate the hardware!)

# Questions?

- Why can it make sense to break up your application into multiple smaller services?
- Why is it difficult to reproduce production issues locally?
- When working at a large scale, why is it important to invest in quality and monitoring?
- How can you scale up applications to deal with a larger volume of users?



# **W7 – AI for Software Engineering**



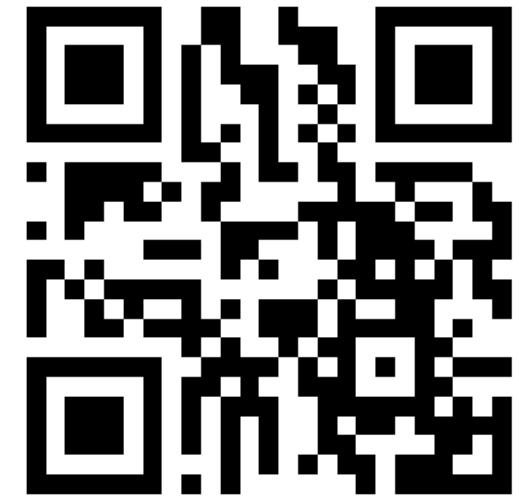
# Questions on this topic?

- This is not an AI course, no details of AI models themselves are required for the exam.

# Example Questions

- Why is it not always better to use a large AI model for AI assistance in an IDE?
  - Large models can suffer from increased latency, or are too big to run locally (cloud is not always an option due to copyright)
- Should you treat code the same as natural language? Why (not)?
  - Code has a stricter structure, and has lots of internal dependencies and connections. It can also be executed, this means it behaves a bit differently and it can be beneficial to use different models.

# Questions?



# General Questions



# Guessing Correction

- Single-answer MCQ
  - Correct → 10
  - Blank or Incorrect →  $- \frac{10}{\# \text{options}-1}$
- Example: Ten 4-option single-answer MCQs
  - 7 correct, 3 incorrect/unanswered
  - Grade =  $\frac{10 \cdot 7 + (-3.3) \cdot 3}{10} = 6$

WebLab applies guess correction at the question level. For both unanswered and incorrectly answered questions, it subtracts points to account for random guessing, ensuring that blind guessing would result in an expected score of zero. The deduction is based on the probability of guessing a correct answer, depending on the number of choices and correct answers.

**WebLab does not distinguish between unanswered and wrongly answered questions.**

This is done for several reasons:

- Students would spend considerable time before, during, and after the exam making tactical decisions about whether to answer questions;
- Students with the same level of knowledge may differ significantly in their willingness to attempt an answer;
- The decision to answer or skip a question is not always made rationally;
- Differences in guessing behavior can also affect students' final grades.

By treating unanswered and wrongly answered questions equally, students can focus on answering the questions without worrying about whether they should leave a question unanswered.

# Select-all-that-apply MCQ

- Every correct option adds points, every incorrect option removes points.
  - $\frac{10}{\# \text{correct options}} \cdot (\#\text{ticked}_{\text{correct}} - \#\text{ticked}_{\text{incorrect}})$
  - Minimum grade is 0 (no guessing correction)
- For 6 option question (a-f) with a+c correct:
  - ac=10, a=5, c=5
  - abc=5, ab=0, bc=0, b=0, bdef=0

# How to prepare?

- Practice Exam!
- Read up on suggested reading / preparation material
- Do lab exercises and check answers
  - Ask questions to TAs!
- Try old exams
  - Different structure and partially different content!
  - Still useful to check whether you have the knowledge

# Quick Questions

- Lab week 9
  - Yes, for questions on Practice Exam
- Only slides? Or also reading material?
  - All reading materials are exam material. See BS weekly task lists
- Cheat Sheet on Exam?
  - No
- Extra practice exams?
  - No, but TAs will gladly (try to) come up with new questions for you on the spot in labs

# Any Other Questions?

# Course Feedback

# Course Feedback

- Many things changed in SEM this year!
- Let us know what you think
- Share ideas for how to improve
- [evasys-survey.tudelft.nl/  
evasys/  
online.php?p=PEM4H](https://evasys-survey.tudelft.nl/evasys/online.php?p=PEM4H)



# **Good Luck!**

# CSE2115 Software Engineering Methods

Taico Aerts & Arie van Deursen