Ca' Foscari University of Venice

News

- The schedule has been finalized:
  - in person lectures always on Thursday at 2pm in Aula B except
    - Nov. 14th in Lab. 3
    - Tuesday Nov. 19th at 8:45 in Aula C
  - Nov. 28th invited lecture by Unox
    - https://www.unox.com/us_us/
  - Dec. 5th invited lecture by Gianluca Caiazza
    - Zamperla project
    - https://www.zamperla.com/

Ca' Foscari University of Venice

Project

- 3 group tasks during the course (two thirds of the grade)
  - 2 small ones at the beginning
  - 1 big task at the end implementing the architecture
- Topic: hospital management, e.g.,
  - Emergencies
  - Patient medical records
  - Schedule of the doctors, exams, …
  - Hospital ward (https://en.wikipedia.org/wiki/Hospital#Departments_or_wards)
  - And whatever you might think…

Ca' Foscari University of Venice

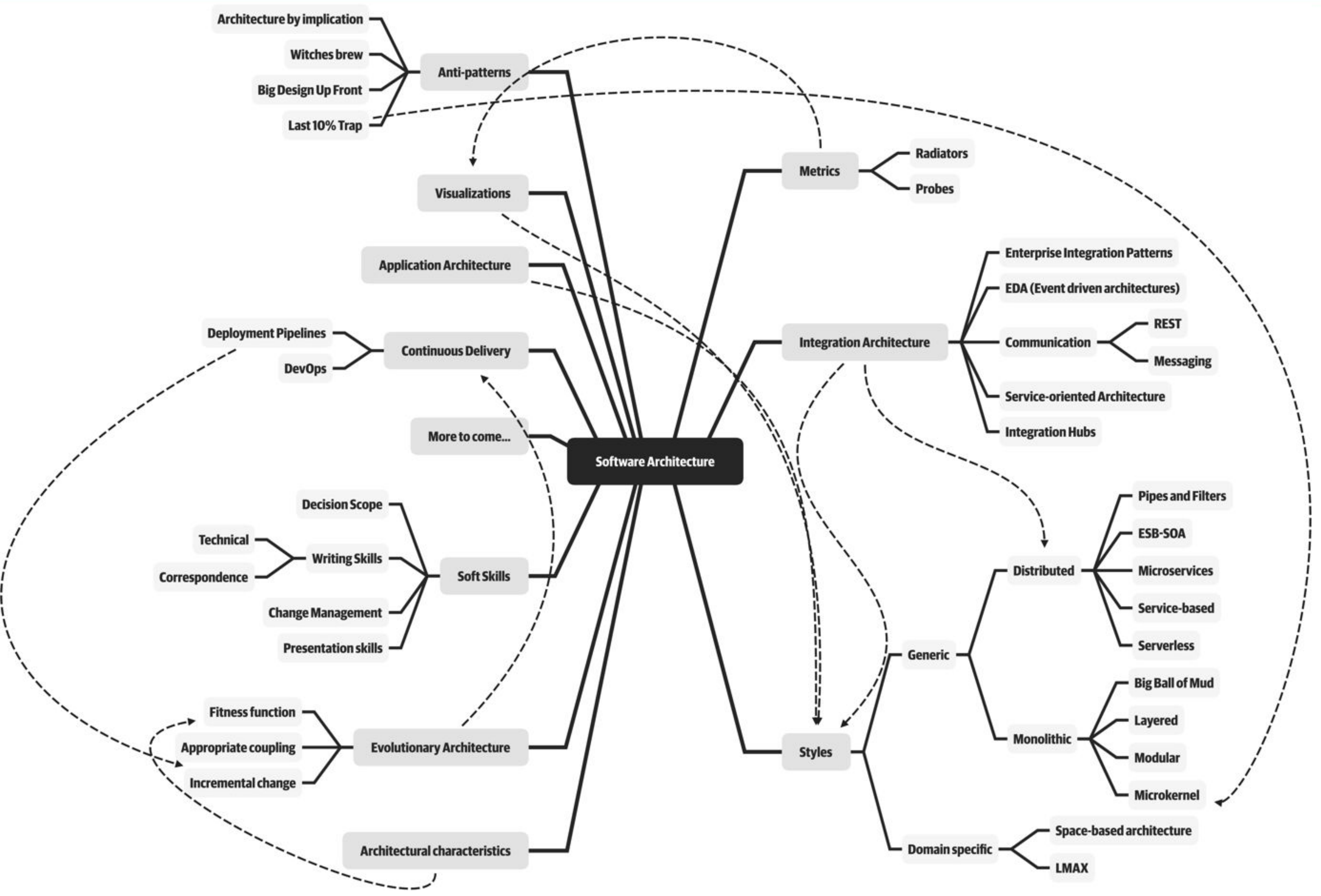
An introduction to software architect[ure] [Concepts]

Software architectures

Pietro Ferrara
pietro.ferrara@unive.it

Ca' Foscari University of Venice

Skills

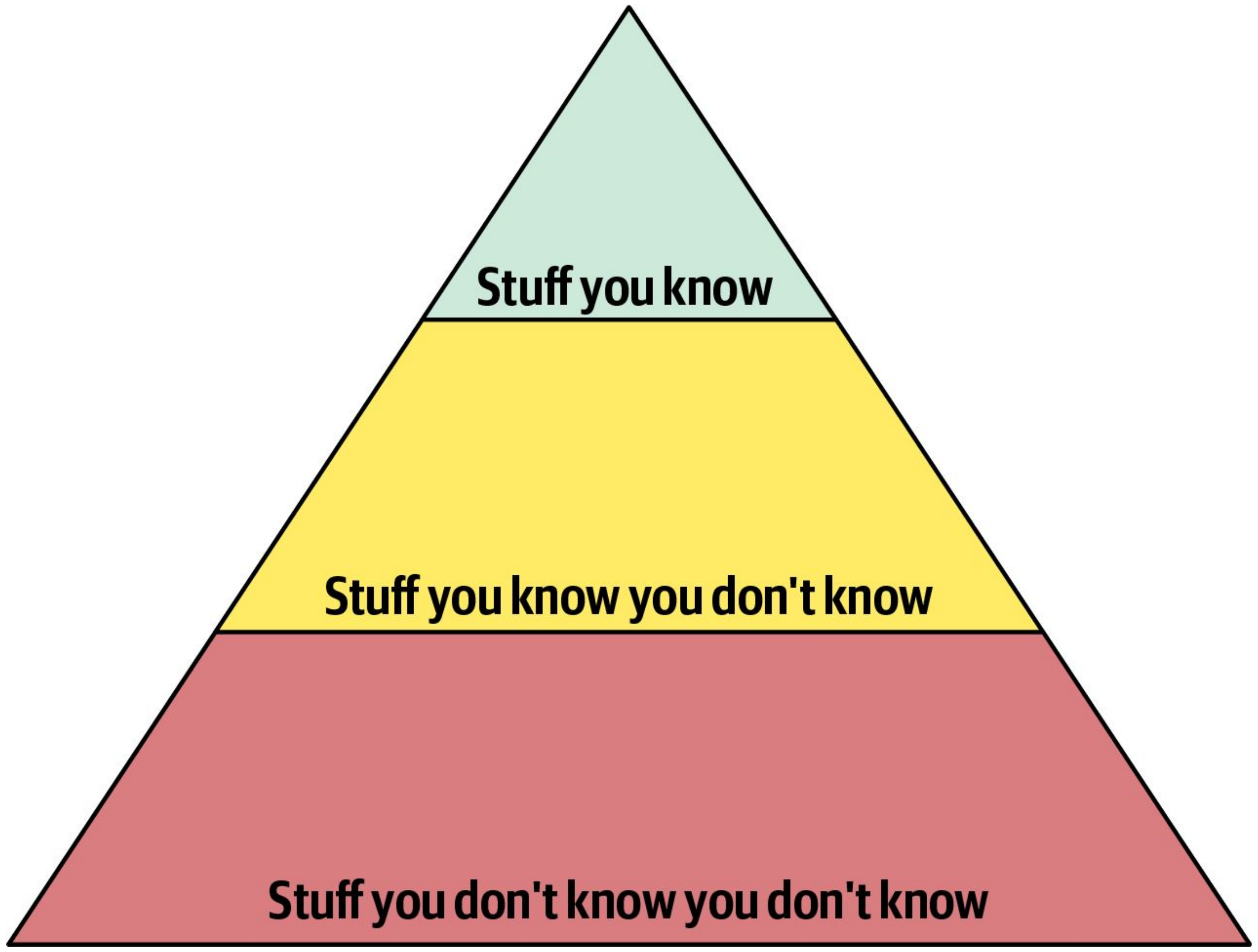


- Technical low level knowledge
- Technical high level knowledge
- System knowledge
- Soft skills

- Nobody can have all those skills together!!!

Ca' Foscari University of Venice

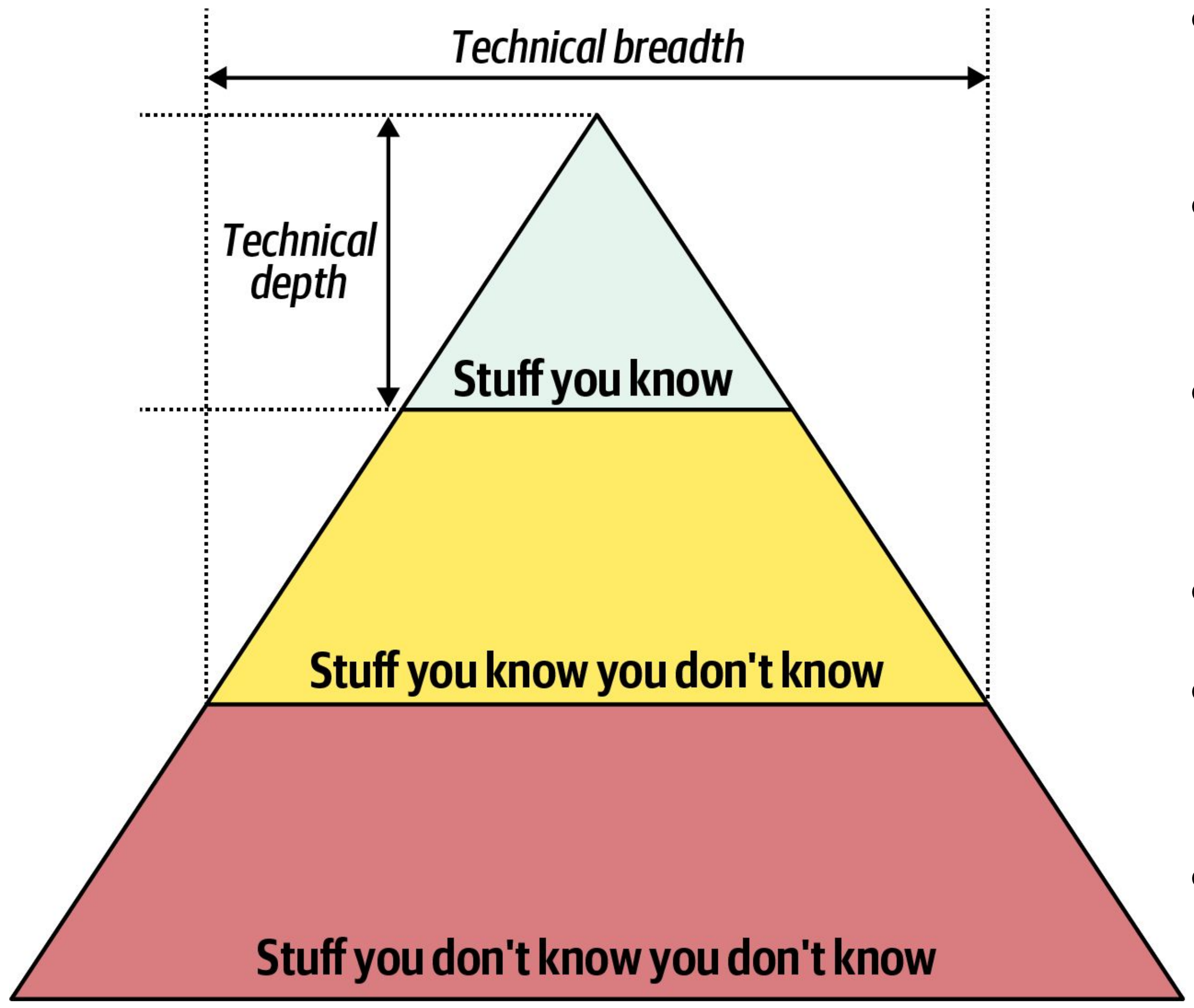
Technical knowledge


Stuff you know
Stuff you know you don't know
Stuff you don't know you don't know

- Technical knowledge is extremely wide
- Much more stuff you don’t know than you know
- Even worse: you ignore even the existence of the most part of technologies
- Should you be ashamed?
  - No!!!
  - We are all on the same boat

Ca' Foscari University of Venice

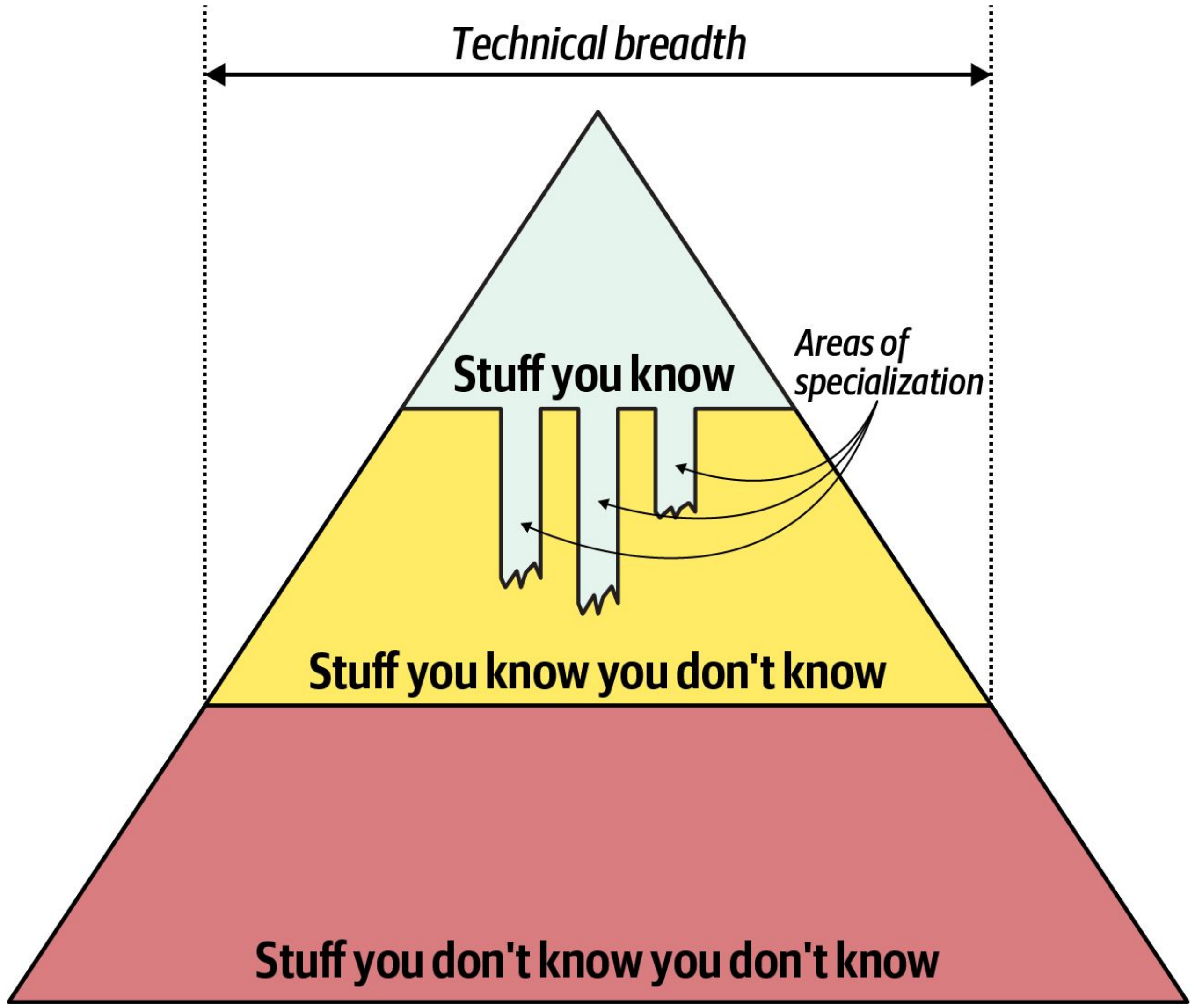
Technical debt vs breadth


Technical breadth
Technical depth
Stuff you know
Stuff you know you don't know
Stuff you don't know you don't know

- Developers: depth!
  - What you know
- Architects: breadth!
  - How much you know
- Uncomfortable: the more you learn, the more you don't know
- Depth to implement
- Breadth to take decisions
  - And then delegate implementation
- Breadth is more important than depth for an architect

Ca' Foscari University of Venice

Areas of specialization


Technical breadth
Stuff you know
Areas of specialization
Stuff you know you don't know
Stuff you don't know you don't know

- Don't completely give up
  - Important to keep some specific areas of expertise
- But be careful about the rest
  - Your knowledge will become outdated (very) soon
  - Don't make decisions with such an outdated knowledge
- Architecture evolves very fast
  - I found only books published after 2020 interesting for this course

Ca' Foscari University of Venice

Software developers career

CTOs

Software architects

Project managers

Software engineers

Software developers

Ca' Foscari University of Venice

From sw developer to sw architect

- A rather standard career promotion
  - Most of software architects were developers
- However, sw architects need to have different skills
  - Developers spend their time coding
    - Technical depth
  - Architects understands business needs, take architectural decisions, manage a team
    - Technical breadth + soft skills
- Good developer does not imply good architect
- Hard to find your path to architect…
- Some good resources: https://developertoarchitect.com/

Ca' Foscari University of Venice

Soft skills

Soft skills, also known as common skills or core skills, are skills applicable to all professions. These include critical thinking, problem solving, public speaking, professional writing, teamwork, digital literacy, leadership, professional attitude, work ethic, career management and intercultural fluency. This is in contrast to hard skills, which are specific to individual professions.
https://en.wikipedia.org/wiki/Soft_skills

WIKIPEDIA
The Free Encyclopedia

- Soft skills are needed to be a good software architect
  - Need to manage a team, interface with customers, etc etc..
- But this course is about software architecture, not soft skills!
- But from time to time we'll discuss things related to soft skills

Ca' Foscari University of Venice

Requirements of IT systems

- A requirement of an IT system is either
  - A service that must be provided
    - E.g., given two numbers a and b, returns a+b
  - A characteristic that must be satisfied
    - E.g., being able to perform 1 million of additions per second
- Functional vs non-functional requirements
  - Architecture is about taking decisions to fulfill (some) non-functional requirements
  - Functional requirements are completely left out from this course
- Topics already covered in standard sw engineering courses
  - https://moodle.unive.it/course/view.php?id=7859

Architecture characteristics

- Textbook: we dislike that term (nonfunctional requirements) because it is self-denigrating. We prefer architecture characteristics
- I personally don’t care!
- Nonfunctional requirements are widely accepted
- Architecture characteristics look fine as well
- Let’s stick to the textbook’s choice
  - But keep in mind that usually they are called nonfunctional requirements
- Auditability, performance, security, requirements, scalability, …

Criteria

Architecture characteristics **must** meet three main criteria:

1. Specifies a nondomain design consideration
   - Operational and design criteria for success
   - How to implement the requirements and why some choices were made
2. Influences some structural aspect of the design
   - Requires special structural consideration to succeed
3. Is critical or important to application success
   - Each characteristic adds complexity
   - Focus only on the ones that are essential

A non-exhaustive questionable list follows!

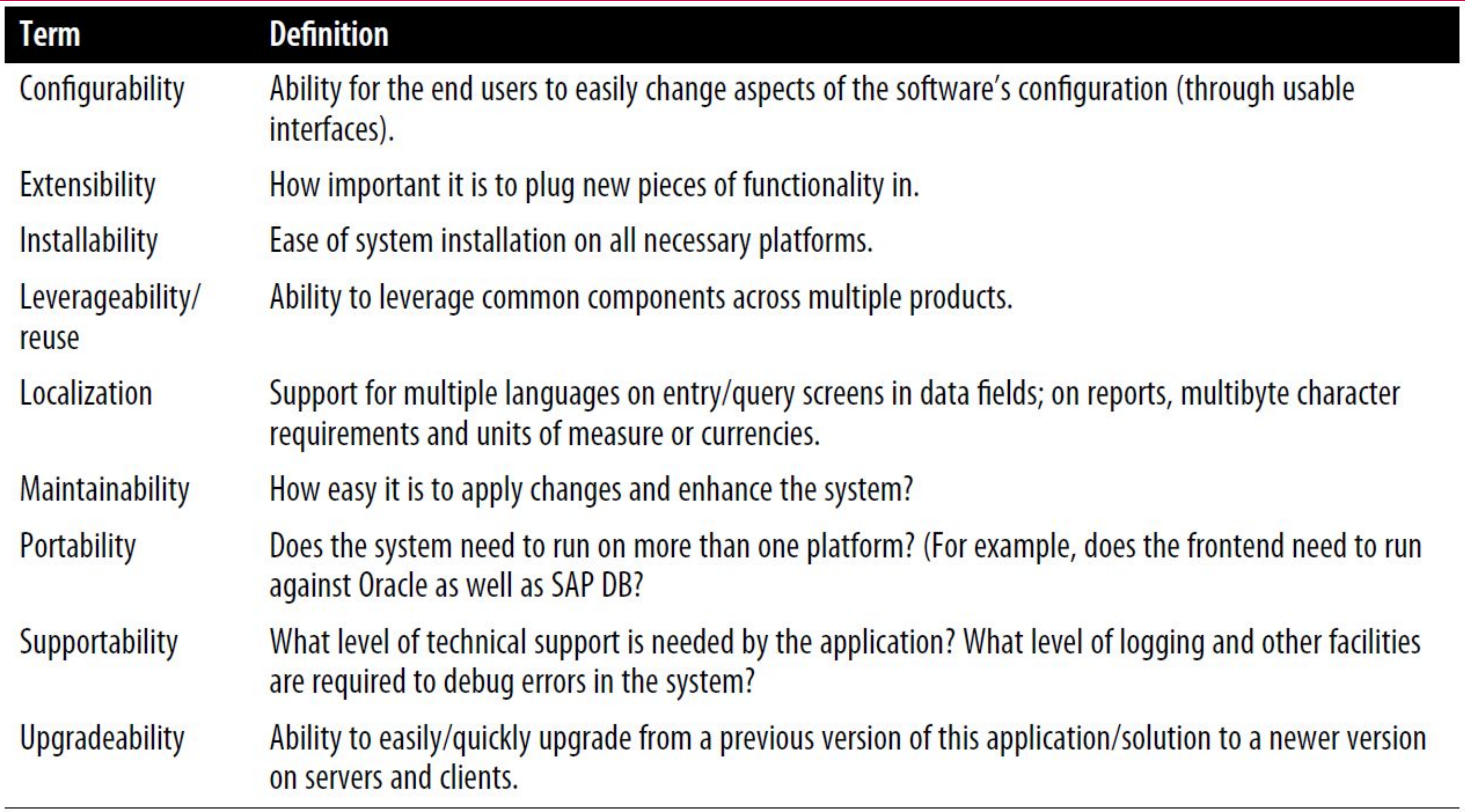
Ca' Foscari University of Venice

Operational architecture characteristics

Term	Definition
Availability	How long the system will need to be available (if 24/7, steps need to be in place to allow the system to be up and running quickly in case of any failure).
Continuity	Disaster recovery capability.
Performance	Includes stress testing, peak analysis, analysis of the frequency of functions used, capacity required, and response times. Performance acceptance sometimes requires an exercise of its own, taking months to complete.
Recoverability	Business continuity requirements (e.g., in case of a disaster, how quickly is the system required to be on-line again?). This will affect the backup strategy and requirements for duplicated hardware.
Reliability/ safety	Assess if the system needs to be fail-safe, or if it is mission critical in a way that affects lives. If it fails, will it cost the company large sums of money?
Robustness	Ability to handle error and boundary conditions while running if the internet connection goes down or if there's a power outage or hardware failure.
Scalability	Ability for the system to perform and operate as the number of users or requests increases.

Ca' Foscari University of Venice

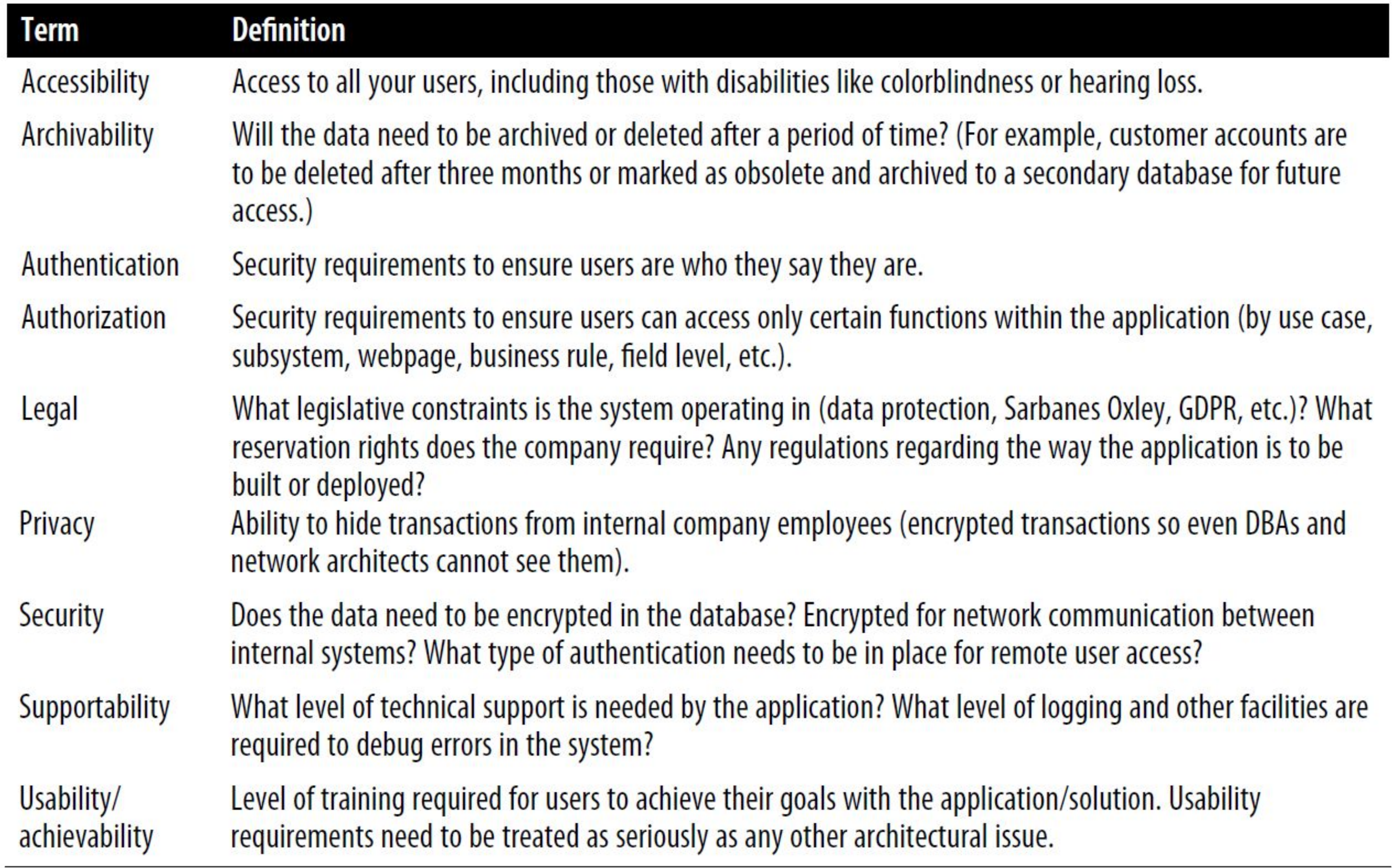
Structural architecture characteristics



Term	Definition
Configurability	Ability for the end users to easily change aspects of the software's configuration (through usable interfaces).
Extensibility	How important it is to plug new pieces of functionality in.
Installability	Ease of system installation on all necessary platforms.
Leverageability/reuse	Ability to leverage common components across multiple products.
Localization	Support for multiple languages on entry/query screens in data fields; on reports, multibyte character requirements and units of measure or currencies.
Maintainability	How easy it is to apply changes and enhance the system?
Portability	Does the system need to run on more than one platform? (For example, does the frontend need to run against Oracle as well as SAP DB?
Supportability	What level of technical support is needed by the application? What level of logging and other facilities are required to debug errors in the system?
Upgradeability	Ability to easily/quickly upgrade from a previous version of this application/solution to a newer version on servers and clients.

Ca' Foscari University of Venice

Cross-cutting architecture characteristics

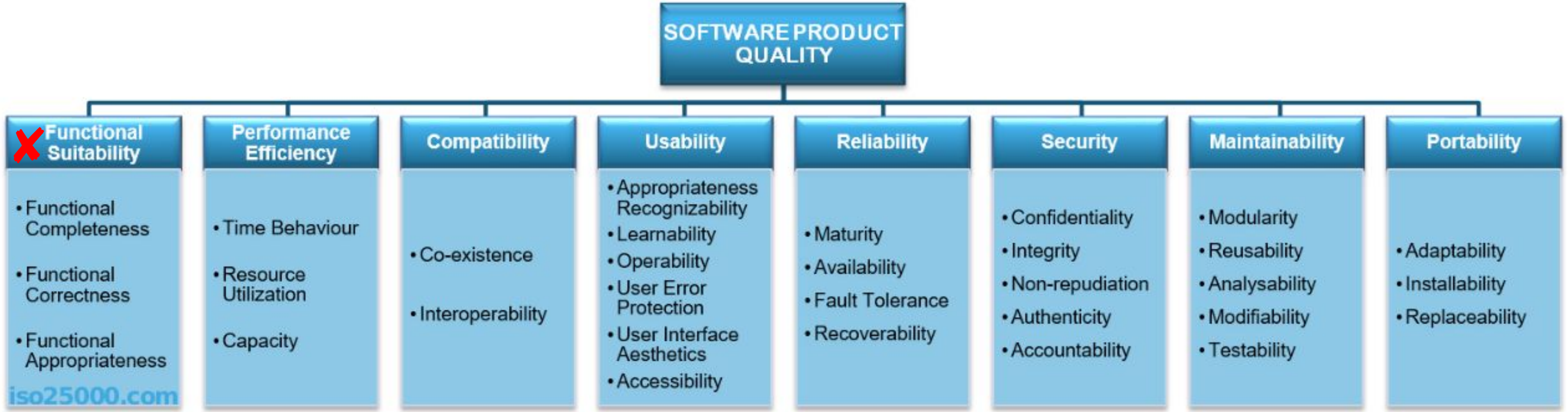


Term	Definition
Accessibility	Access to all your users, including those with disabilities like colorblindness or hearing loss.
Archivability	Will the data need to be archived or deleted after a period of time? (For example, customer accounts are to be deleted after three months or marked as obsolete and archived to a secondary database for future access.)
Authentication	Security requirements to ensure users are who they say they are.
Authorization	Security requirements to ensure users can access only certain functions within the application (by use case, subsystem, webpage, business rule, field level, etc.).
Legal	What legislative constraints is the system operating in (data protection, Sarbanes Oxley, GDPR, etc.)? What reservation rights does the company require? Any regulations regarding the way the application is to be built or deployed?
Privacy	Ability to hide transactions from internal company employees (encrypted transactions so even DBAs and network architects cannot see them).
Security	Does the data need to be encrypted in the database? Encrypted for network communication between internal systems? What type of authentication needs to be in place for remote user access?
Supportability	What level of technical support is needed by the application? What level of logging and other facilities are required to debug errors in the system?
Usability/ achievability	Level of training required for users to achieve their goals with the application/solution. Usability requirements need to be treated as seriously as any other architectural issue.

Ca' Foscari University of Venice

ISO 25010 list of capabilities

https://iso25000.com/index.php/en/iso-25000-standards/iso-25010



Slightly different and overlapping proposals. We will focus on a given (sub)set.

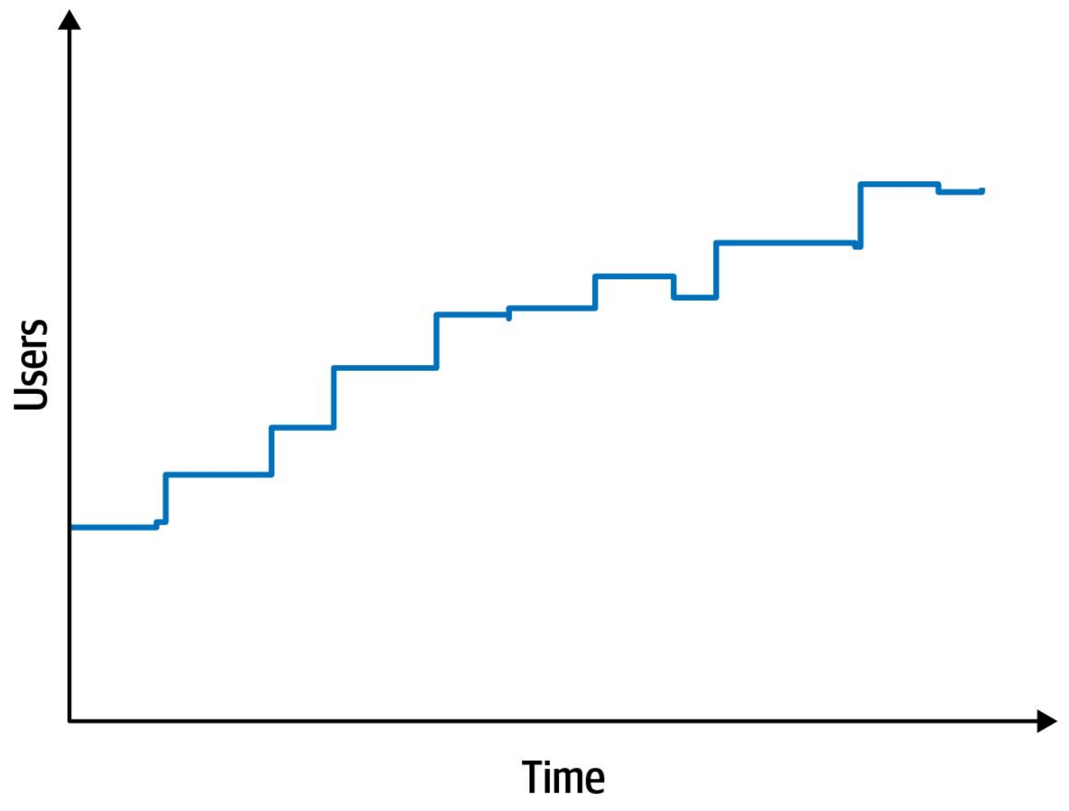
Ca' Foscari University of Venice

Our architecture characteristics

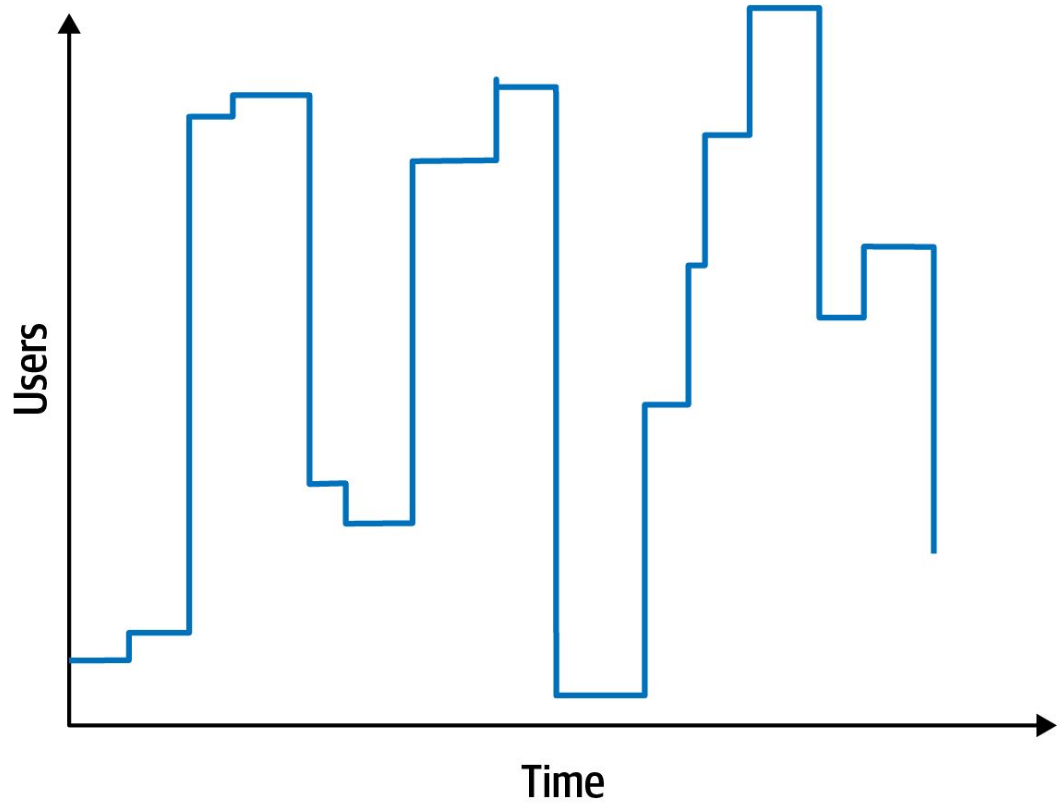
Deployability	the ease, frequency, and risk of deployment	**Performance**	amount of used resources, including time behavior, resource utilization, and capacity
Elasticity	the ability to handle bursts of users	**Reliability**	degree to which a system functions under specified conditions for a specified period of time
Evolutionary	the ability accept incremental and guided changes	**Scalability**	ability for the system to perform and operate as the number of users or requests increases
Fault tolerance	does the software operate as intended despite hardware or software faults	**Simplicity**	the ease of understanding the overall system
Modularity	degree to which the software is composed of discrete components	**Testability**	the ease of and completeness of testing
Overall cost	“additional” effort required to implement the system		

Ca' Foscari University of Venice

Scalability vs Elasticity



Scalability



Elasticity

Ca' Foscari University of Venice

Which characteristics?

Never shoot for the *best* architecture, but rather the *least worst* architecture.

- We'd like an IT system that is scalable, reliable, efficient, modular, cheap to implement, etc..
  - But we should focus on what it really matters!
  - (and it is not feasible anyway)
- Minimal set of architecture characteristics
  - With a minimal level of objectives
  - That are needed for the success of the IT system

Ca' Foscari University of Venice

So far so good… maybe!

- The list of architecture characteristics is given
  - There does not exist a clear description of each one
    - Even if it's the list chosen in the textbook!
  - The boundaries are unclear
    - Reliability vs fault tolerance? Elasticity vs scalability?
  - The description is anyway informal
    - Thus ambiguous
- We need to establish some metrics to evaluate them
  - But this is not possible on a generic description of the architecture
  - Think about that during the course to make all this more concrete!

Ca' Foscari University of Venice

SMART objectives

- When we set up an object we should have that it is:
  - Specific: target a specific area for improvement.
  - Measurable: quantify or at least suggest an indicator of progress.
  - Assignable: specify who will do it.
  - Realistic: state what results can realistically be achieved, given available resources.
  - Time-related: specify when the result(s) can be achieved.
- https://en.wikipedia.org/wiki/SMART_criteria
- We are still missing measurable
  - And we are going to underspecify it, but we must discuss…

Measuring architecture characteristics

- Architecture characteristics are not physics
  - There is no absolute way to measure them
  - Different people will interpret them differently
  - The target might be different in different systems
    - E.g., scalable to 1K or 1M or 1B users?
- Ideally, automatically assess their achievement
  - Integrate this in the DevOps lifecycle as tests
- For each characteristic we choose, we must define a measure
  - In an agile way: set a number, iterate the DevOps lifecycle, and then modify it based on the feedback

Operational characteristics

Elasticity*: the ability to handle bursts of users*
Performance*: amount of used resources, including time behavior, resource utilization,...*
Reliability*: degree to which a system functions under specified conditions for a specified period of time*
Scalability*: ability for the system to perform and operate as the number of users or requests increases*

Your wooclap poll will be displayed here

1 Install the **Chrome** or **Firefox extension**

2 Make sure you are in **presentation mode**

wooclap

https://app.wooclap.com/events/SADM/0 questions 8-11

Ca' Foscari University of Venice

Operational characteristics

***Elasticity**: the ability to handle bursts of users*
***Performance**: amount of used resources, including time behavior, resource utilization,...*
***Reliability**: degree to which a system functions under specified conditions for a specified period of time*
***Scalability**: ability for the system to perform and operate as the number of users or requests increases*

Elasticity: how much time it takes to handle new computational load, or to scale down when computational load is dropping
Performance: how much resources (CPU, RAM, …) are needed per how much computational load
Reliability: how many failures the system can accept per amount of time
Scalability: if we add more computational resources, can we handle more computational load?

Ca' Foscari University of Venice

Structural characteristics

***Deployability**: the ease, frequency, and risk of deployment*
***Fault tolerance**: does the software operate as intended despite hardware or software faults*
***Simplicity**: the ease of understanding the overall system*
***Testability**: the ease of and completeness of testing*

- Deployability: automatically deployed + time required
- Fault tolerance: simulation of faults
- Simplicity: hard to measure… (see next slide)
- Testability: automatic testing + time required + code coverage

Ca' Foscari University of Venice

Measuring modularity

***Modularity**: degree to which the software is composed of discrete components*

- Module: each of a set of independent units that can be composed to construct a more complex structure
  - Group of classes in object-oriented programming languages
    - E.g., packages or modules in Java, namespaces in C#
  - Group of functions in imperative or functional programming languages
- Logical distinction, not necessarily a physical one
  - Even if modules usually are shipped through artifacts (e.g., jar files)
- Modules might depend on each other
  - But in general we cannot have static cyclic dependendencies

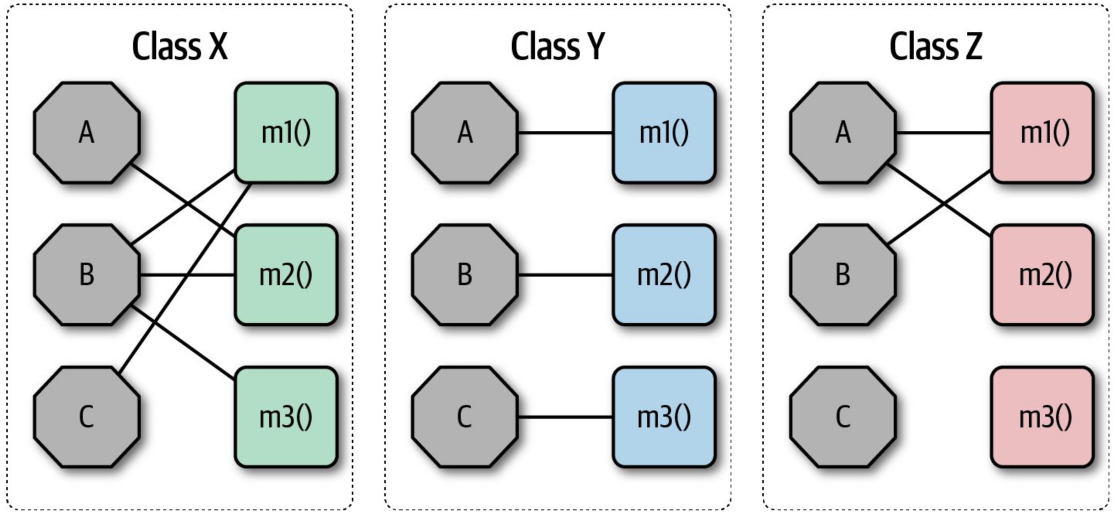
Measuring modularity Cohesion

***Modularity**: degree to which the software is composed of discrete components*

- Cohesion measures how related the parts are to one another
- Many different types: functional, sequential, communicational, procedural, temporal, etc…
- Chidamber and Kemerer Object-oriented metrics suite
  - https://en.wikipedia.org/wiki/Programming_complexity#Chidamber_and_Kemerer_Metrics
  - Lack Of Cohesion Metric

$$LCOM = \begin{cases} |P| - |Q|, & \text{if } |P| > |Q| \\ 0, & \text{otherwise} \end{cases}$$

- P: methods not accessing a shared field
- Q: methods sharing a shared field



High Low Medium

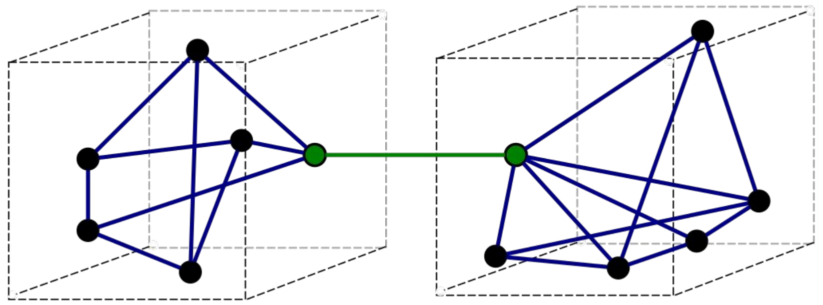
Ca' Foscari University of Venice

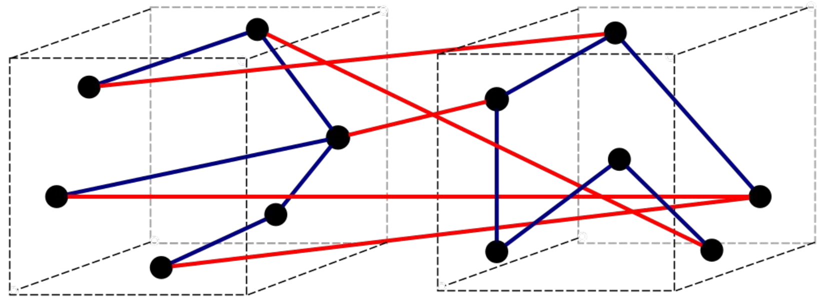
Measuring modularity Coupling

***Modularity**: degree to which the software is composed of discrete components*

Attempting to divide a cohesive module would only result in increased coupling and decreased readability. (Larry Constantine)

- Coupling: the degree of interdependence between modules
- Afferent: the number of incoming connections to a code artifact
- Efferent: the number of outgoing connections to other code artifacts


a) Good (loose coupling, high cohesion)


b) Bad (high coupling, low cohesion)

Ca' Foscari University of Venice

Measuring modularity Coupling

***Modularity**: degree to which the software is composed of discrete components*

- Abstractness: measures the amount of abstract elements
  - Too low: just a main method
  - Too high: hard to reason

$$A = \frac{\sum m^a}{\sum m^c}$$

#abstract elements (abstract classes, interfaces, …)

#concrete elements (non abstract classes)

- Instability: % efferent coupling
  - Too high: code breaks more easily when changed
    - Because it delegates most of the work to other modules

$$I = \frac{C^e}{C^e + C^a}$$

efferent (outgoing) coupling

afferent (incoming) coupling

Ca' Foscari University of Venice

Measuring modularity Coupling

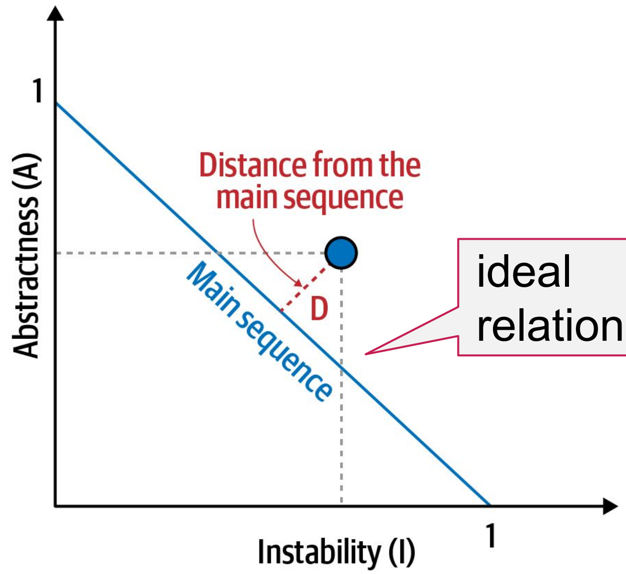
***Modularity**: degree to which the software is composed of discrete components*

- Abstractness and instability always fall between zero and one
  - In practical contexts… maybe…
- More abstract -> more stable
- If above the line, uselessness
  - Too abstract and too instable, impossible to reason on it
- If below the line, pain
  - Too much implementation and not enough abstraction, hard to mantain

abstractness

$$D = |A + I - 1|$$

instability



ideal relationship

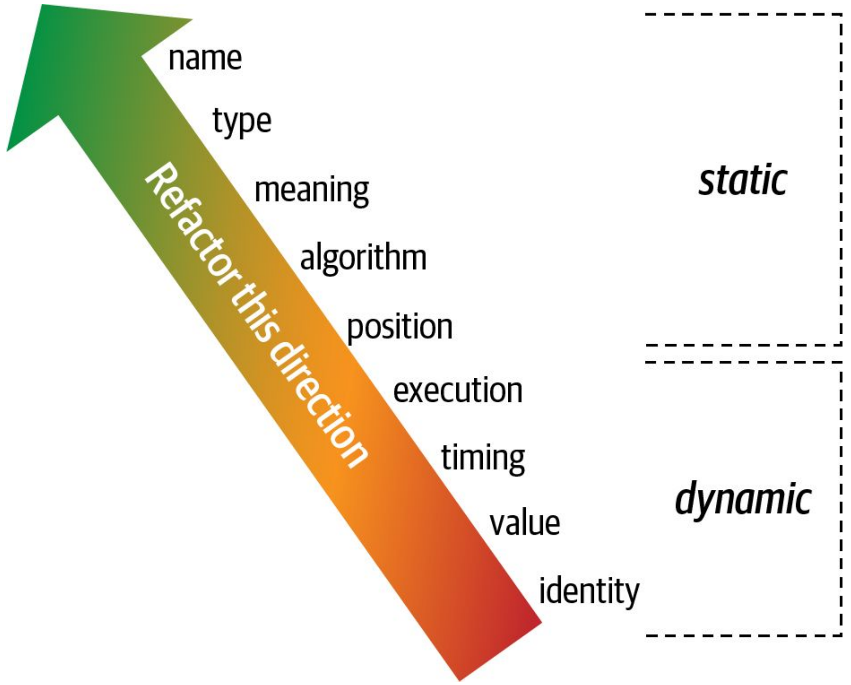
Ca' Foscari University of Venice

Measuring modularity
Connascence

***Modularity**: degree to which the software is composed of discrete components*

Two components are connascent if a change in one would require the other to be modified in order to maintain the overall correctness of the system (Meilir Page-Jones)

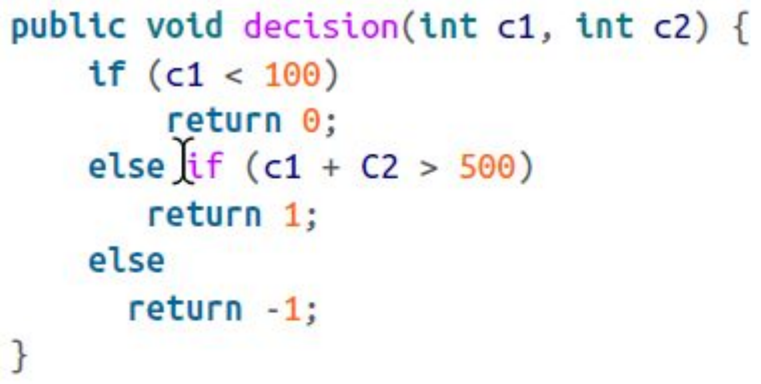
- Static: source code level coupling
  - Need to modify the code
  - Name, type, position, algorithm, …
- Dynamic: execution/runtime coupling
  - Order of execution, timing, values, …



Measuring modularity Cyclomatic complexity

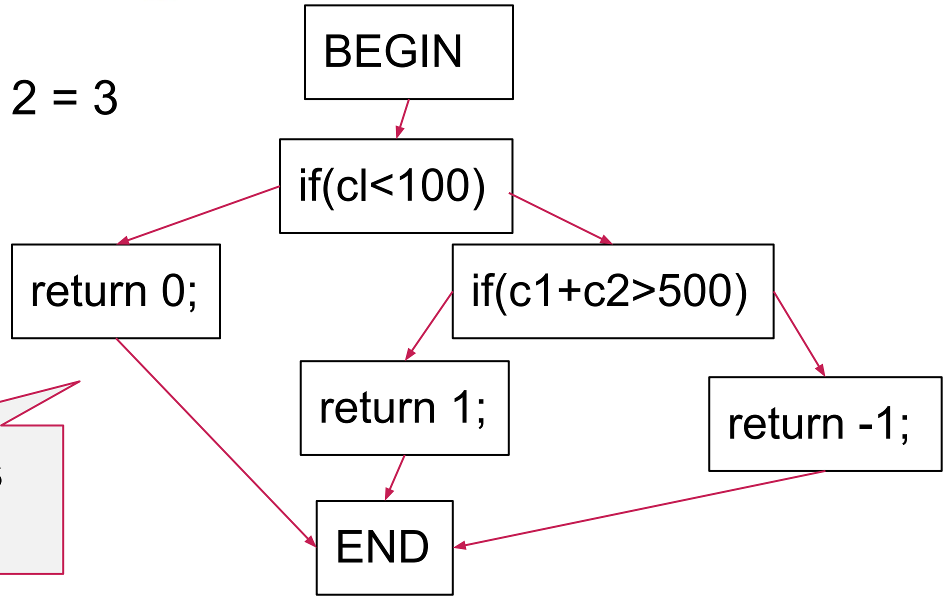
***Modularity**: degree to which the software is composed of discrete components*

- Measure the structure of the code
- Cyclomatic complexity
  - CC = E - N + 2
- Program as a (control flow) graph
  - N: number of lines of code
    - Statements or sequential blocks
  - E: number of “jumps”
    - E.g., if statements have two jumps
- Questionable anyway but
  - Provides a clear index
  - It can be automatically assessed

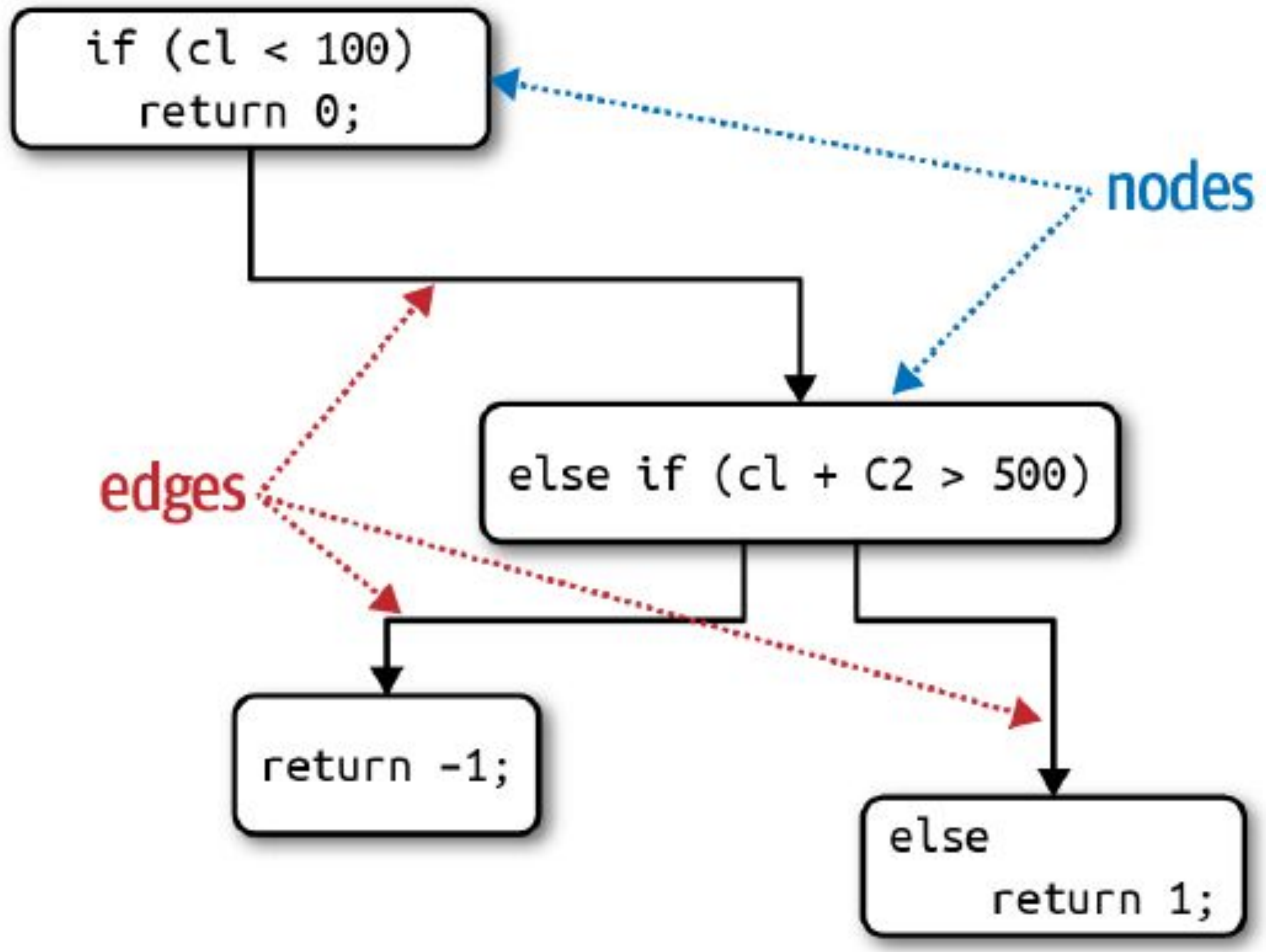


```
public void decision(int c1, int c2) {
    if (c1 < 100)
        return 0;
    else if (c1 + C2 > 500)
       return 1;
    else
      return -1;
}
```

CC = 8 - 7 + 2 = 3



Textbook is different


if (cl < 100)
return 0;
nodes
edges
else if (cl + C2 > 500)
return -1;
else
return 1;

Ca' Foscari University of Venice

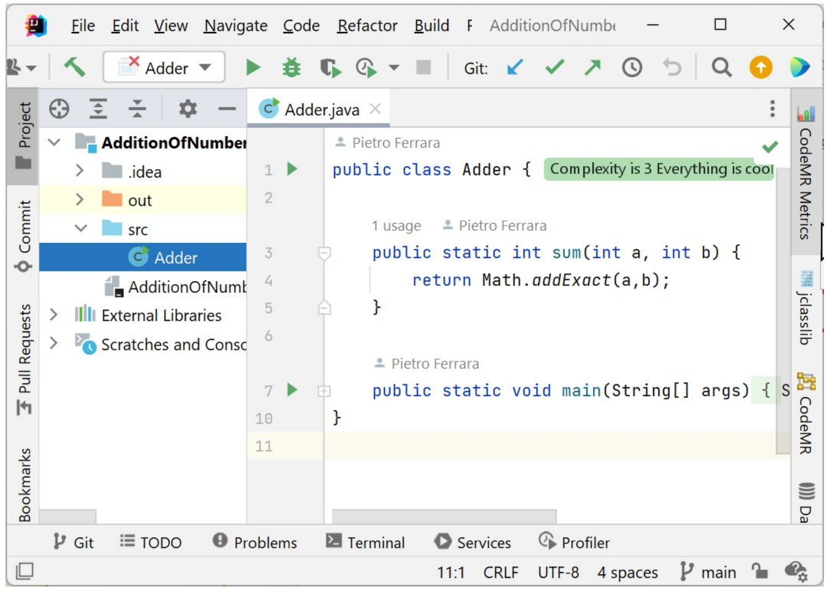
Summing up

Characteristic	What is needed			What it concerns		
	Code	Artifact	Deployment	Speed	Robustness	Structure
Deployability		X				X
Elasticity			X	X		
Evolutionary						X
Fault tolerance			X		X	
Modularity	X					X
Overall cost						
Performance		X		X		
Reliability		X	X		X	
Scalability			X	X		
Simplicity	X					X
Testability		X			X	

Ca' Foscari University of Venice

Technologies

Source code



```
Pietro Ferrara
public class Adder {   Complexity is 3 Everything is cool

    1 usage   Pietro Ferrara
    public static int sum(int a, int b) {
        return Math.addExact(a,b);
    }

    Pietro Ferrara
    public static void main(String[] args) { S
}
```

Compile and build

Gradle

Artifact

JAR

Deploy

spring

Server

docker

Katas
https://nealford.com/katas/list.html

- From where do we start when thinking about an IT system?
- Let’s adopt the approach of katas
  - In martial arts, it is an individual exercise
- High-level domain targeted description of an IT system
- Next step: derive architecture characteristics from katas
- Katas are composed by:
  - Description: The overall domain problem the system is trying to solve
  - Users: The expected number and/or types of users of the system
  - Requirements: Domain/domain-level requirements, as an architect might expect from domain users/domain experts
  - Additional context

First task

Implement an IT system that manages hospitals, e.g.,

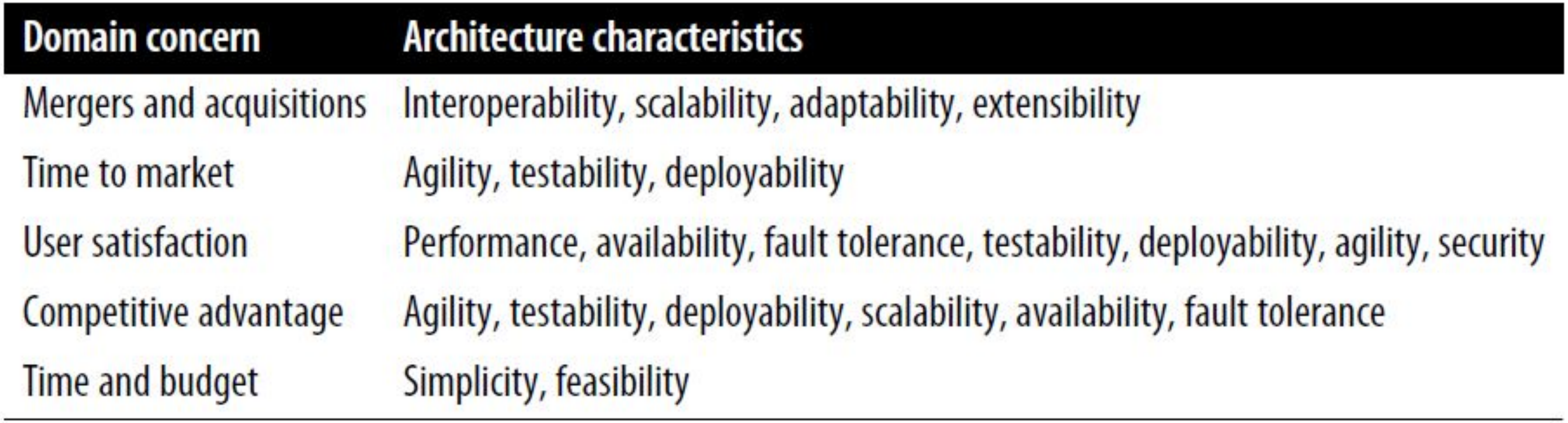
- Emergencies, patient medical records, schedule of the doctors, exams, hospital ward, …

- Focus on a subsets of (functional and nonfunctional) features
  - You will need to implement the system at the end!

From domain concerns to architecture characteristics

- Need to find an agreement with domain stakeholders
  - But software architects talk a different language
- Architecture characteristics mapped to domain concerns?
  - It is not a one to one relationship
- Let's focus on few domain concerns:
  - mergers and acquisitions, time to market, user satisfaction, competitive advantage, time and budget
- What architecture characteristics we need?
  - Elasticity, Performance, Reliability, Scalability, Deployability, Fault tolerance, Simplicity, Testability, Modularity

Textbook “solution”



Domain concern	Architecture characteristics
Mergers and acquisitions	Interoperability, scalability, adaptability, extensibility
Time to market	Agility, testability, deployability
User satisfaction	Performance, availability, fault tolerance, testability, deployability, agility, security
Competitive advantage	Agility, testability, deployability, scalability, availability, fault tolerance
Time and budget	Simplicity, feasibility

Explicit characteristics from requirements

- Sometimes requirements make explicit the characteristic
  - E.g., “need to deal with ~10Ks concurrent users”
- Consider each single requirement in the kata/specification
  - Understand what consequences it has on characteristics
  - Both requirements and users sections are relevant
- However, carefully consider what is impacted
  - E.g., “the IT system is expected to get 100Ks users”
    - Concurrent users? Always or just in some timeslots (e.g., promotion)?
    - Elasticity vs scalability
- Always discuss with the stakeholders what they want

Ca' Foscari University of Venice

Implicit characteristics

- Unfortunately, stakeholders leave many things implicit
- E.g., probably nobody will put a requirement on security
  - But are you willing to get an unsecure IT system at the end?
- Another common example is availability
  - One expects that the system will be available <u>most of the time</u>
- But more characteristics lead to more complexity
  - That lead to higher costs!
- Common sense rules
  - Make explicit all the implicit requirements with the stakeholders
  - Choose the requirements with the highest priority

Ca' Foscari University of Venice

References

- Textbook, Part I
  - Technical depth/breadth: chapter 2
  - Architecture characteristics: chapter 4
  - Measuring architecture characteristics: chapter 6
  - Measuring modularity: chapter 3
  - From requirements to architecture characteristics: chapter 5