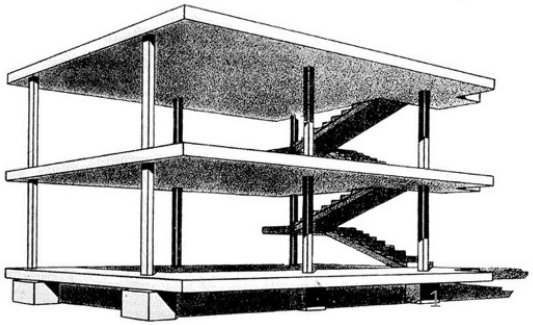
TU Delft IN4315: Software Architecture Lecture 2: Envisioning the Product

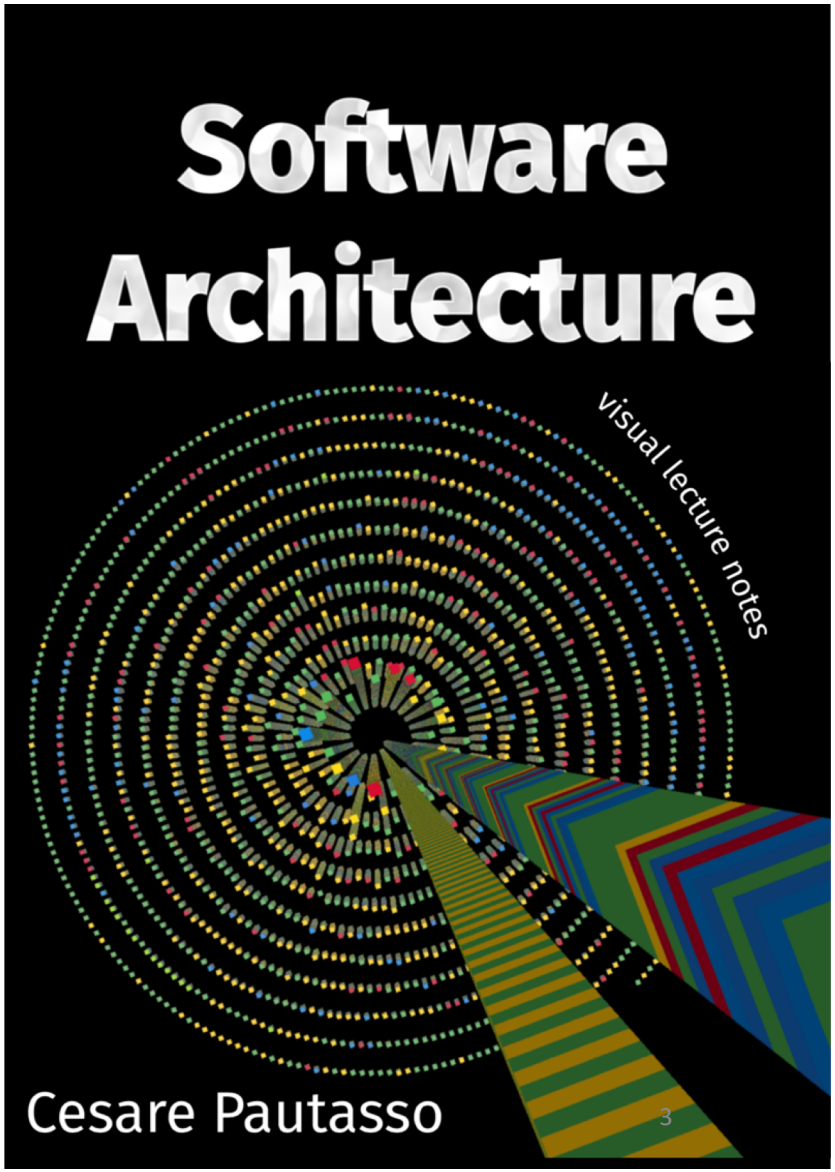
Arie van Deursen



Wikipedia, Dom-Ino House, Corbusier

Date	Start	End	Activity	Teacher	Topic
Wed Feb 10	13:45	15:30	Lecture 1	Arie van Deursen	Introduction and Course Structure
Fri Feb 12	08:45	10:30	Lecture 2	Arie van Deursen	Envisioning the System
Wed Feb 17	13:45	15:30	Lecture 3	Arie van Deursen	Realizing the Vision
Fri Feb 19	08:45	10:30	Lecture 4	Arie van Deursen	Continuous Evolution
Wed Feb 24	13:45	15:30	Lecture 5	Luís Cruz	Architecting for Sustainability
Fri Feb 26	08:45	10:30	Lecture 6	Burcu Kulahcioglu Ozkan	Architecting for Distribution
Wed Mar 3	13:45	15:30	Lecture 7	Diomidis Spinellis	50 years of Unix Architecture
Fri Mar 5	08:45	10:30	Lecture 8	TBD	
Wed Mar 10	13:45	15:30	Lecture 9	TBD	
Fri Mar 12	08:45	10:30	Lecture 10	Xavier Devroey	Software Variability Management
Wed Mar 17	13:45	15:30	Lecture 11	TBD	
Fri Mar 19	08:45	10:30	Lecture 12	Daniel Gebler (Picnic)	Architecting for business as *unusual*
Wed Mar 24	13:45	15:30	Lecture 13	TBD	
Fri Mar 26	08:45	10:30	Lecture 14	Ferd Scheepers (ING)	Architecting for the Enterprise
Thu Apr 1	08:45	17:30	Finale	All students	Final presentations

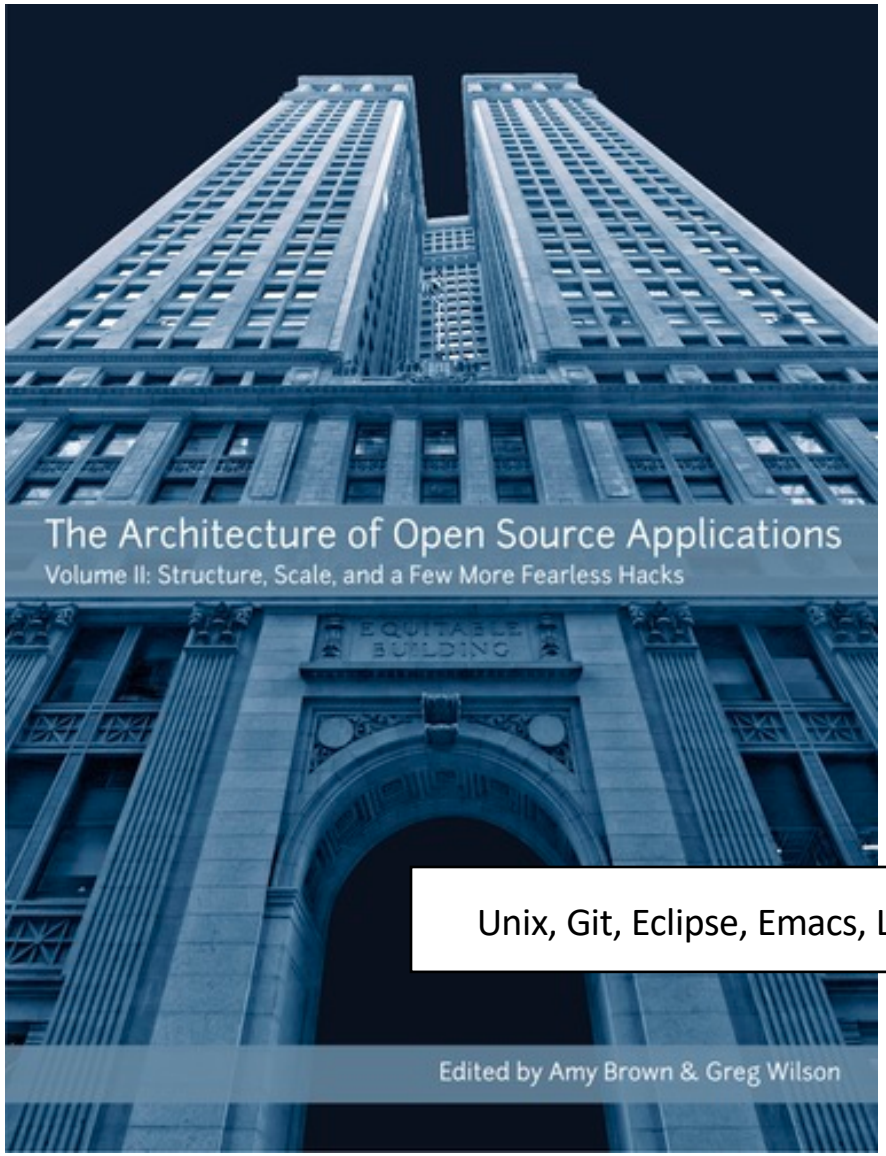
1. Introduction
2. Quality Attributes
3. Definitions
4. Modeling Software Architecture
5. Modularity and Components
6. Reusability and Interfaces
7. Composability and Connectors
8. Compatibility and Coupling
9. Deployability, Portability and Containers
10. Scalability
11. Availability and Services
12. Flexibility and Microservices

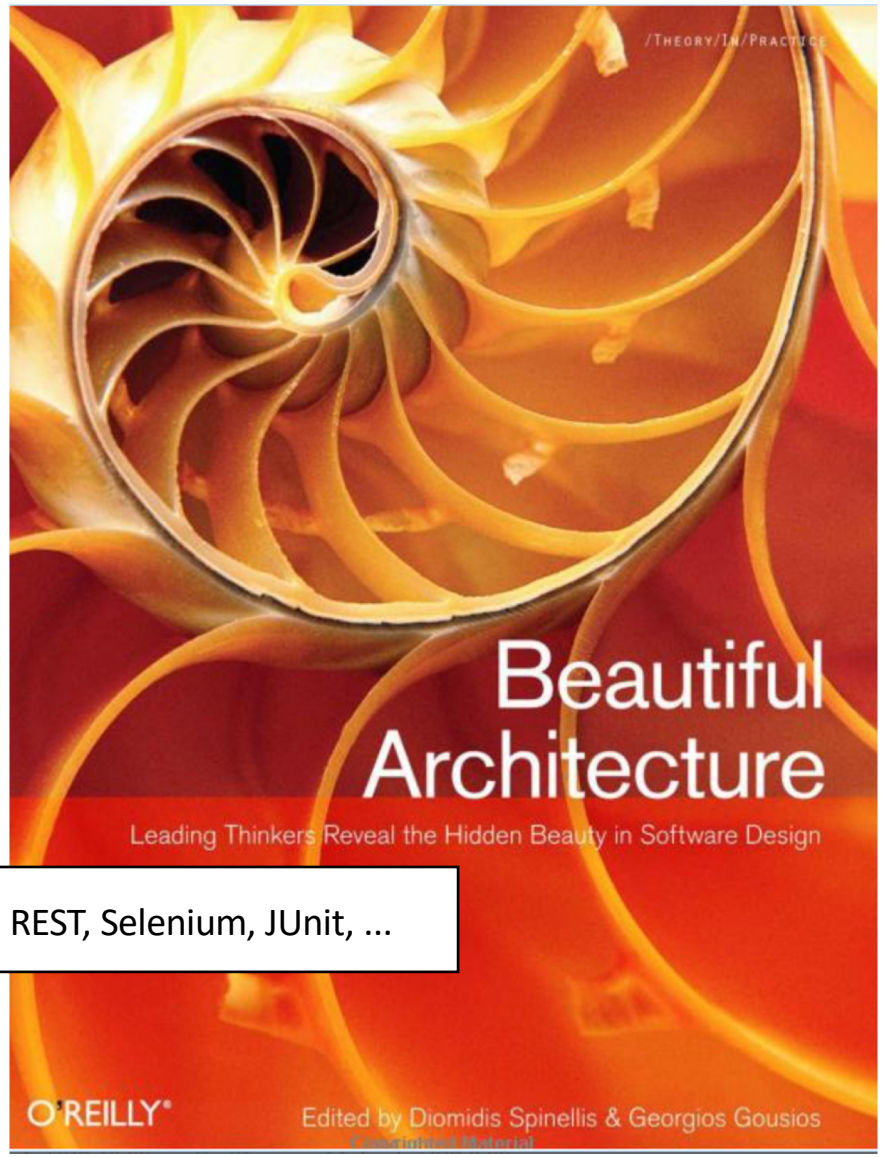


Defining Software Architecture

1. How would you define software architecture?
2. How does your definition of software architecture relate to what an architect does?
3. Can you name examples of well known systems with great or influential architectures?

Please enter your thoughts in the chat!





Unix, Git, Eclipse, Emacs, LLVM, REST, Selenium, JUnit, ...

The Architecture of a System (IEEE):

- The set of fundamental concepts or properties
- of the system in its environment,
- embodied in its elements and relationships,
- and the principles of its design and evolution.

The Architecture of a System (Roy Fielding)

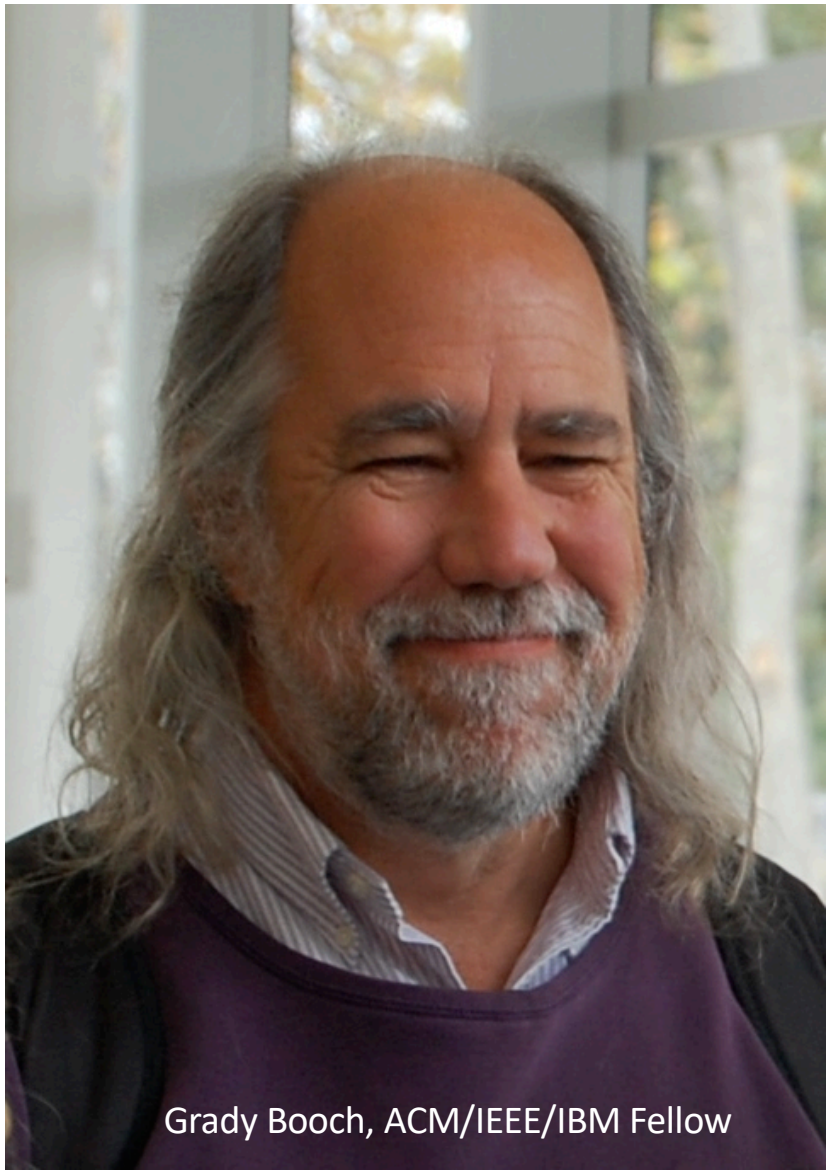
- A configuration of architectural elements (components, connectors, and data)
- constrained in their relationships
- in order to achieve a desired set of architectural properties.

Constraints: Focus on what is and is *not* allowed

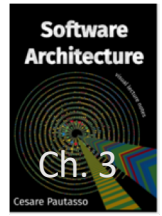
Architecture Defined

Architecture represents
the significant design decisions
that shape a system
where significant is measured
by cost of change

(Grady Booch, March 2006)



Grady Booch, ACM/IEEE/IBM Fellow



Basic Definition

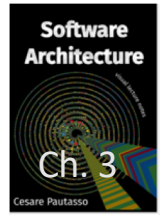
- A software system's architecture is the set of **principal design decisions** made about the system.

Architecture = {Principal Design Decisions}

- It is the blueprint for a software system's shared understanding, necessary for its construction and evolution

N. Taylor et al.

“Principal Design Decisions”



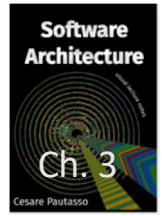
Aspects:

- Structure
- Behavior
- Interaction
- Deployment
- User Interface
- Implementation

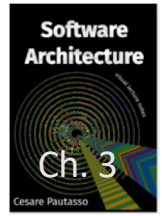
Principal

- Needed to meet (quality) goals of stakeholders
- Shape (likely) future design decisions
  - “already taken” to make life easier
  - “simplify”: offering guidance on future decisions
  - “limit”: In retrospect unwise, but hard to change.

Descriptive or Prescriptive?

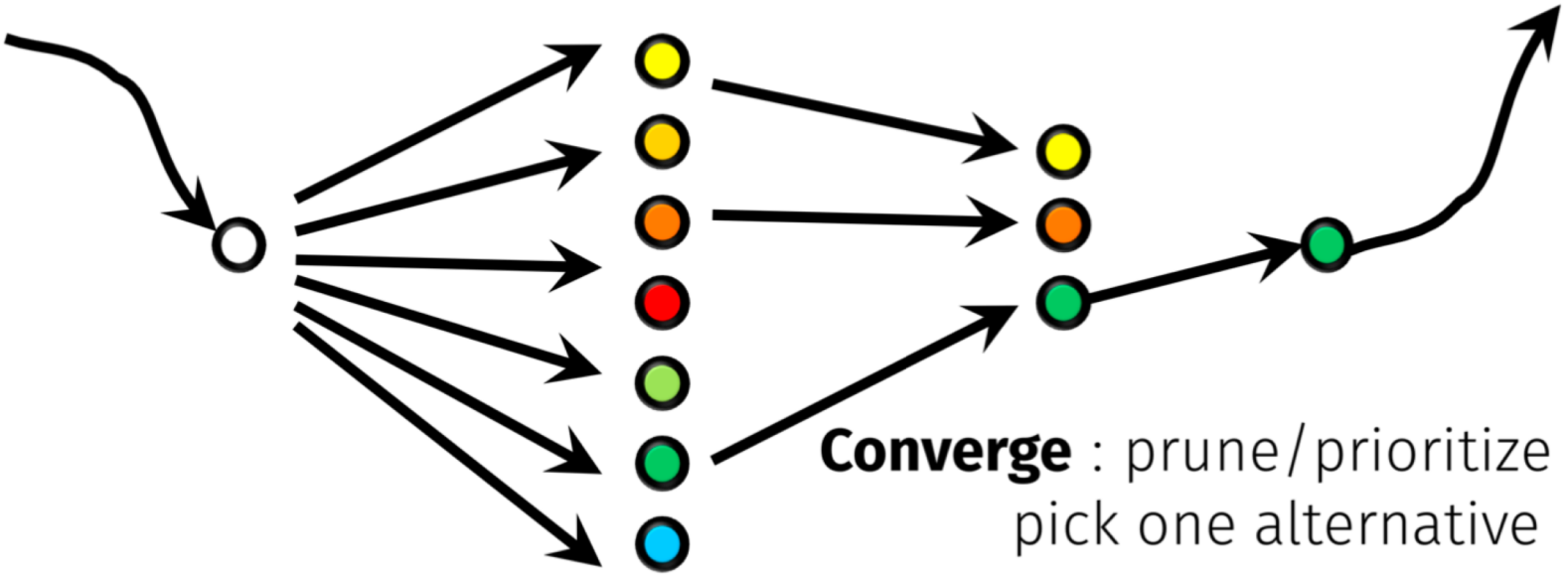


- Prescriptive:
  - Decisions of the future – architecture as it should be
  - Idealized version of the past – architecture as it should have been
- Descriptive:
  - Decisions of the past – architecture as it is
  - Maybe complex due to repeated violations / short cuts
  - May have to be “recovered” from actual system (architectural archeology)
- Erosion:
  - Undermining of originally prescribed architecture



Decision Making Phases

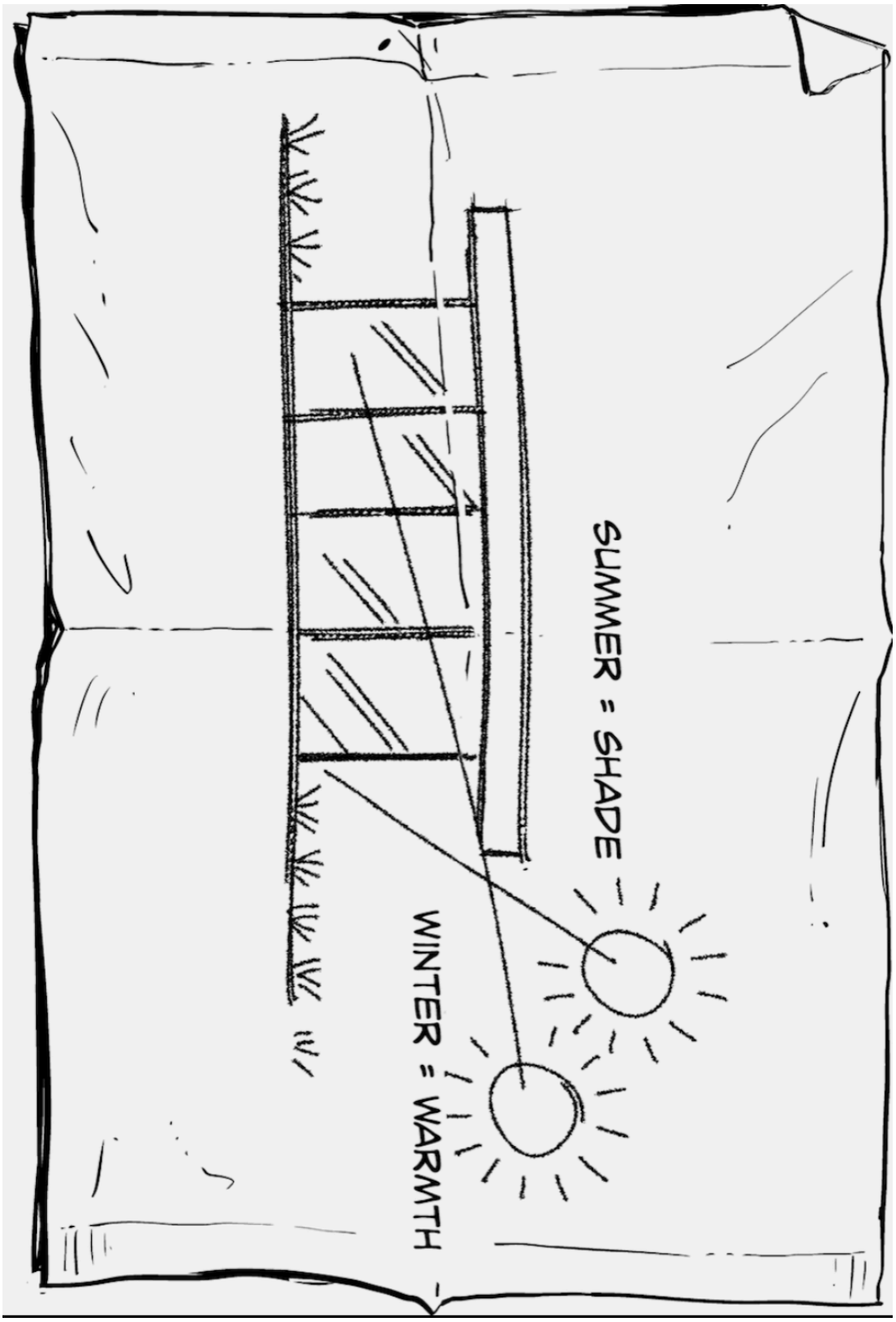
Diverge : brainstorm/generate many
possible alternatives to solve a given design issue



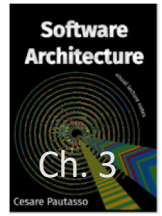
Converge : prune/prioritize
pick one alternative

Design issue Design alternatives Design decision, with rationale

Gregor Hohpe. The architect elevator.

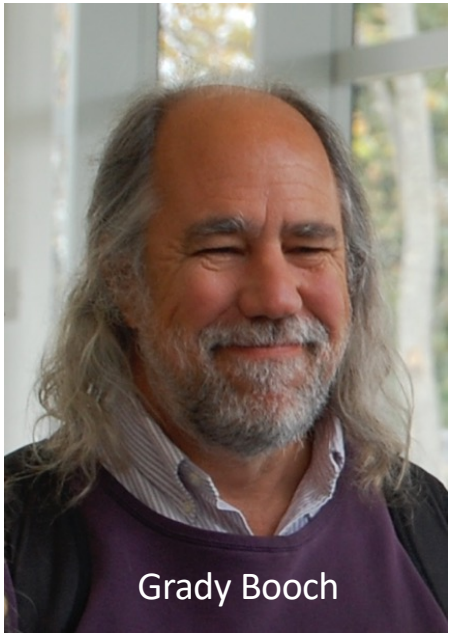


Modeling Architectural Decisions?

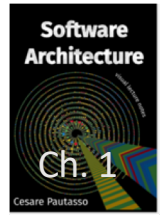


- Static
  - Structure
  - Decomposition
  - Interfaces
  - Components
  - Connectors
  - Dependencies
- Dynamic
  - Behavior
  - Deployment

- Styles and Patterns
- Design Process
  - Constraints
  - Rationale
  - Quality Assurance
  - Team Organization
- Target Audience
  - Technical Developers
  - Marketing/Customers
  - Management

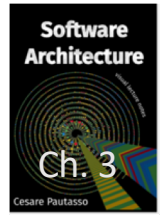


Why Software Architecture?



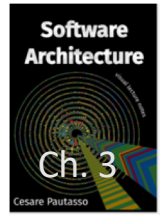
- Manage complexity through abstraction
- Communicate, remember and share global design decisions among the team
- Visualize and represent relevant aspects (structure, behavior, deployment, ...) of a software system
- Understand, predict and control how the design impacts quality attributes of a system
- Define a flexible foundation for the maintenance and future evolution of the system

“Art or Science”?



- The “artistic” part of software architecture is minimal
- Creativity, Originality and Aesthetics are less valued than Feasibility, Viability and Fitness to a purpose.
- Even the best architects **copy solutions, styles and patterns that have proven themselves in practice**, adapt them to the current context, improve upon their weaknesses, and then assemble them in novel ways with incremental improvements.
- An architectural process can be established with intentional artifacts, clear activities, and well-defined project management milestones

“Science or Art”?

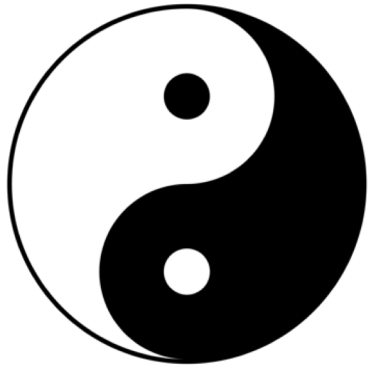


- There exists only a modest body of knowledge about software architecture
- Scientific and analytical methods are still lacking - those that do exist are hard to apply in practice
- There is **no perfect design**: architecture is a wicked problem involving subjective tradeoffs and the management of extreme ambiguity and contradiction
- **Experience counts**: the best architects are grown, not born

Dialectic Learning in Architecture

1. Just do it: Engage in architectural activities in realistic setting
2. Study / *internalize* existing theories and approaches
3. Confront the two with each other
   - How does this theory really work?
   - Does this theory apply to my system? Why? Why not?

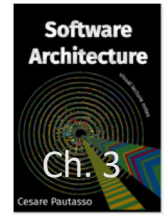
Thesis:	Theory
Anti-Thesis:	Practice
Synthesis:	Understanding

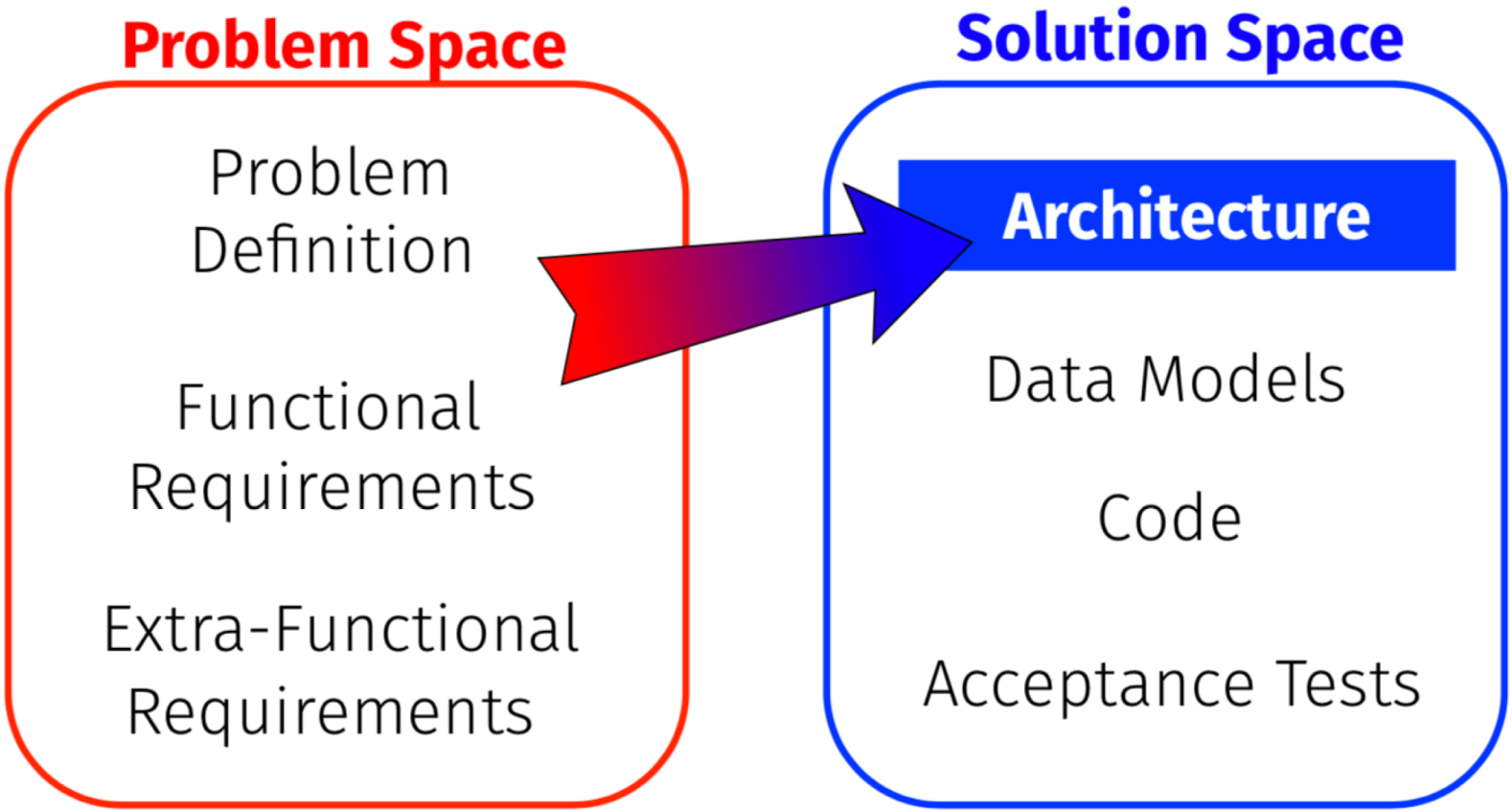


Essay 1:
Articulating the Product's Vision and Architectural Drivers

Software Architecture is about People

- “Maybe half of software development is about nerd stuff happening at the whiteboard and about typing at the keyboard.”
- “The other half is about people and relationships. “
- There are few software team activities where this becomes more obvious than during architecture formulation.


Software
Architecture
Ch. 3
Cesare Pautasso


Problem Space
Problem Definition
Functional Requirements
Extra-Functional Requirements
Solution Space
Architecture
Data Models
Code
Acceptance Tests

Mapping Between Problems and Solutions?

- The relationship between problem and solution is rich and complex.
- You can't just start with a problem definition and methodically elaborate it into a solution
- The mapping from problems to solutions is many-to-many
- Multiple seemingly unrelated solutions might be required to solve what you perceive as a single problem
- Some problems seem to defy any mapping at all

Modern: Turn technical capabilities into new business opportunities

Shared Story = Product Vision

- Clear vision of what the product is and will do
- Simple, compelling, articulated, shared
- Comes with a credible roadmap towards this vision.
- Expressible in terms that are understandable to end users
- Driven / enabled by sound architectural foundations

- Co-production of product manager and architect

Our Mission

[redacted] is the leading destination for short-form mobile video. Our mission is to inspire creativity and bring joy.

[redacted]

Our mission is to unlock the potential of human creativity—by giving a million creative artists the opportunity to live off their art and billions of fans the opportunity to enjoy and be inspired by it.

About

[redacted] is an open source driver assistance system. [redacted] performs the functions of Automated Lane Centering and Adaptive Cruise Control for over 85 supported car makes and models.

[redacted]

Long before we knew that it would be called [redacted] we knew what we wanted it to be. Instead of teaching the rest of the world cryptography, we wanted to see if we could develop cryptography that worked for the rest of the world. At the time, the industry consensus was largely that encryption and cryptography would remain unusable, but we started [redacted] with the idea that private communication could be simple.

Our Mission

TikTok is the leading destination for short-form mobile video. Our mission is to inspire creativity and bring joy.

For the Record

Our mission is to unlock the potential of human creativity—by giving a million creative artists the opportunity to live off their art and billions of fans the opportunity to enjoy and be inspired by it.

About

openpilot is an open source driver assistance system. openpilot performs the functions of Automated Lane Centering and Adaptive Cruise Control for over 85 supported car makes and models.

Signal Foundation

moxie0 on 21 Feb 2018

Long before we knew that it would be called Signal, we knew what we wanted it to be. Instead of teaching the rest of the world cryptography, we wanted to see if we could develop cryptography that worked for the rest of the world. At the time, the industry consensus was largely that encryption and cryptography would remain unusable, but we started Signal with the idea that private communication could be simple.

Recognizing a System's Stakeholders

(End) Users

- Video / music producers and consumers
- Advertisers
- Regulators (privacy, intellectual property)
- …

Business

- Marketing and sales
- Management
- Investors
- …

Development & Operations

- Developers
- Operations
- Testers
- Designers
- Architect
- …

Stakeholders in Open Source?

- Rich trail of communication in issues, discussions, reviews, ...
- Sometimes the end users are other developers

marcphilipp commented 7 mo

I tried to take a step back and think in terms of who wants to use JUnit and in which way. Here are my thoughts. Plea free to correct me or ask questions if anything is not clear.

We have (at least) the following existing use cases:

- Maven users with `jar` dependencies to JUnit.
  - From 4.11 on there will only be junit:junit which does not include Hamcrest but declares a dependency to i
  - The old junit:junit-dep simply points to junit:junit.
- Non-Maven users who simply want to download an all in one jar.
  - That's what `junit.jar` is for.
- Non-Maven Java users that want to use a different (newer) version of Hamcrest with JUnit.
  - They can download `junit-dep.jar` and use their custom Hamcrest JAR.

Adding support for OSGi will not replace any of these use cases but rather add new ones:

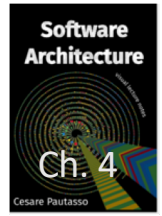
1. Apache Felix Maven Users want a `bundle` artifact version of JUnit.
   - Open Question: Can this be the same Maven artifact?
2. Non-Maven OSGi users want to use an all-in-one OSGi bundle version of JUnit.
   - Here, we could package Hamcrest into the same JAR.
3. Non-Maven OSGi users want to do it right and use an OSGi-like bundle that declares a dependency to a Hamcr package that comes from some other (unspecified) bundle.

We should definitely think about which ones of these we need to support.

Thinking about solutions:

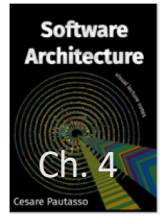
- `2` and `3` could be solved by shipping OSGi manifests in `junit.jar` and `junit-dep.jar`. They do not re Maven build, we could simply write the manifest ourselves or use the Ant bnd plugin to do it. Regular JUnit user not be affected by the modified manifest.
- For `1` it would make sense to have a Maven build process like the one **@Tibor17** has sketched in #472. Here have to make sure that we will not affect regular Maven users.

The “Domain Model”



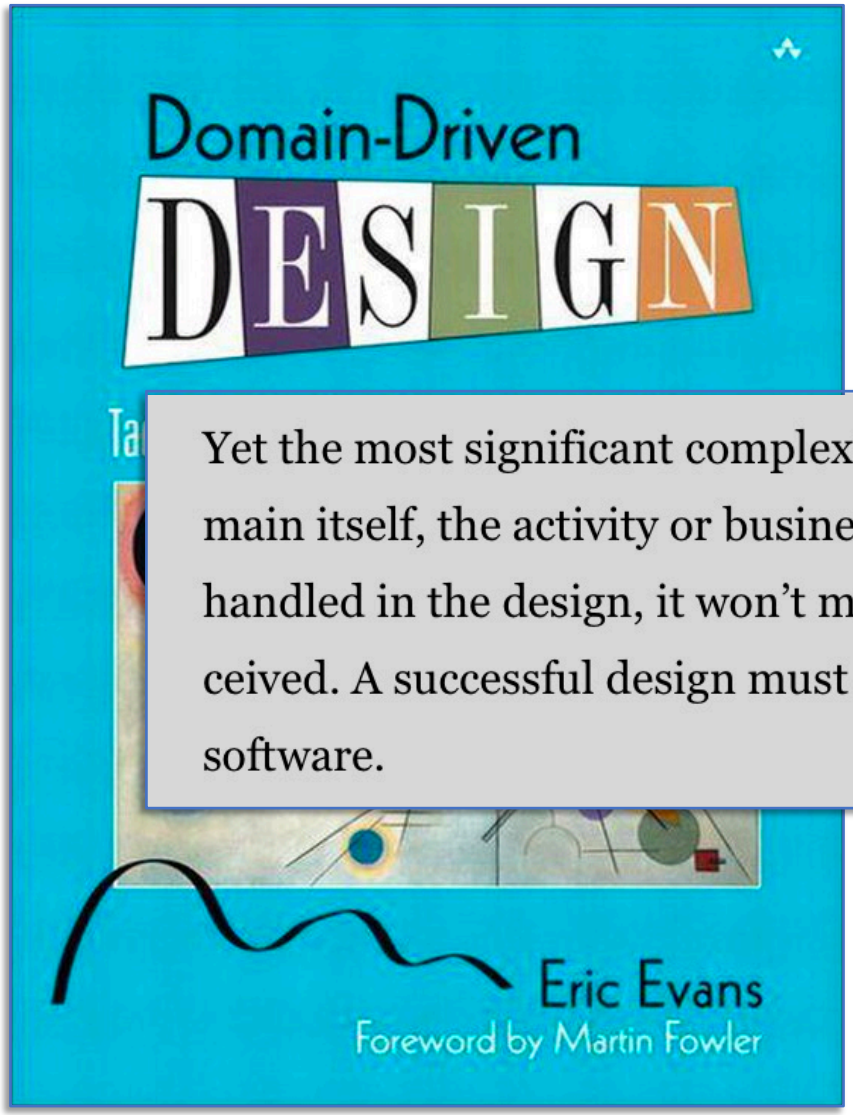
- Refutable truths about the real-world
- Outside your control
- Your system will be evaluated against it
- Architecturally significant requirements

- Problem domain description:
  - Information (invariants, navigation, snapshots)
- Functionality (use-case scenarios, feature models)
- Define shared vocabulary and understanding towards your customer, domain expert



Example Domain Model

- Music songs are organized in albums
- The same song can be authored by many artists
- Listening to each song costs 0.99 CHF, but short samples can be heard for free
- Songs can be downloaded and also live streamed
- Songs are stored in files of standard MP3 format



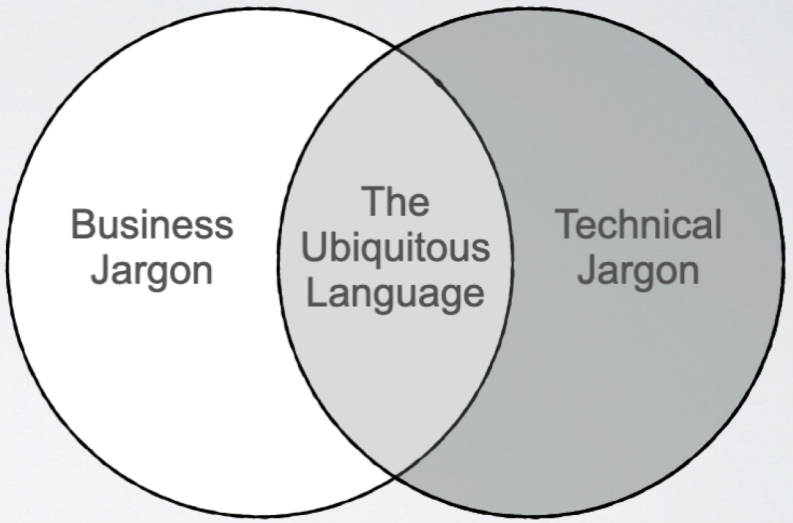
Yet the most significant complexity of many applications is not technical. It is in the domain itself, the activity or business of the user. When this domain complexity is not handled in the design, it won't matter that the infrastructural technology is well conceived. A successful design must systematically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

DDD

The ubiquitous language:

To know what a spade is; how to use it; what it does … ask someone who knows.

Each class, each method, each variable should be carefully named so that the story they tell is the business story you're writing.

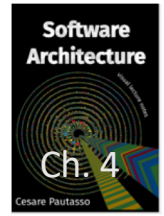


Slide by Mitchell Olsthorn

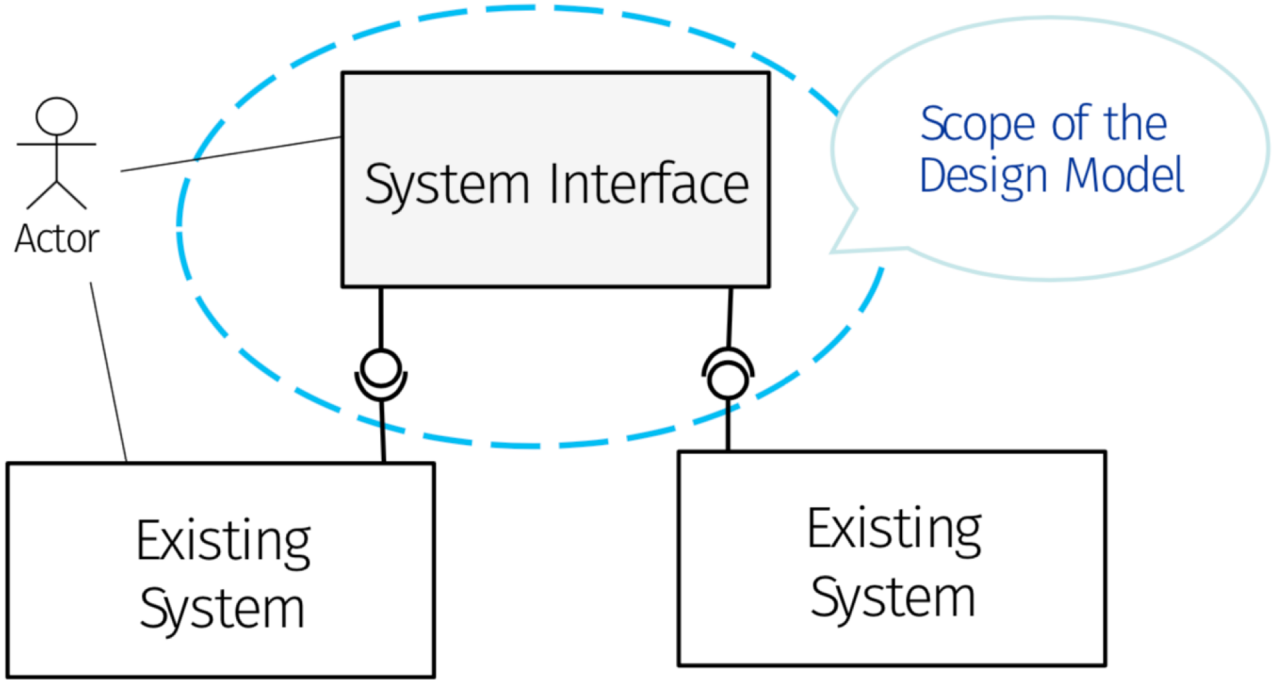
Always design a thing by considering it in its next larger context – a chair in a room, a room in a house, a house in an environment, an environment in a city plan.

— ELIEL SAARINEN

https://quotesondesign.com/eliel-saarinen/



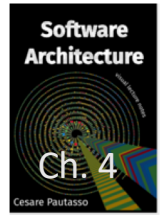
System Context View



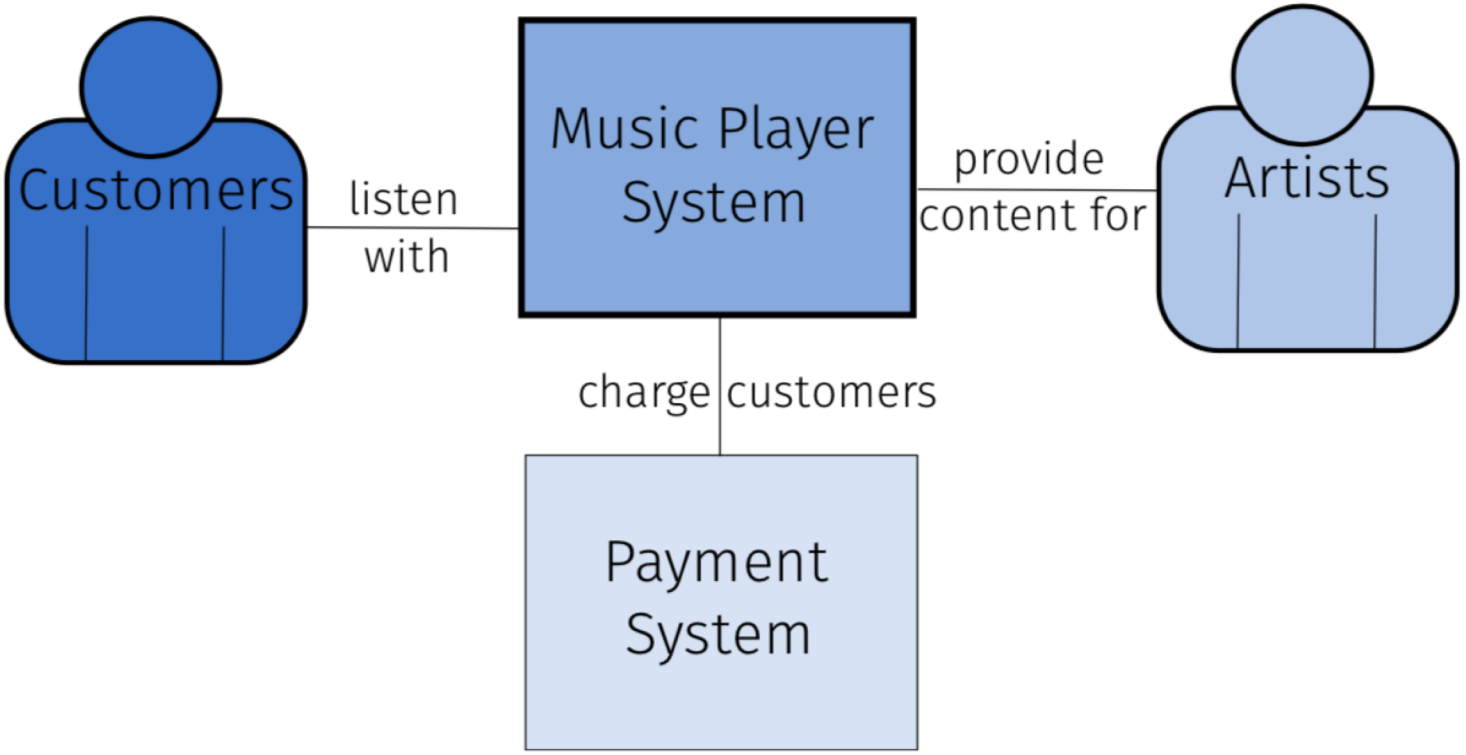
- Distinguish what needs to be built from what already exists and define the dependencies and the integration points

System Context View

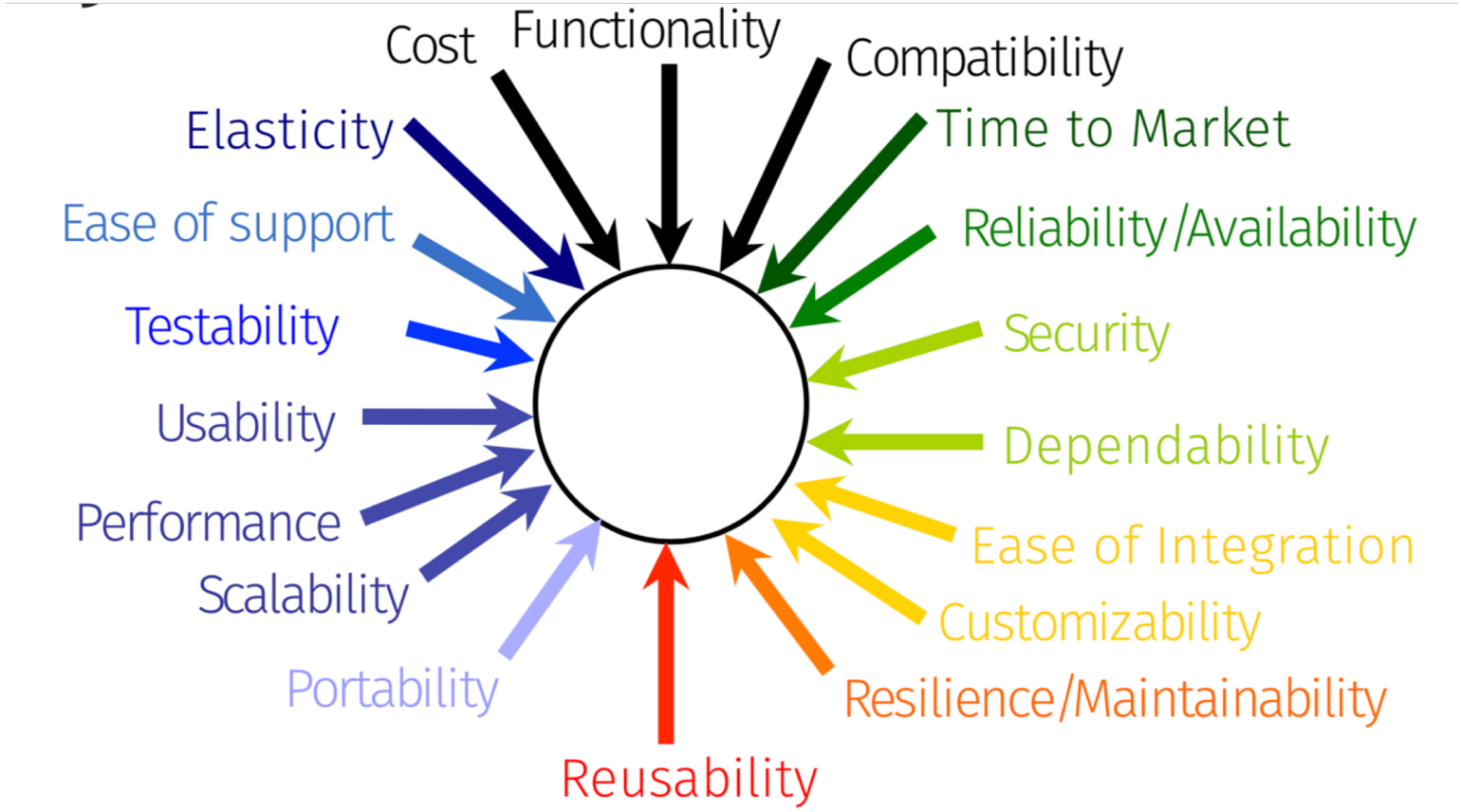
- **User** roles, personas - who do you expect will use the system? Are the users all the same? How many users can share the system at the same time?
- **Dependencies** - which external systems need to be integrated with the system? are there some open API that let other (unknown or known) systems interact with the system?



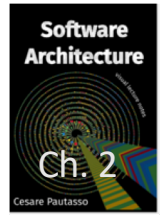
System Context View Example



Quality Attributes



Quality Attributes

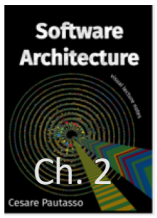


- Desirable properties of a system (under construction)

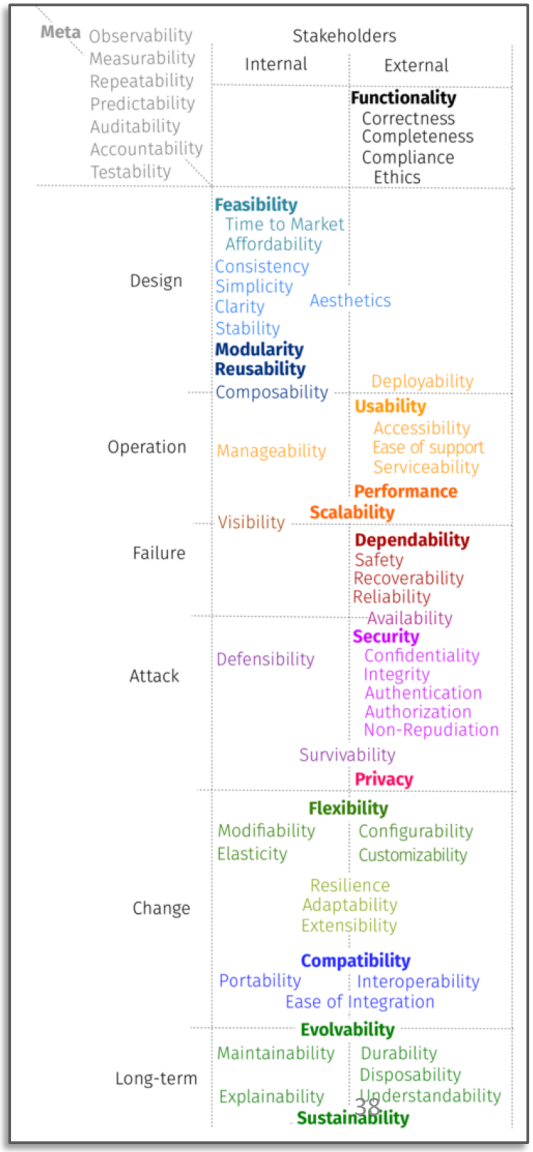
- **External**: fitness for purpose / does it meet stakeholder needs
- **Internal:** fitness for engineering process / can devs work with it
  - Internal attributes indirectly affect many external attributes

- **Static:** structural properties of the design / code
- **Dynamic:** run time properties of the system in action

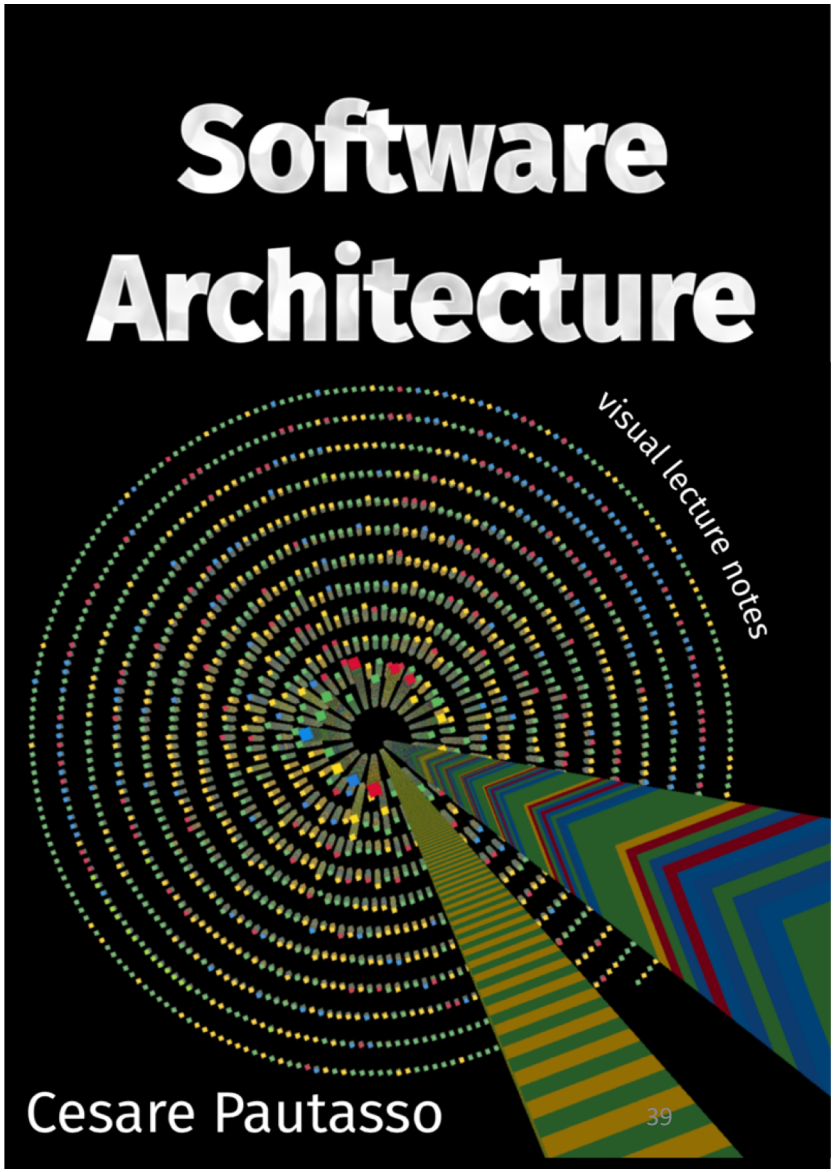
A Catalogue of “ilities”



- **Meta** Measurability, auditability
- **Functionality** Correctness, completeness
- **Design** Modularity, feasibility
- **Operation** Usability, performance, scalability
- **Failure** Recoverability, reliability, availability
- **Attack** Privacy, confidentiality, integrity
- **Change** Flexibility, extensibility, configurability
- **Long-term** Maintainability, explainability



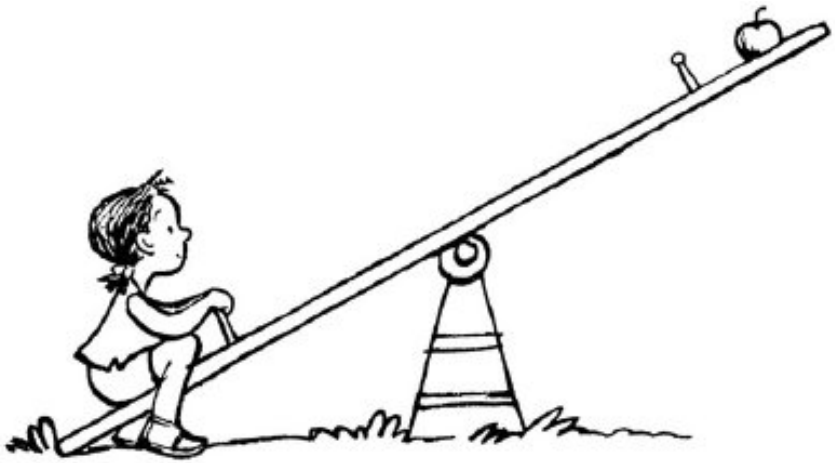
1. Introduction
2. Quality Attributes
3. Definitions
4. Modeling Software Architecture
5. Modularity and Components
6. Reusability and Interfaces
7. Composability and Connectors
8. Compatibility and Coupling
9. Deployability, Portability and Containers
10. Scalability
11. Availability and Services
12. Flexibility and Microservices



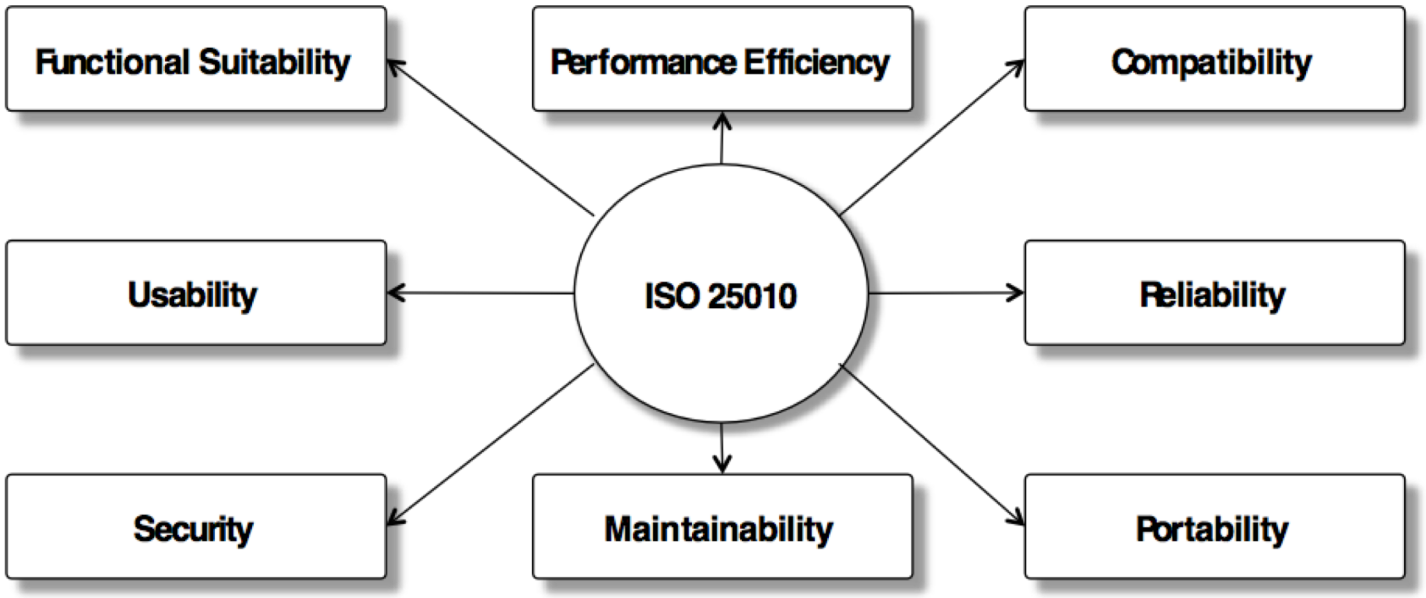
Managing “ility” tradeoffs

- Privacy vs usability
- Modularity vs time-to-market
- Availability vs configurability
- Extensibility vs integrity
- Performance vs interoperability
- Performance vs confidentiality
- …

The architect needs to understand which attributes must be optimized, and which ones can be sacrificed



ISO Software Quality Characteristics



Robust

- Traditional IT
- Failure prevention
- Planning & Verification
- Infrastructure based

Resilient

- Distributed Application
- Failure recovery
- Redundancy & Autom.
- Application-based

Antifragile

- Self-healing System
- Always failing
- Design for Failure
- System-based

Nassim Taleb: Antifragile

Some things benefit from shocks;

they thrive and grow when exposed to volatility, randomness, disorder, and stressors and love adventure, risk, and uncertainty.

Yet, in spite of the ubiquity of the phenomenon, there is no word for the exact opposite of fragile.

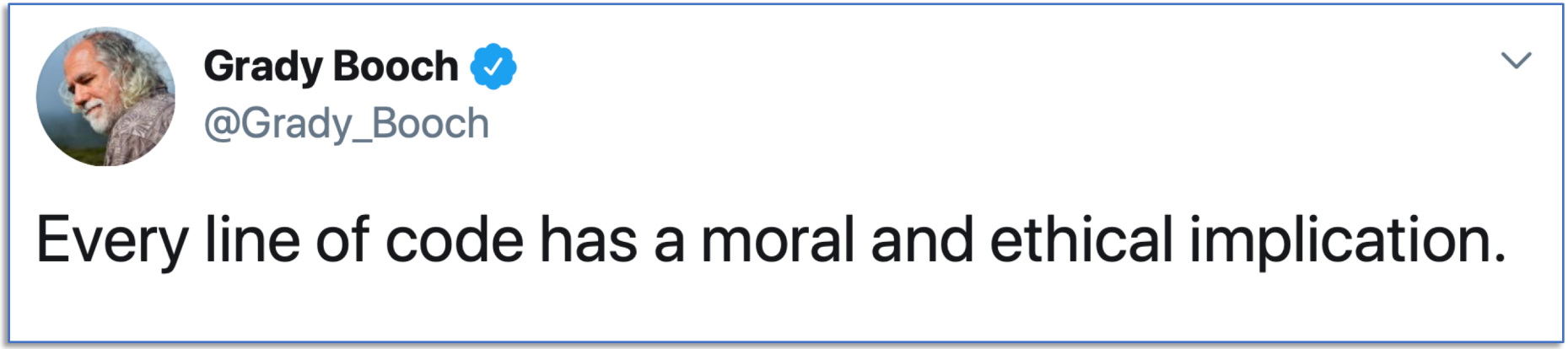
Let us call it antifragile.

Antifragility is beyond resilience or robustness.

The resilient resists shocks and stays the same;

the antifragile gets better

The Ethical Software Architect



1

Ethics

Well-founded standards of right and wrong

that prescribe what humans ought to do,

usually in terms of rights, obligations, benefits to society, fairness, or specific virtues.

The continuous effort of studying

our own moral beliefs and our moral conduct,

and striving to ensure that we, and the institutions we help to shape,

live up to standards that are reasonable and solidly-based

https://www.scu.edu/ethics/ethics-resources/ethical-decision-making/what-is-ethics/

ACM Code of Ethics and Professional Conduct

Preamble

Computing professionals' actions change the world. To act responsibly, they should reflect upon the wider impacts of their work, consistently supporting the public good. The ACM Code of Ethics and Professional Conduct ("the Code") expresses the conscience of the profession.

The Code is designed to inspire and guide the ethical conduct of all computing professionals, including current and aspiring practitioners, instructors, students, influencers, and anyone who uses computing technology in an impactful way. Additionally, the Code serves as a basis for remediation when violations occur. The Code includes principles formulated as statements of responsibility, based on the understanding that the public good is always the primary consideration. Each principle is supplemented by guidelines, which provide explanations to assist computing professionals in understanding and applying the principle.

Section 1 outlines fundamental ethical principles that form the basis for the remainder of the Code. Section 2 addresses additional, more specific considerations of professional responsibility. Section 3 guides individuals who have a leadership role, whether in the workplace or in a volunteer professional capacity. Commitment to ethical conduct is required of every ACM member, and principles involving compliance with the Code are given in Section 4.

On This Page

The Ethical Software Architect?

Ethics of the Product

- “First, do no harm”
- Legal limits
- Fair pricing
- Dual use
- Human dignity
- Human control
- ...

Ethics of the Construction

- Bias in data sets
- Accessibility
- Resource usage, energy consumption
- Code “reuse”
- Tracking and privacy
- Code of conduct, inclusion
- ...

Essay E1: Product Vision

1. Characterization of what the project aims to achieve
2. The key domain concepts (underlying domain model)
3. The system's main capabilities (e.g. use cases), visible to (end) user
4. The current/future (external) context in which the system operates
5. The stakeholders involved in the project, and what they need from the system so that it is beneficial to them
6. The key quality attributes the system must meet
7. A product roadmap for the upcoming years
8. Ethical considerations of the system and its construction process