

TU Delft IN4315: Software Architecture Lecture 2: The Vision (E1) and the System (E2)

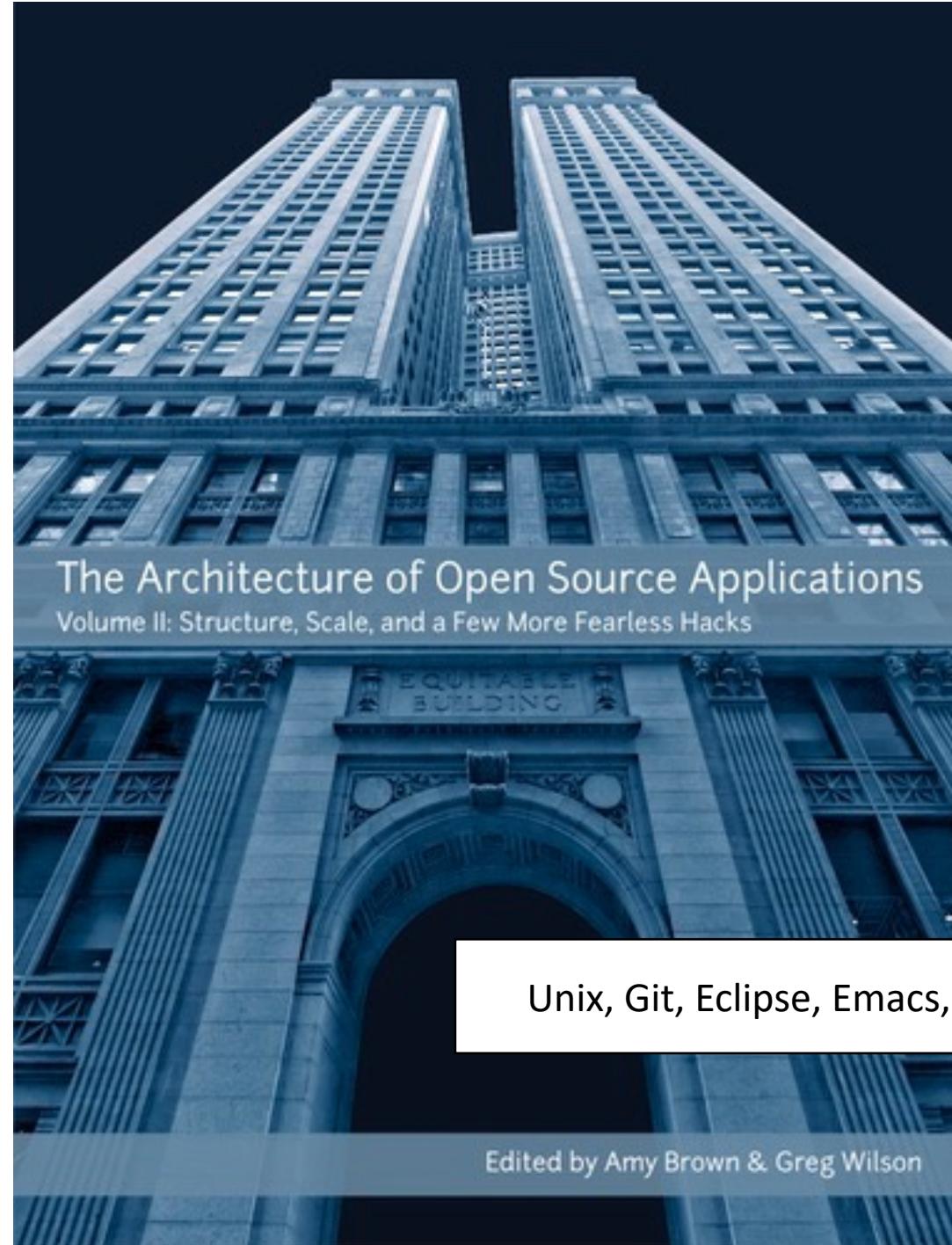
Arie van Deursen

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!

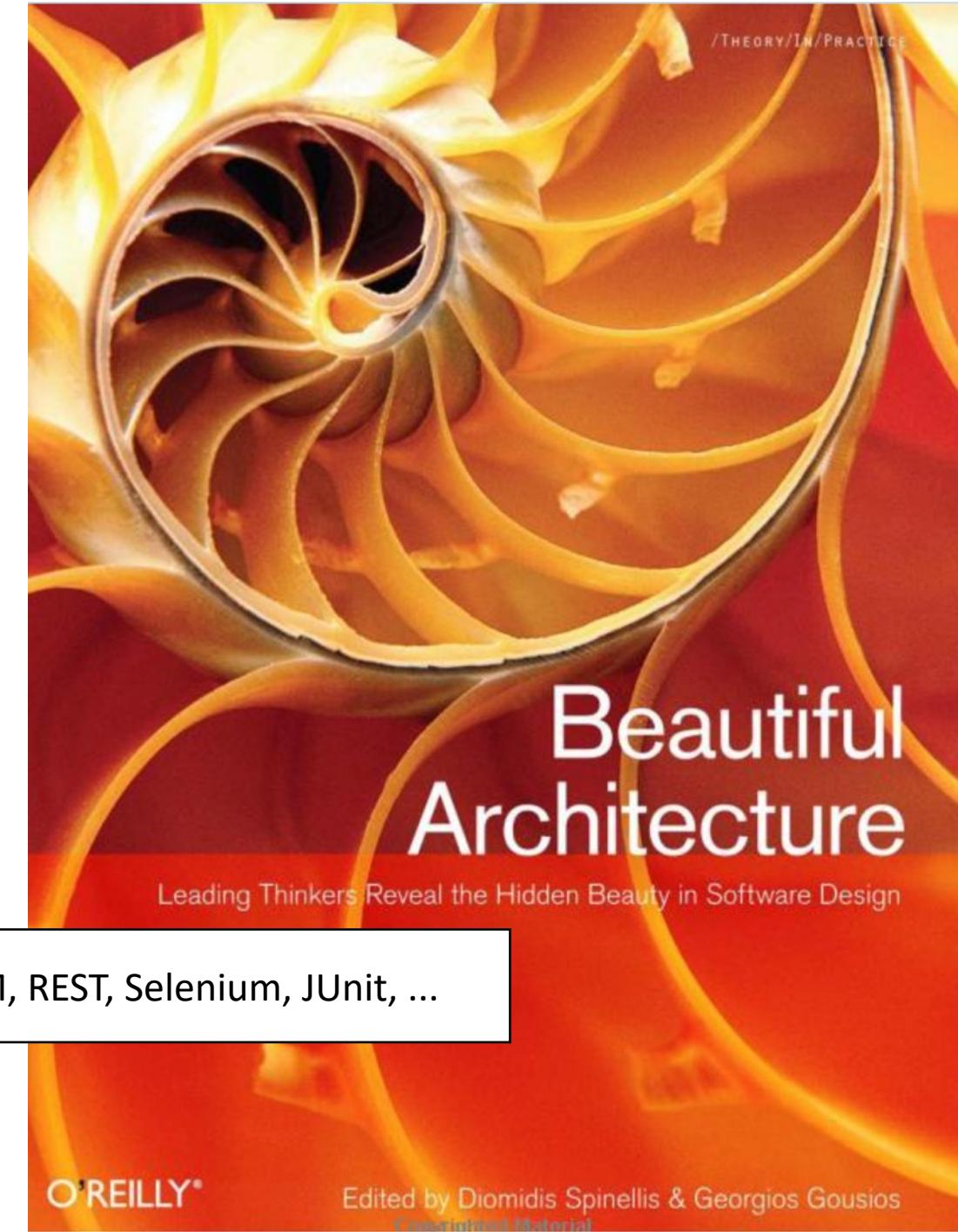


The Architecture of Open Source Applications
Volume II: Structure, Scale, and a Few More Fearless Hacks

EQUITABLE
BUILDING

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

Edited by Amy Brown & Greg Wilson



/ THEORY/IN PRACTICE
O'REILLY®
Edited by Diomidis Spinellis & Georgios Gousios
Copyright Material

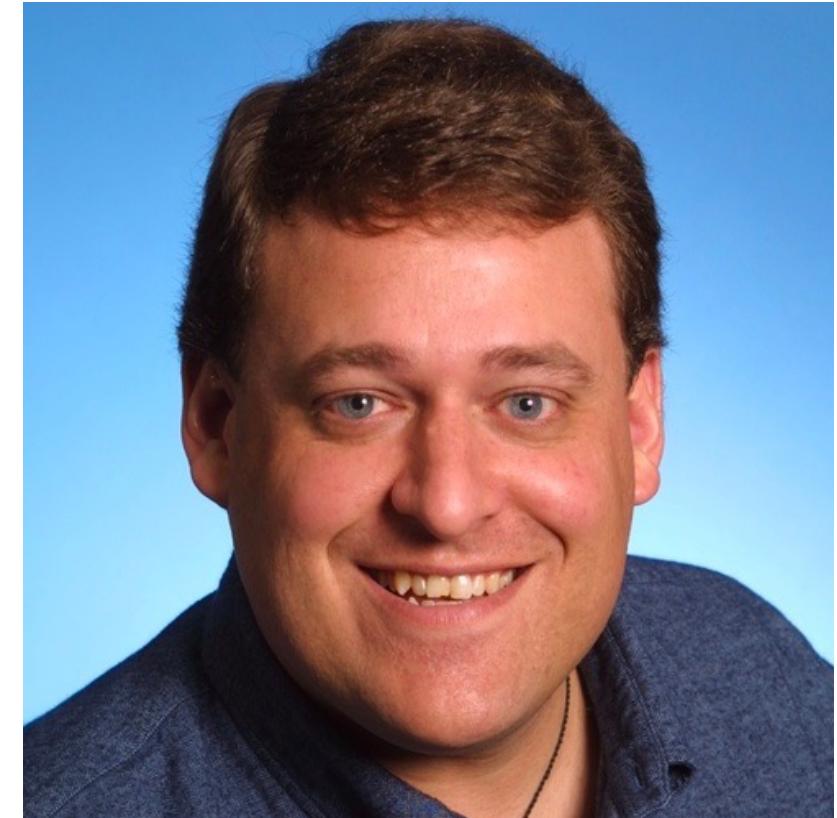
The Architecture of a System (IEEE):

- 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.

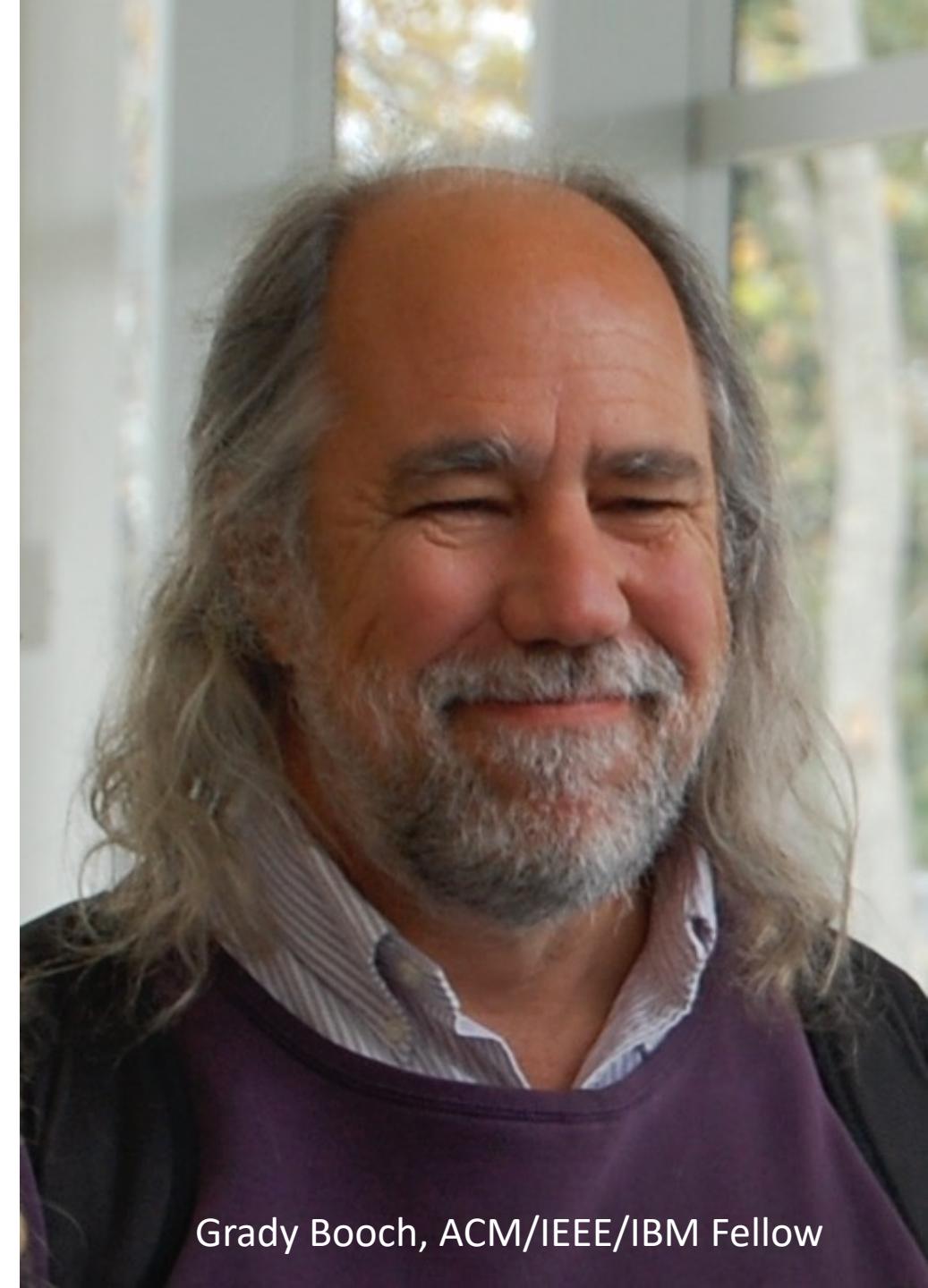
Constraints: Focus on what is and is *not* allowed



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

“Principal Design Decisions”

Aspects:

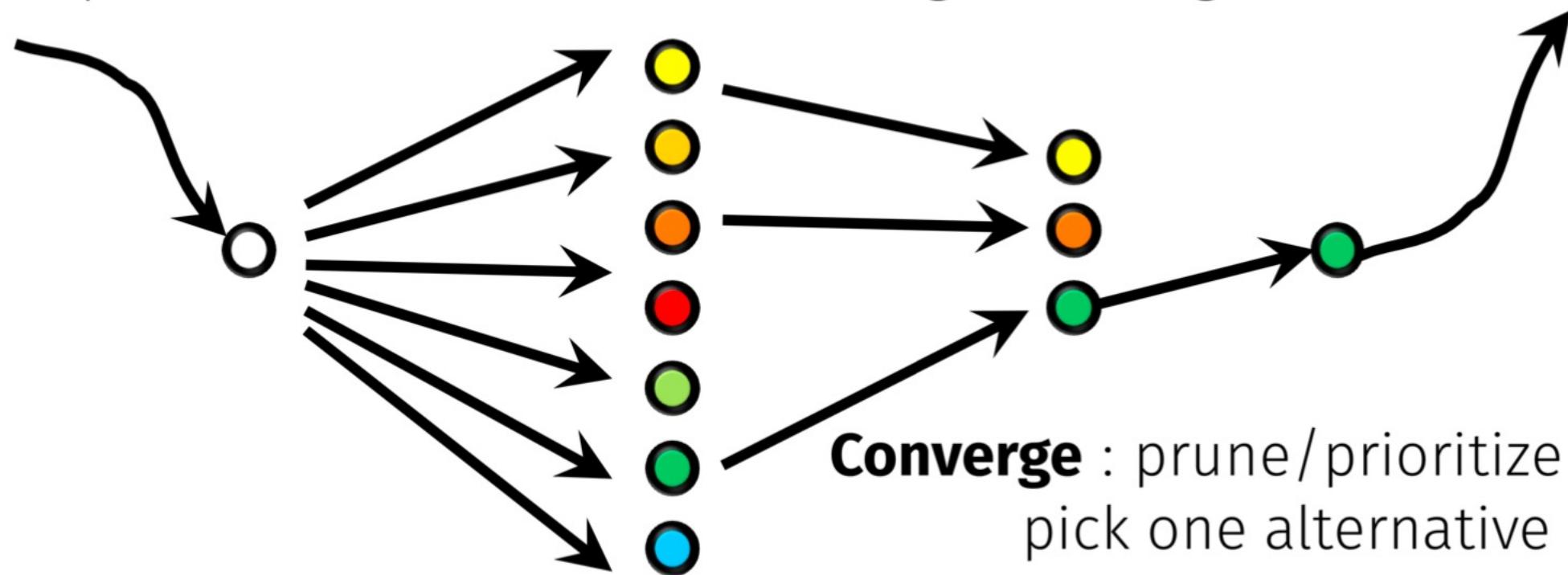
- Structure
- Behavior
- Interaction
- Deployment
- User Interface
- Implementation

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.

Decision Making Phases

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



Design issue

Design alternatives

Design decision, with rationale

Converge : prune/prioritize pick one alternative

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

Essay 1: The Product in its Context

- What do we want?
 - The guiding principles / product vision
 - Key scenarios illustrating the usage
 - Analysis of key domain concepts
- What do we need from the outside world?
 - The system context and external dependencies
- Who do we want it for?
 - Users and other stakeholders
- How good should it be?
 - Key quality attributes

Answering these questions
imposes order
on the problem:

This involves design
already

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.

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.

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.

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.



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.

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.

The “Domain Model”

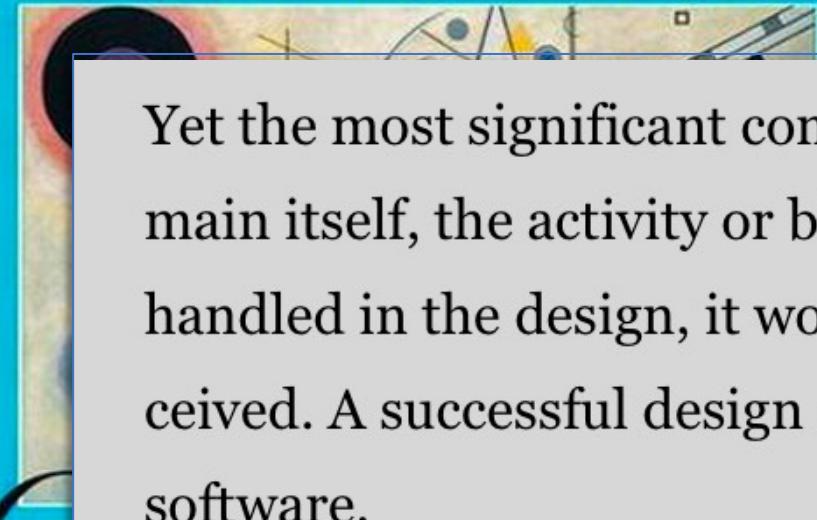
- Refutable truths about the real-world
- Outside your control
- Your system will be evaluated against it
- Architecturally significant requirements
- Problem domain description:
 - Information (invariants, navigation, snapshots)
- Functionality (use-case scenarios, feature models)
- Define shared vocabulary and understanding towards your customer, domain expert

Example Domain Model

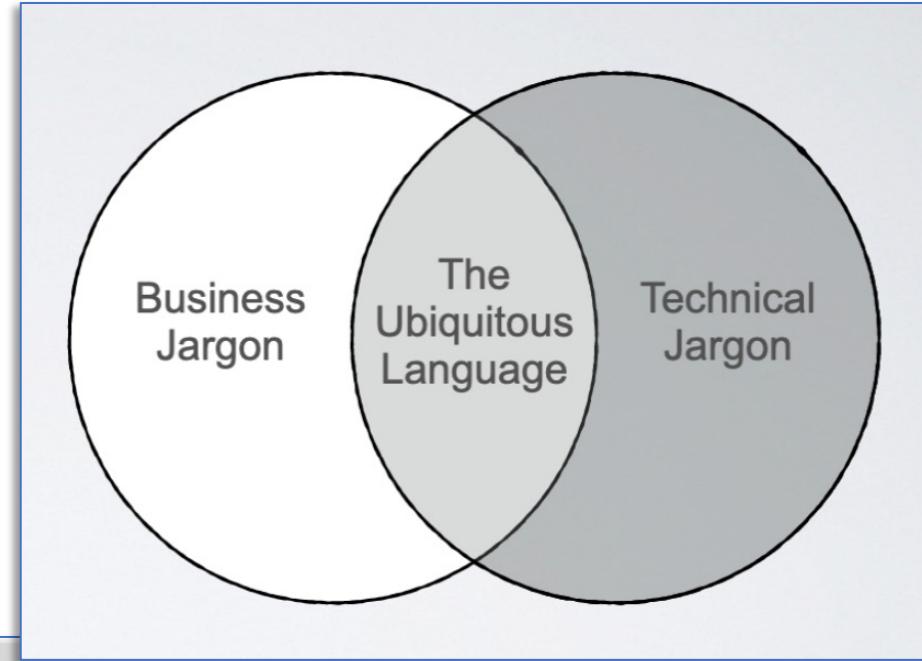
- Music songs are organized in albums
- The same song can be authored by many artists
- Listening to each song costs 0.99 CHF, but short samples can be heard for free
- Songs can be downloaded and also live streamed
- Songs are stored in files of standard MP3 format

Domain-Driven DESIGN

Tackling Complexity in the Heart of Software



Eric Evans
Foreword by Martin Fowler

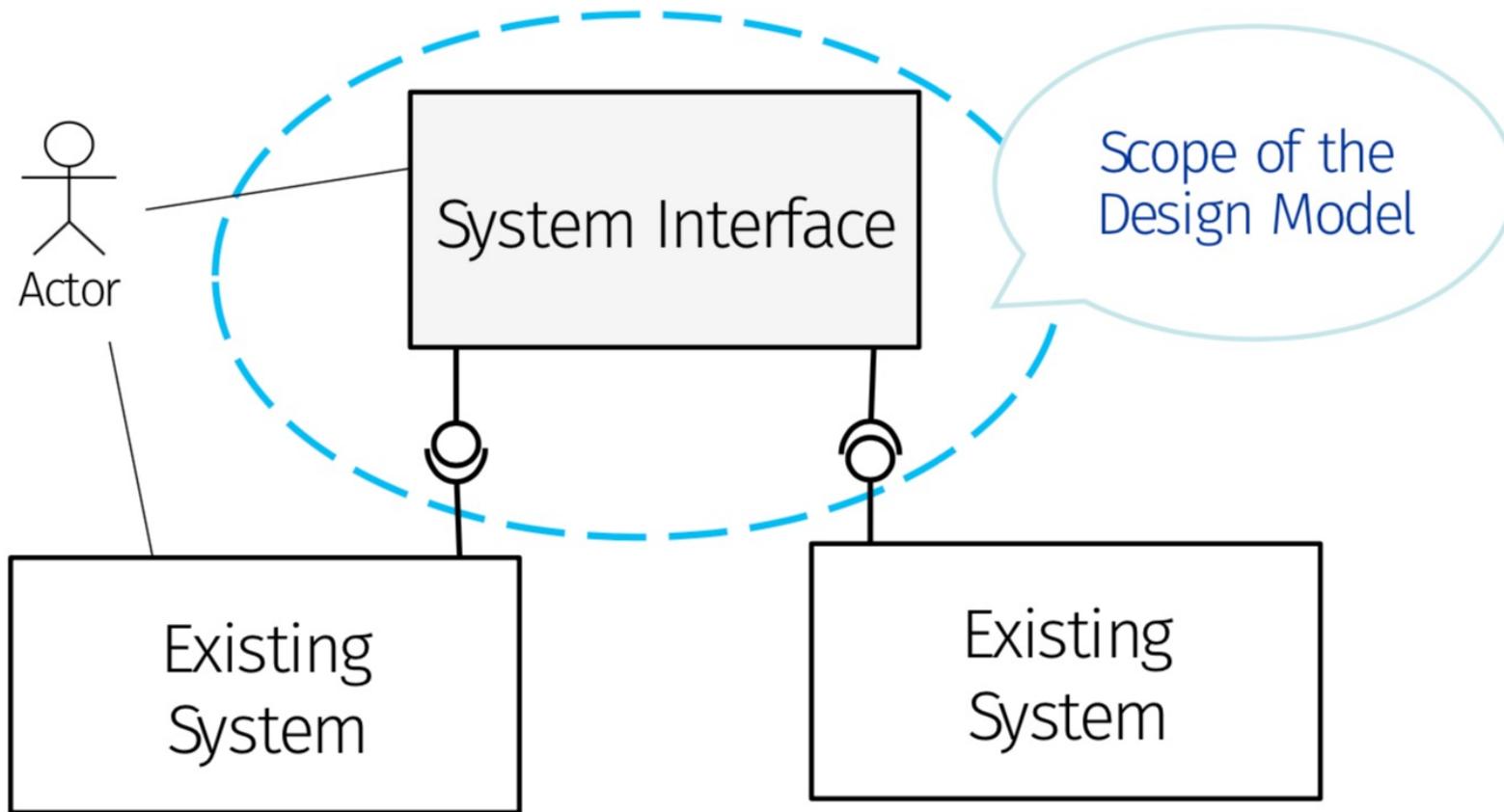


Yet the most significant complexity of many applications is not technical. It is in the domain itself, the activity or business of the user. When this domain complexity is not handled in the design, it won't matter that the infrastructural technology is well conceived. A successful design must systematically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

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

System Context View

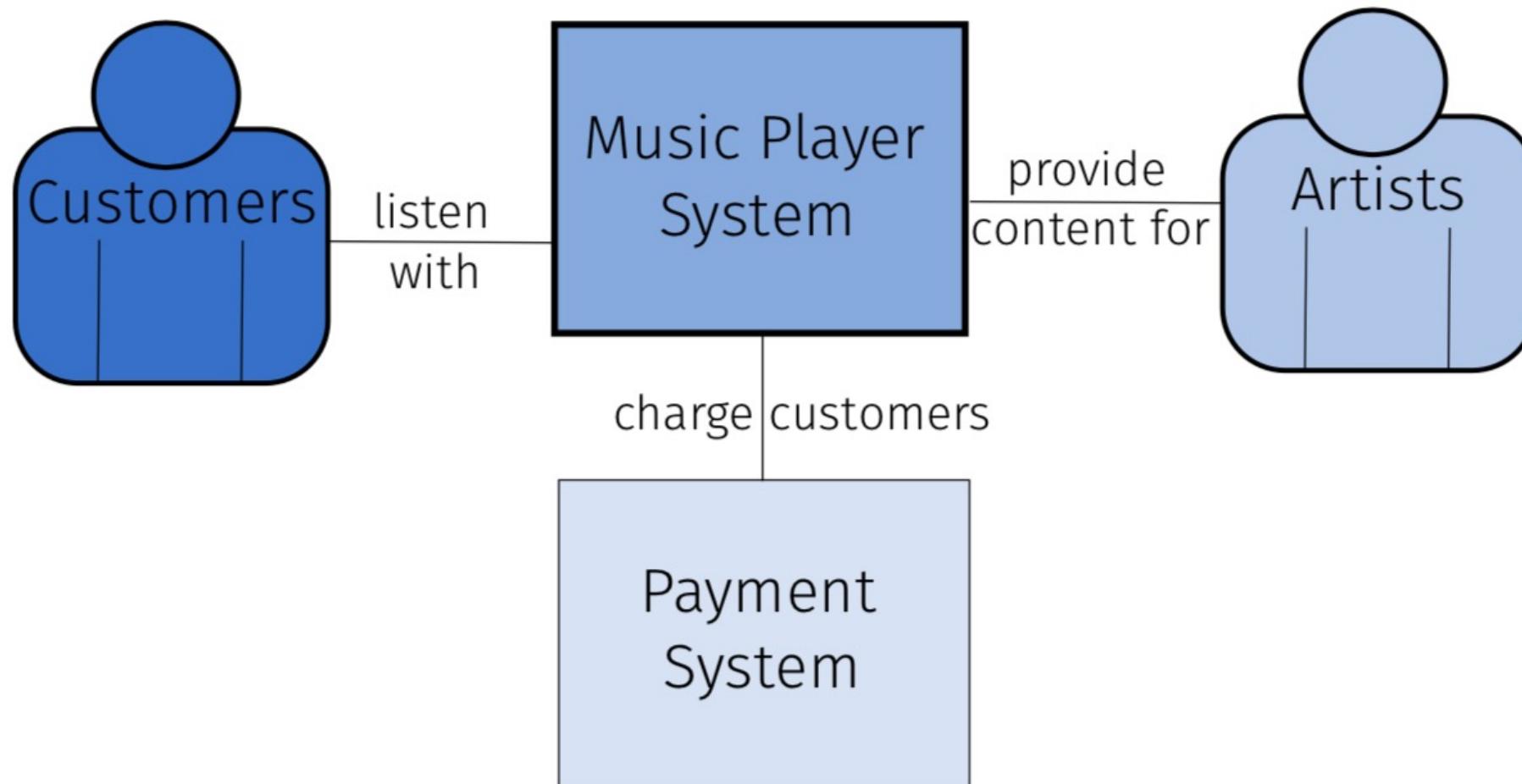


- Distinguish what needs to be built from what already exists and define the dependencies and the integration points

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



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?

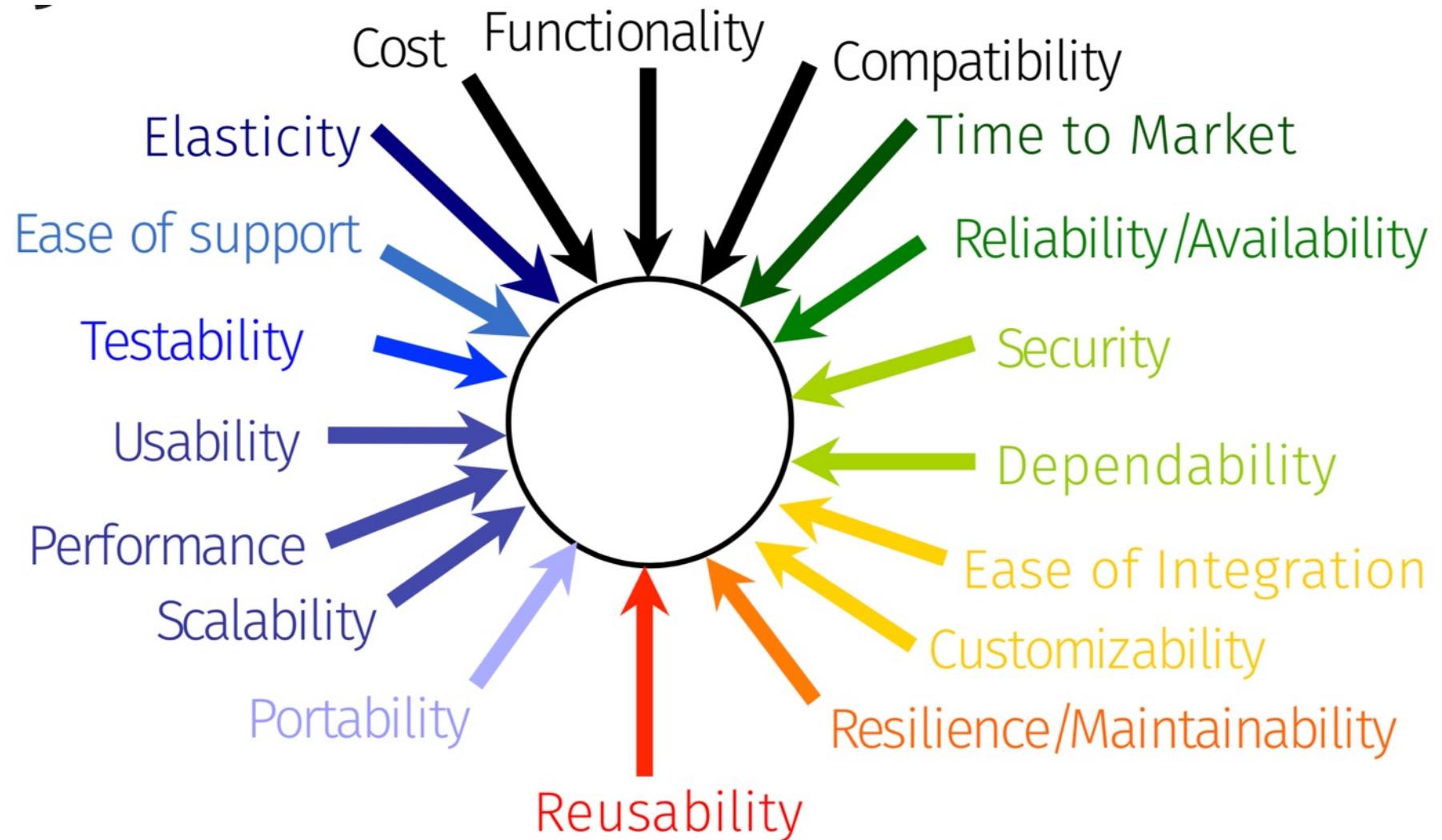
Types of Stakeholders

- Developers, testers, reviewers,...
- End users, API consumers
- Apache, Linux, Eclipse, ... foundations
- Companies relying on open source
- Companies selling open source

Finding Stakeholders

- Open documentation
- Issue tracking system
- Pull request discussions
- License
- ...

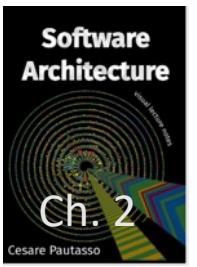
How Good Should the System be?



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”



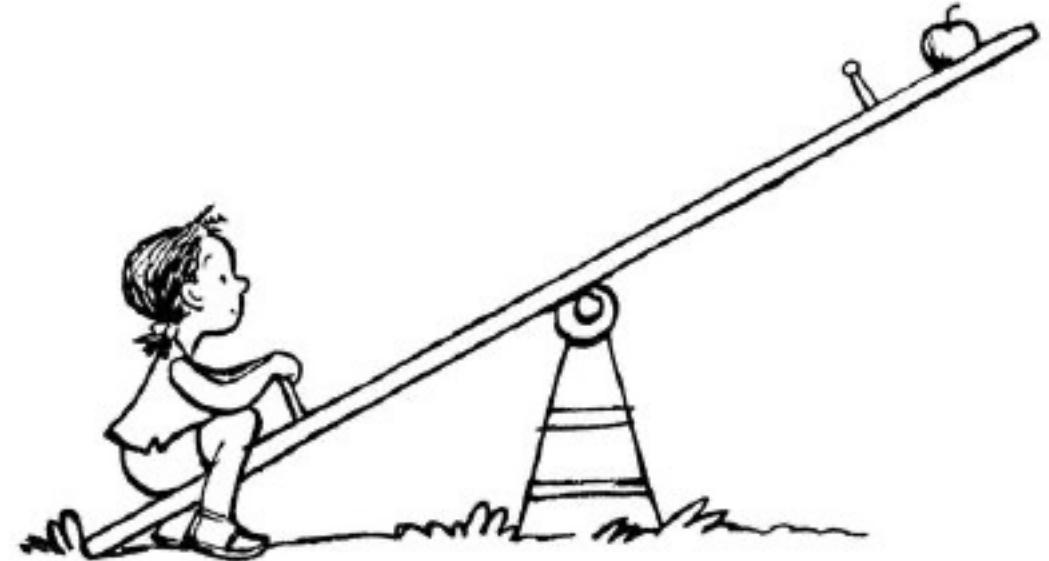
- **Meta** Measurability, auditability
- **Functionality** Correctness, completeness
- **Design** Modularity, reusability
- **Operation** Usability, performance, scalability
- **Failure** Recoverability, reliability, availability
- **Attack** Privacy, confidentiality, integrity
- **Change** Flexibility,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



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



Right and Wrong in Software Architecture



Grady Booch

@Grady_Booch



Every line of code has a moral and ethical implication.

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

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

Preamble

1. GENERAL ETHICAL PRINCIPLES.

1.1 Contribute to society and to human well-being, acknowledging that all people are stakeholders in computing.

1.2 Avoid harm.

1.3 Be honest and trustworthy.

1.4 Be fair and take action not to discriminate.

1.5 Respect the work required to produce new ideas, inventions, creative works, and computing artifacts.

1.6 Respect privacy.

The Ethical Software Architect?

Ethics of the Product

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

Ethics of the Construction

- Bias in data sets
- Accessibility
- Resource usage, energy consumption
- Violating licenses of reused libraries
- Robustness guarantees
- Engineering shortcuts to save costs
- Code of conduct

☰ 77 lines (58 sloc) | 3.26 KB

<>

📄

Raw

Blame

💻

🔗

✍

⌫

Contributor Covenant Code of Conduct

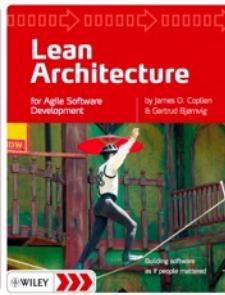
Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to make participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

Our Standards

Examples of behavior that contributes to creating a positive environment include:

Software Architecture is about People

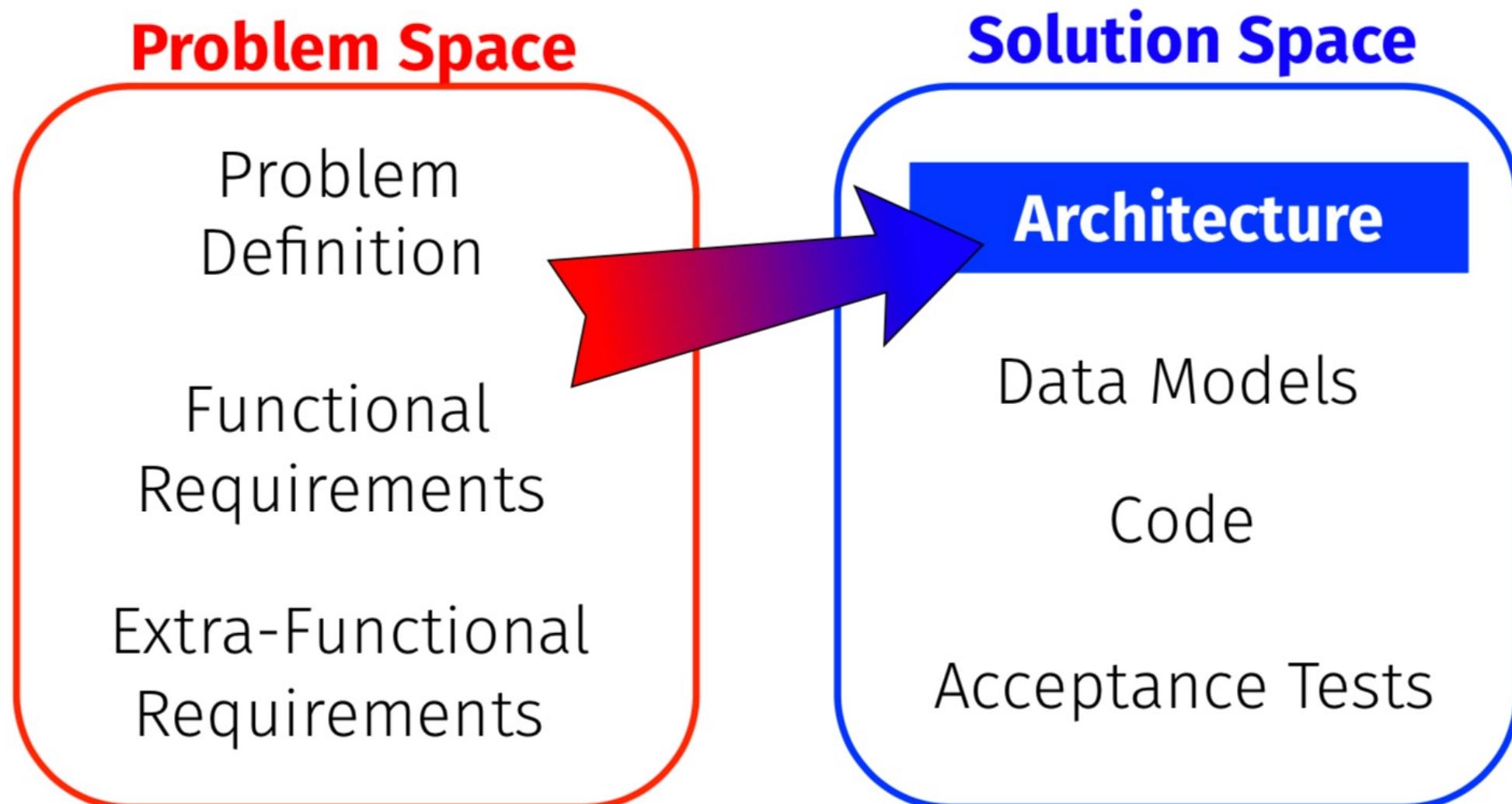


- “Maybe half of software development is about nerd stuff happening at the whiteboard and about typing at the keyboard.”
- “The other half is about people and relationships. ”
- “There are few software team activities where this becomes more obvious than during architecture formulation.”

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

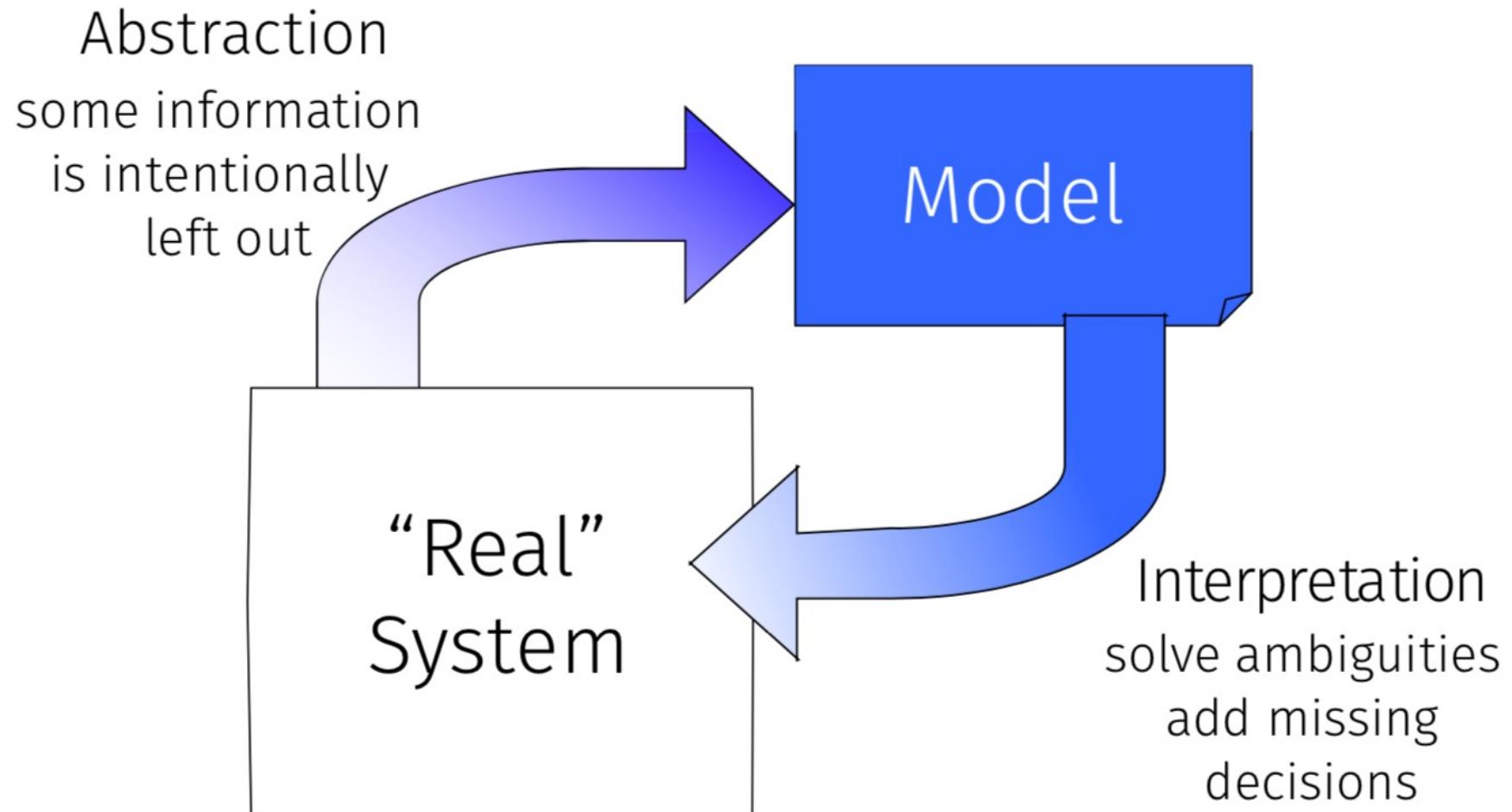
Essay 2: The System's Architecture



From Vision to System – Solution Modeling

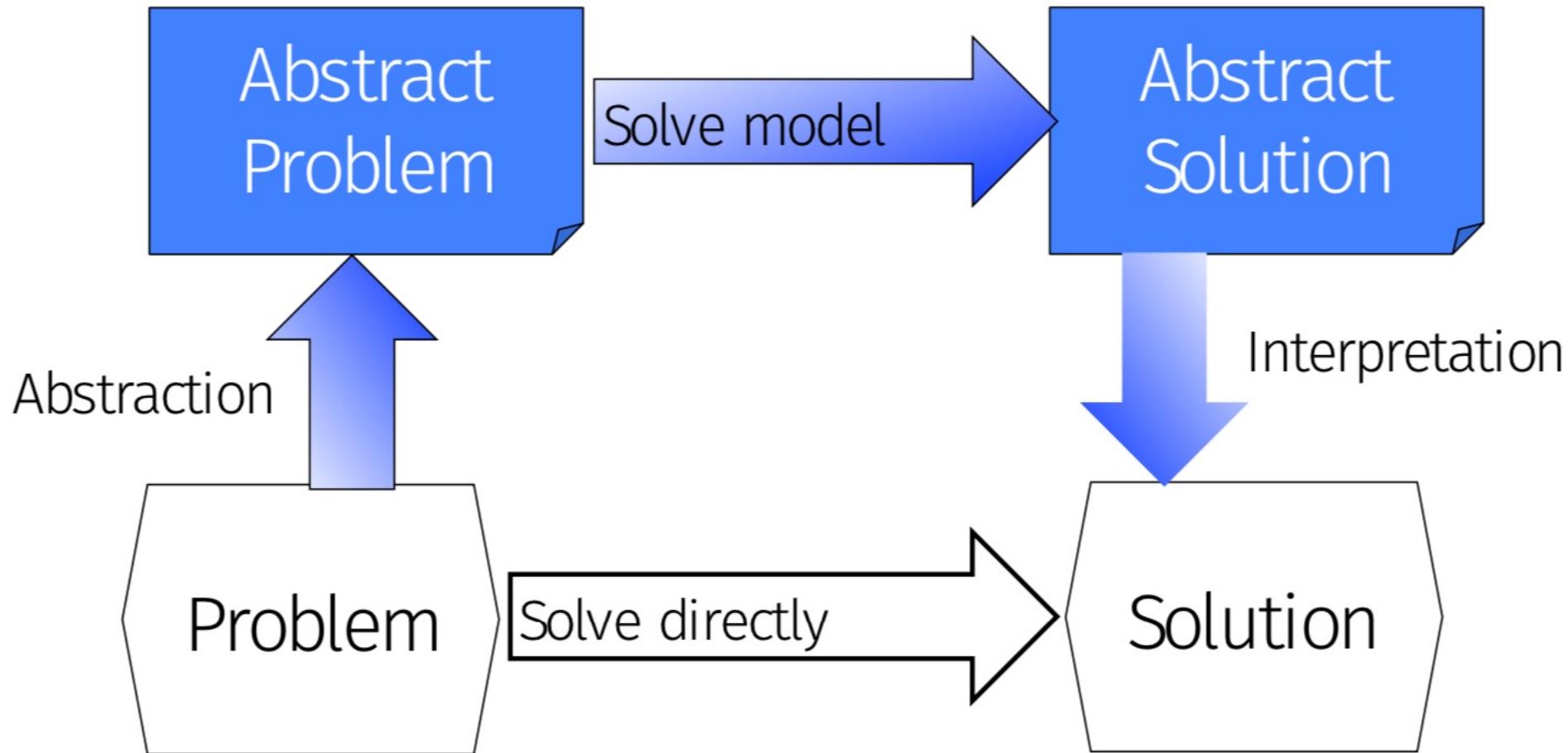
- A system's architecture may be **visualized and represented using models** that are somehow related to the code
- An architectural **model** is an artifact that captures a selection of key design decisions
- Architectural **modeling** is the reification and documentation of those design decisions.
- Every system has an architecture
 - Some architectures are manifest and visible, many others are not

Abstraction and Interpretation



- The architecture models only some interesting aspects of a software system.

Solving Problems with Models



- Abstract models help to find solutions to difficult engineering problems.

Question First, Model Second

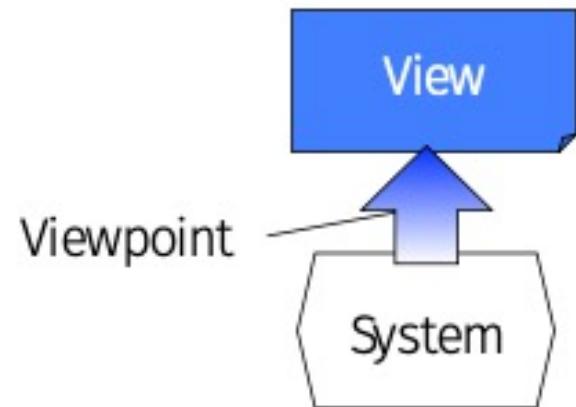
- Different models have different purposes
- Know what questions you want the model to answer before you build it

George Box: All models are wrong, but some are useful

Shneiderman's (visualization) mantra:
Overview first, zoom and filter, details on demand

Organize our Models in “Views”

- No single modeling approach can capture the entire complexity of a software architecture
- Various parts of the architecture (or views) may have to be modeled with a different:
 - Notation
 - Level of detail
 - Target audience
- A **view** is a set of design decisions related by common concerns (the viewpoint)



Which views can you think of?

C4

Context

Containers

Components

Classes

Software Architecture

for Developers

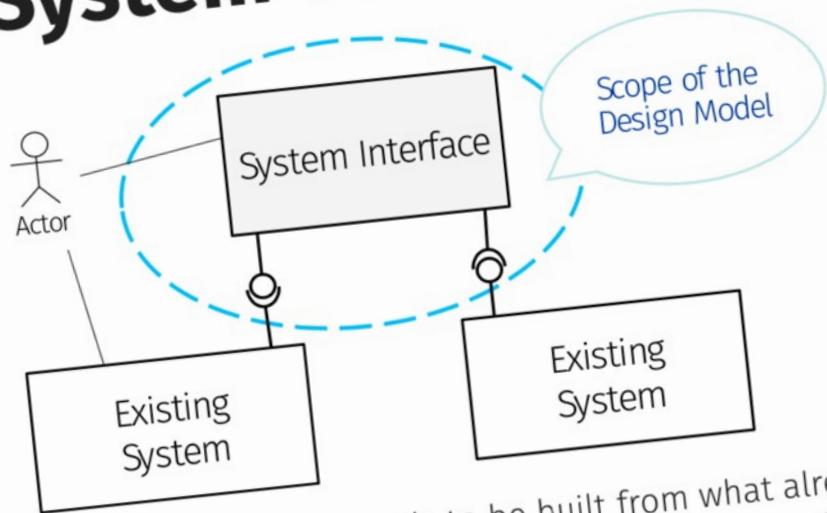
Volume
2

Visualise, document and explore
your software architecture

Simon Brown

<https://c4model.com/>

System Context View

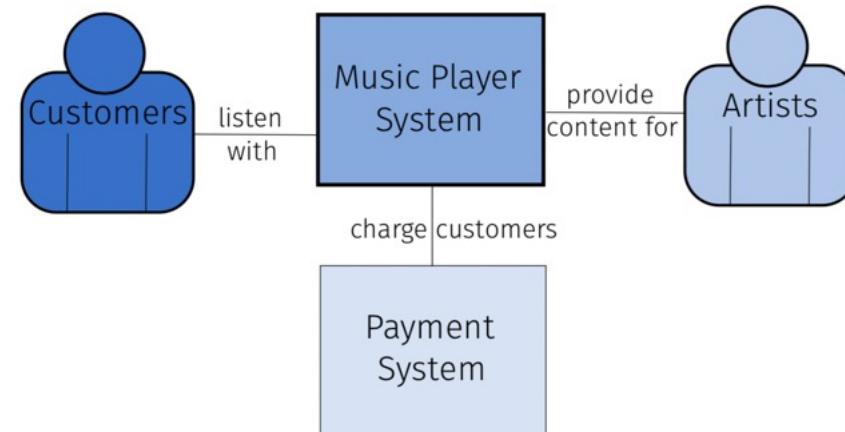


- Distinguish what needs to be built from what already exists
define the dependencies and the integration points

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



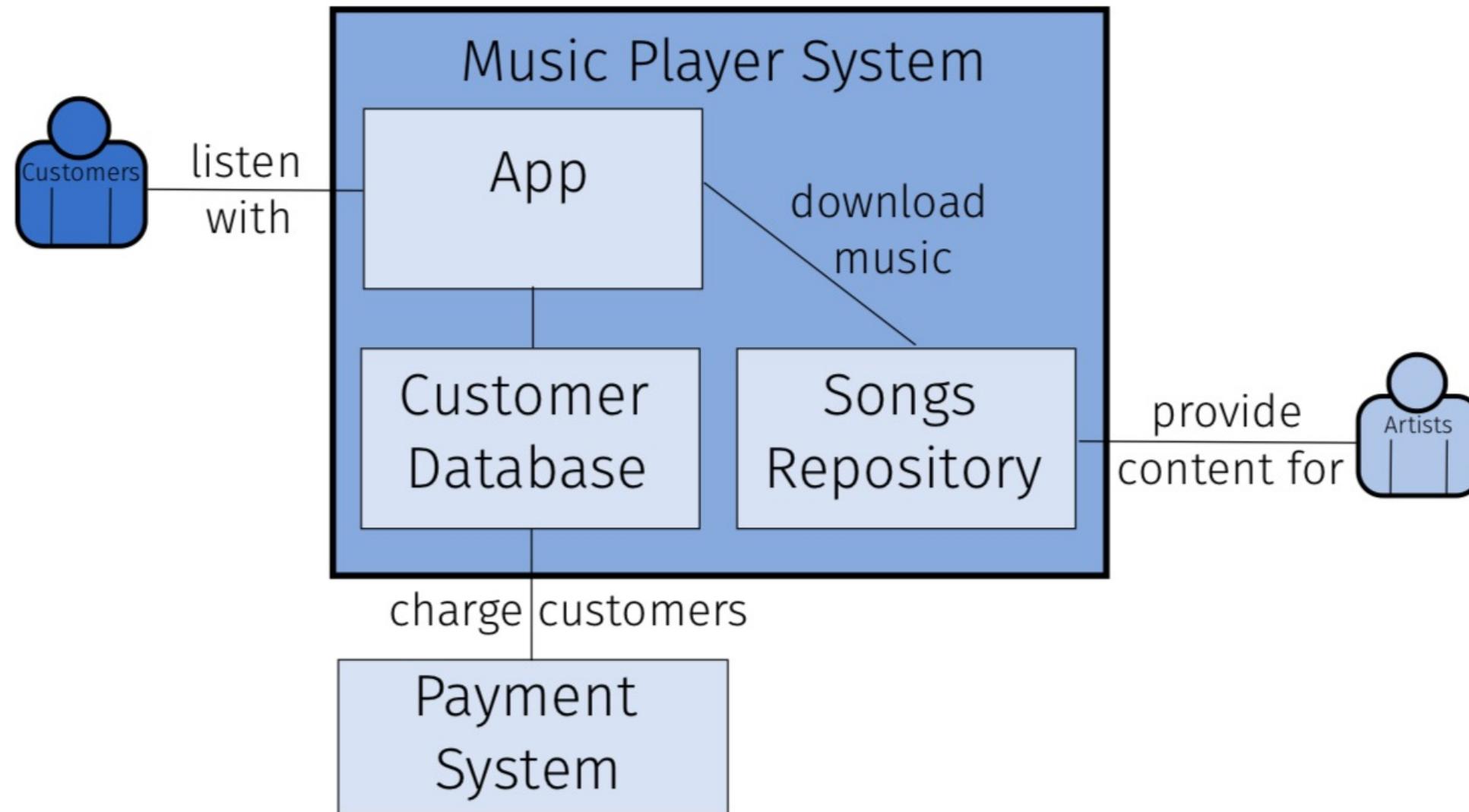
Containers View

- What are the main logical execution environments in which the system can run?
- Containers can be deployed separately and independently evolved
- Container: architectural abstraction (beyond Docker)

Examples:

- Server-side Web application
- Client-side Web application
- Client-side desktop application
- Mobile app
- Server-side console application
- Shell script
- Microservice
- Data store

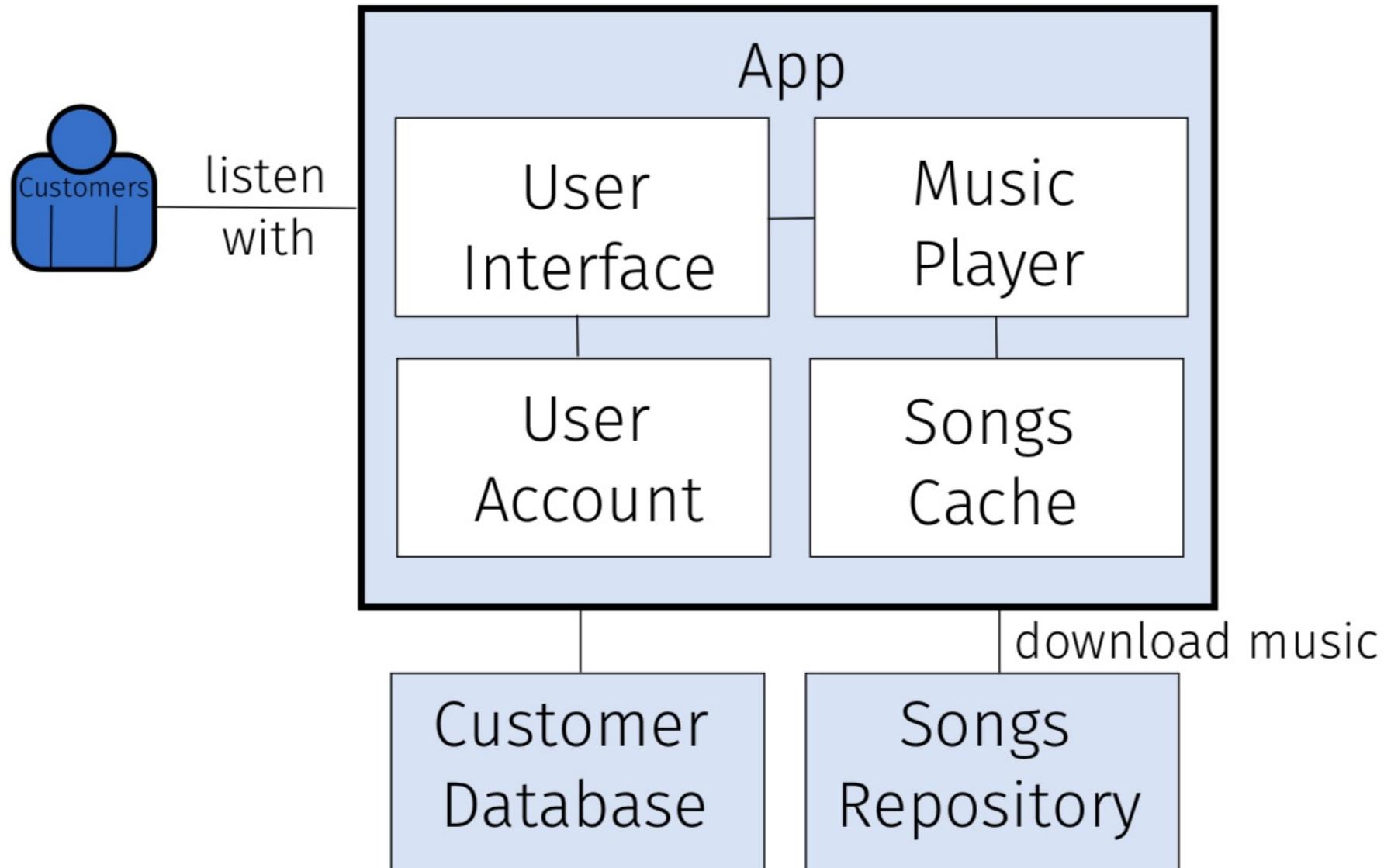
Container View Example



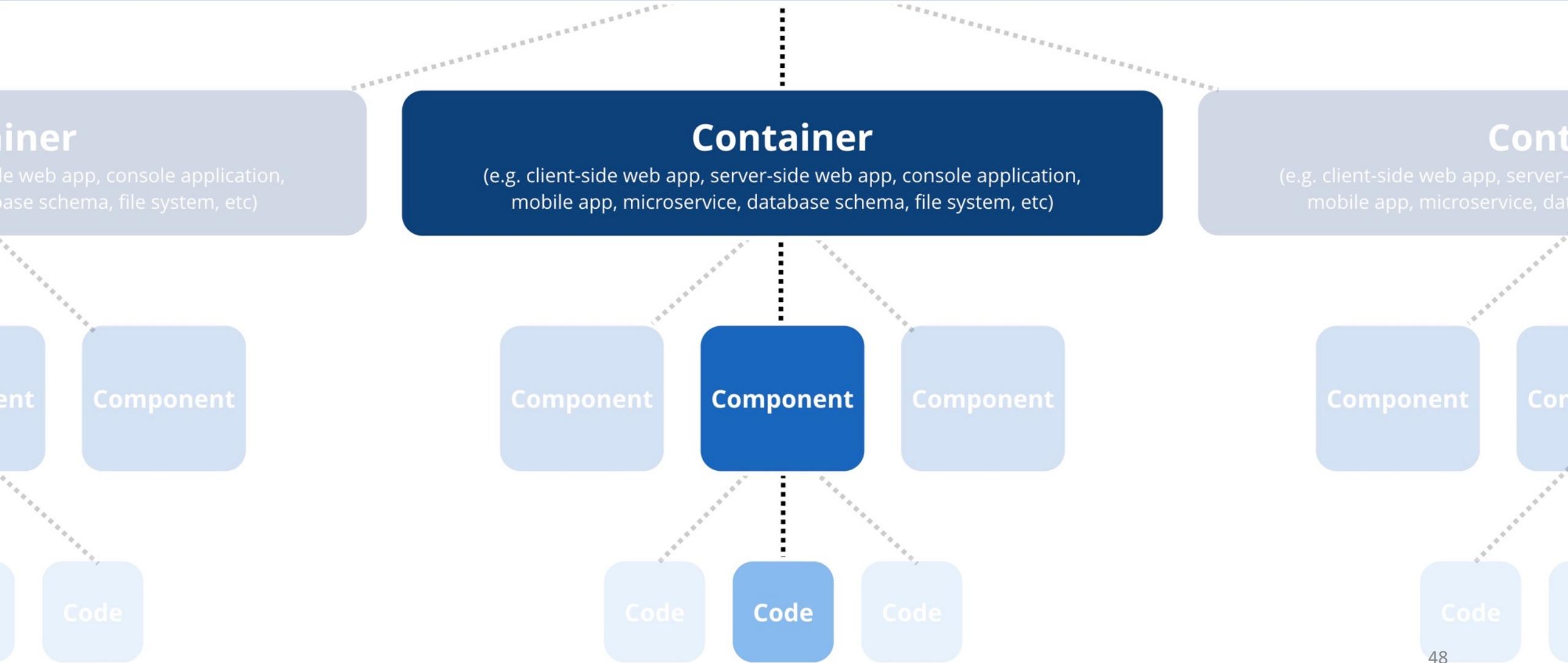
Components View

- What is the structural decomposition of the software with related functionality encapsulated behind a well-defined interface?
- What are the dependencies between components?
- Are there shared components that will be deployed in multiple containers?
- What is the technology used to build the components?
(programming languages, framework decisions)

Components View Example

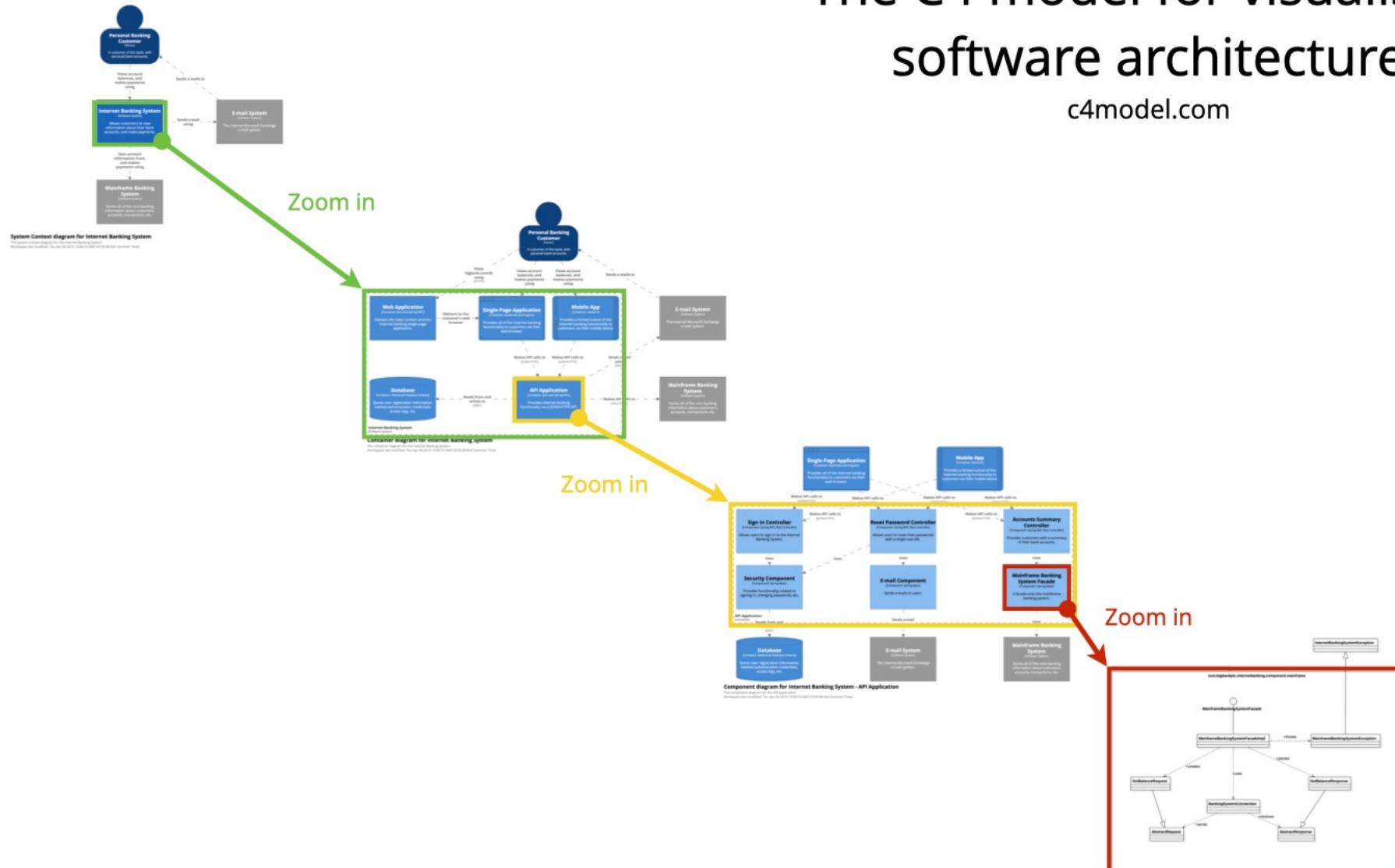


Software System



The C4 model for visualising software architecture

c4model.com



Level 1
Context

Level 2
Containers

Level 3
Components

Level 4
Code



System Context

The system plus users
and system dependencies



Containers

The overall shape of the architecture
and technology choices



Components

Logical components and their
interactions within a container



Classes

Component or pattern
implementation details

**Overview
first**

**Zoom and
filter**

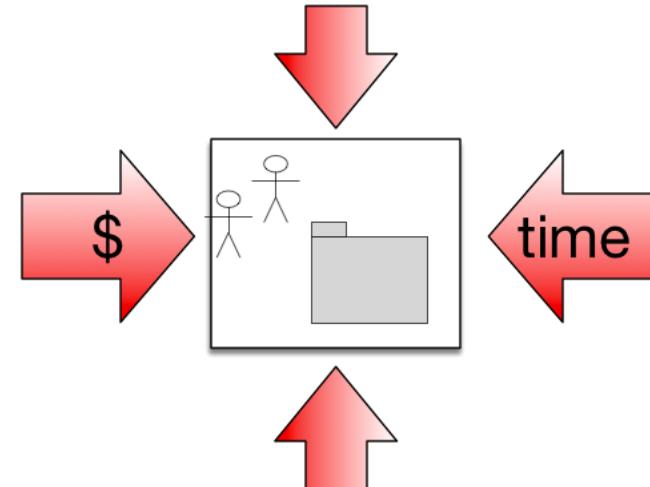
**Details
on demand**

arc42.org: A Template for Architecture Communication and Documentation

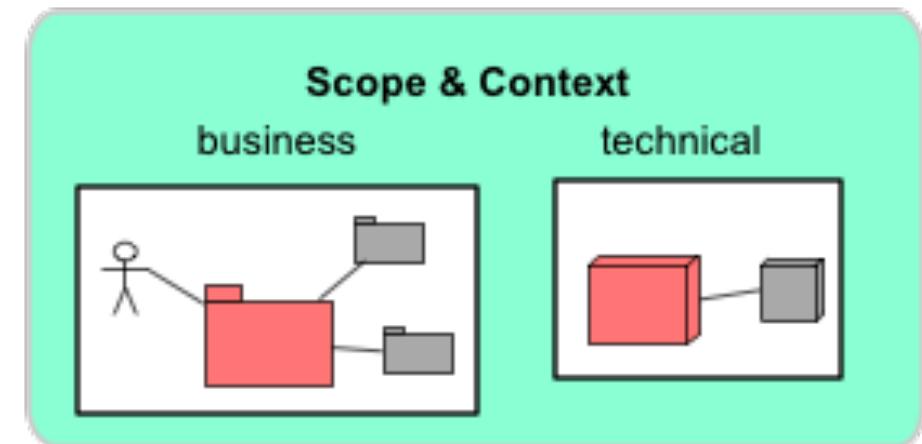
1. Introduction and Goals



2. Constraints

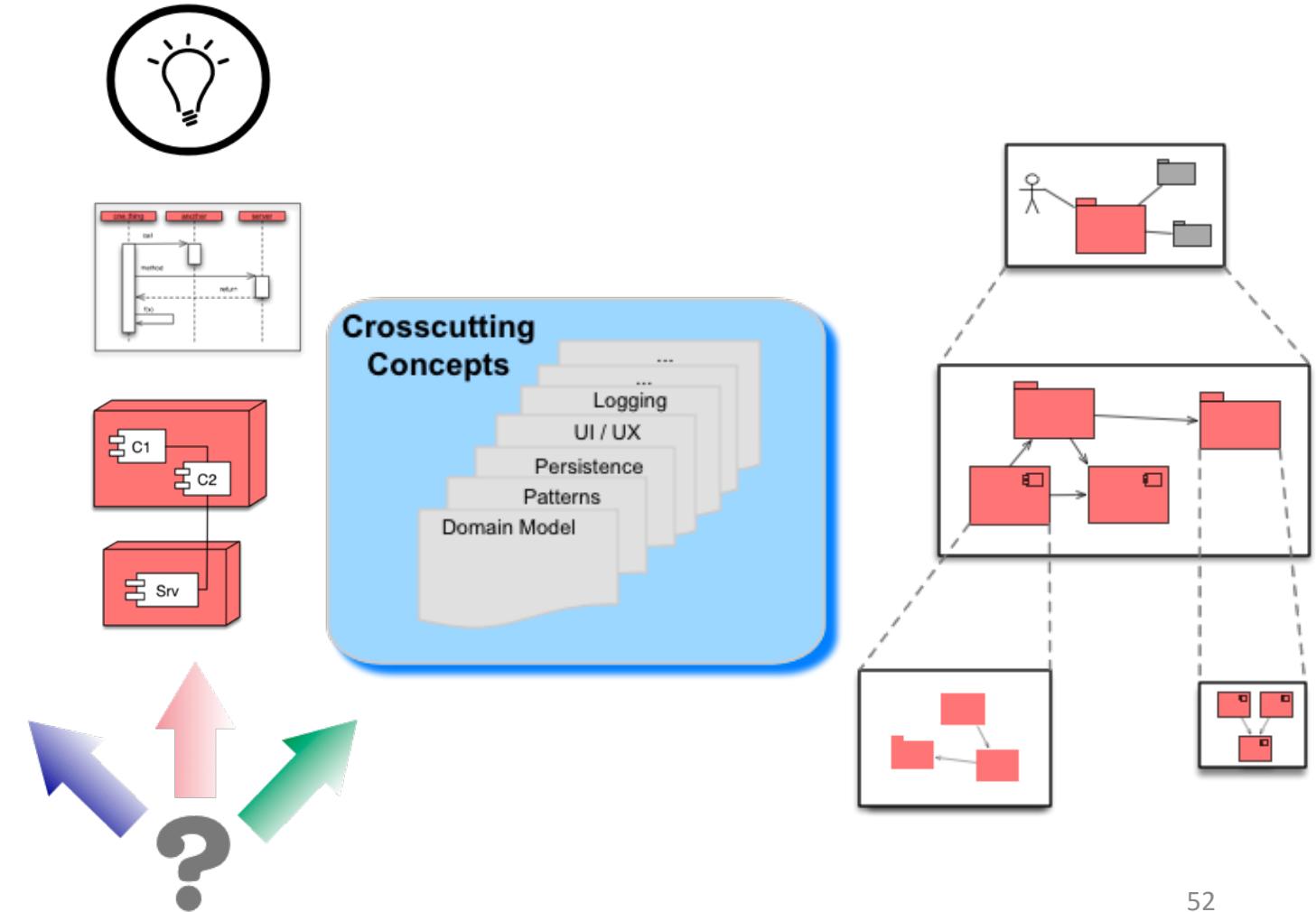


3. Context and Scope



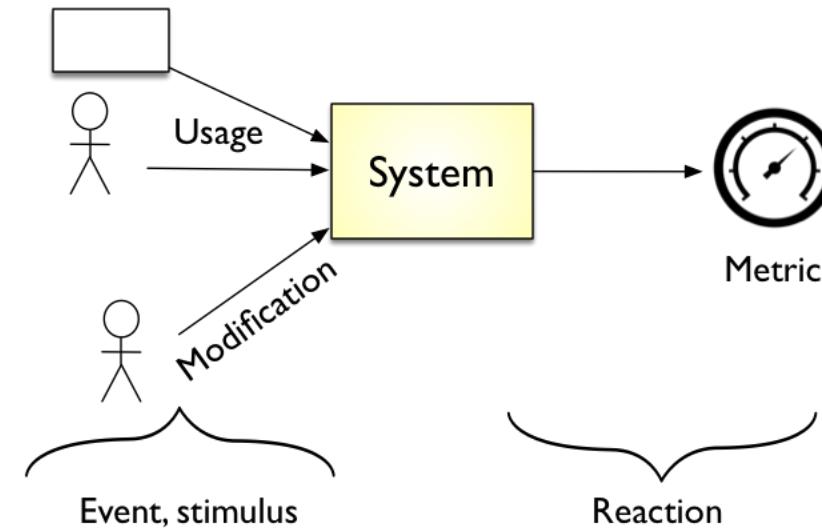
The arc42.org Template for Architecture Communication and Documentation

4. Solution strategy
5. Building block view
6. Run time view
7. Deployment view
8. Crosscutting concepts
9. Architectural decisions



The arc42.org Template for Architecture Communication and Documentation

10. Quality Requirements



11. Risks and Technical Debt



A screenshot of a web browser showing the arc42 Documentation page. The address bar contains the URL docs.arc42.org/keywords/. The page features the arc42 logo (a blue stylized 'arc' with a red '42' inside a circle) and the text "arc42 Documentation" followed by "139 tips how to use the arc42 template." at the bottom.

Essay 2: The System's Architecture

1. The main architectural style or patterns applied (if relevant), such as layering or model-view-controller architectures.
2. Containers view: The main execution environments, if applicable, as used to deploy the system.
3. Components view: Structural decomposition into components with explicit interfaces, and their inter-dependencies
4. Connectors view: Main types of connectors used between components / containers.
5. Development view, covering the system decomposition and the main modules and their dependencies, as embodied in the source code.
6. Run time view, indicating how components interact at run time to realize key scenarios, including typical run time dependencies
7. How the architecture realizes key quality attributes, and how potential trade-offs between them have been resolved.
8. API design principles applied