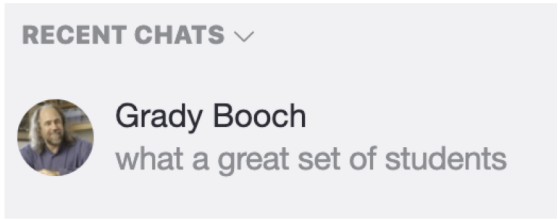
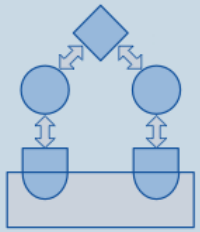
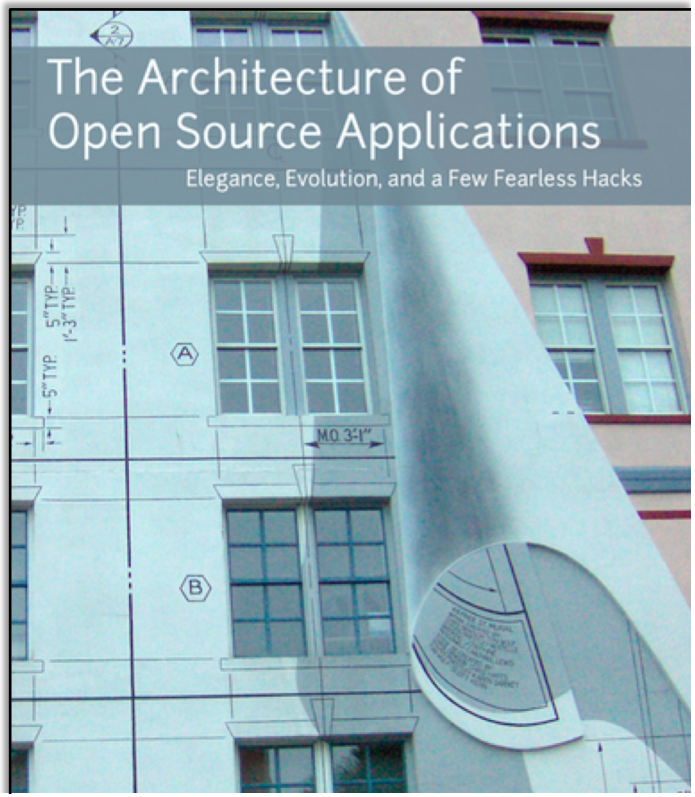
Documenting & Communicating Software Architectures

Arie van Deursen

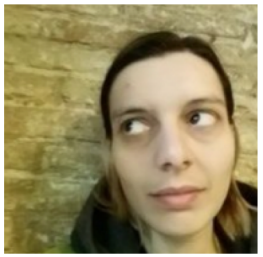




The Architecture of Open Source Applications

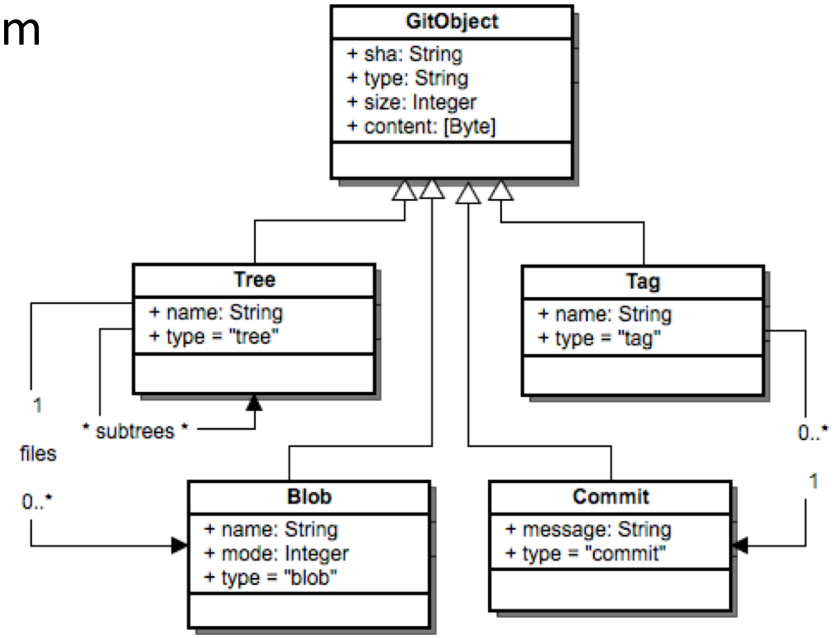


AOSA Example: Git

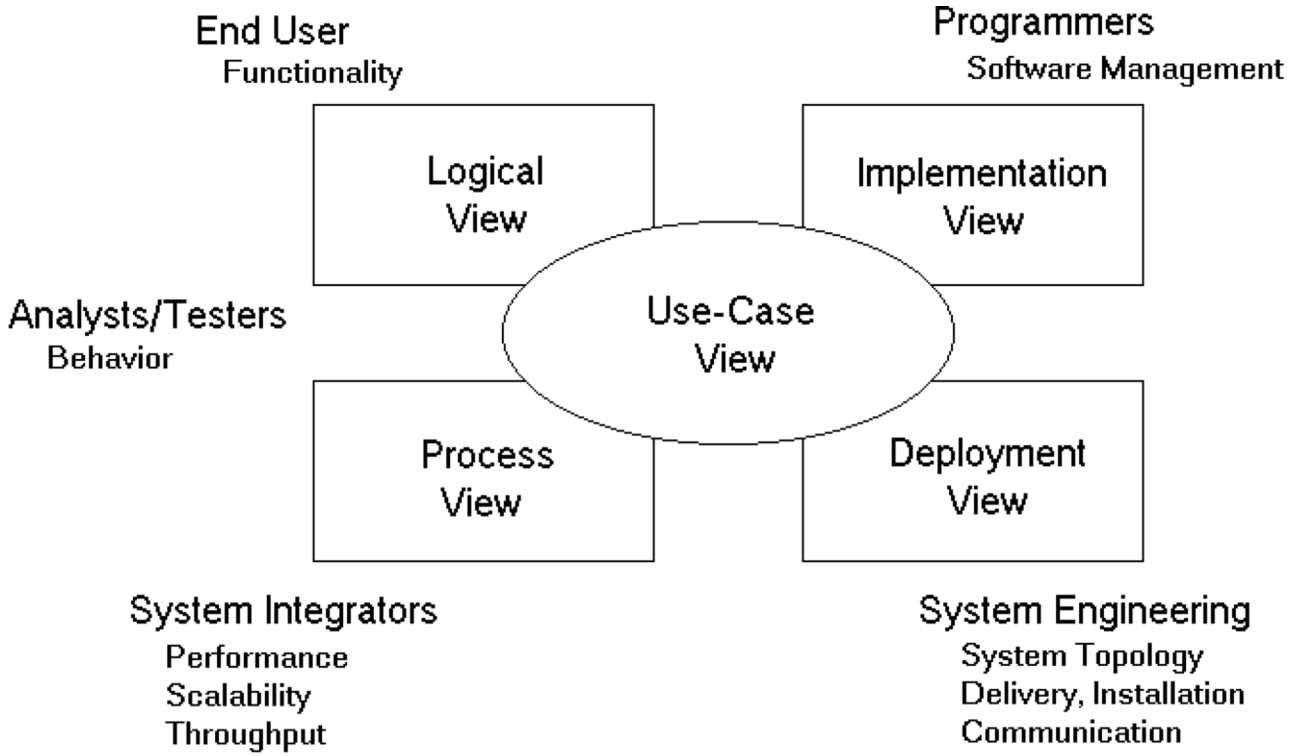


Susan Potter

1. Git in a Nutshell
2. Git's Origin
3. Version Control System Design
4. The Toolkit
5. The Repository
6. The Object Database
7. Storage
8. Merge Histories
9. What's Next?
10. Lessons learned

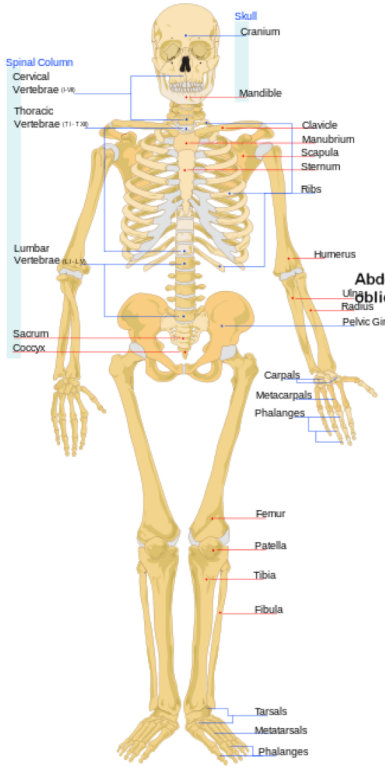


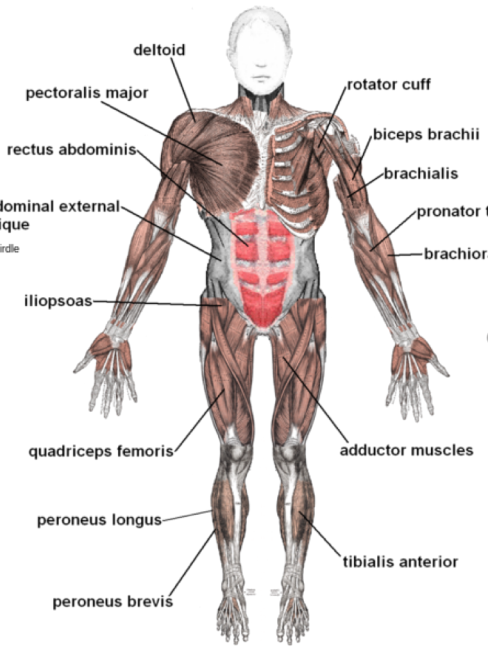
Kruchten's "4+1 Views"

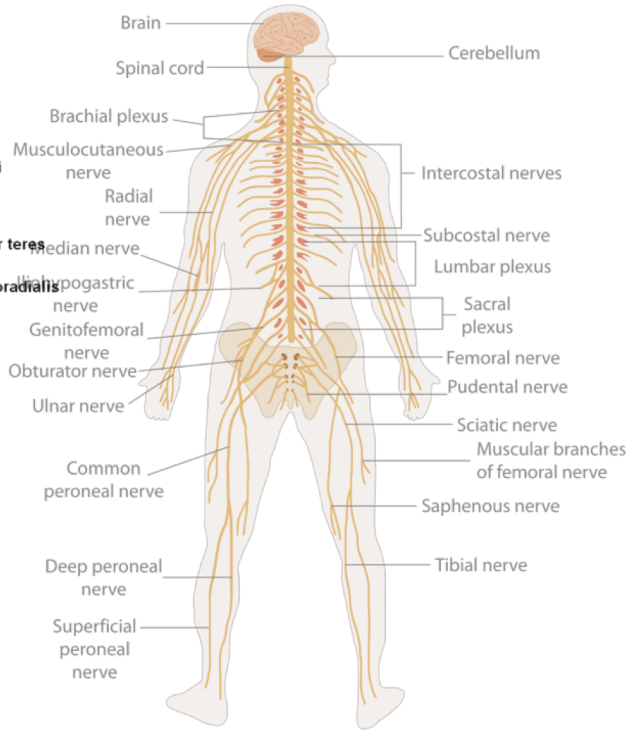


IEEE Software, November 1995

Architectural Views: Bones, Muscles, Nerves





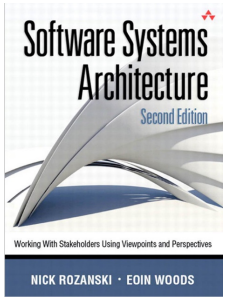


Viewpoints

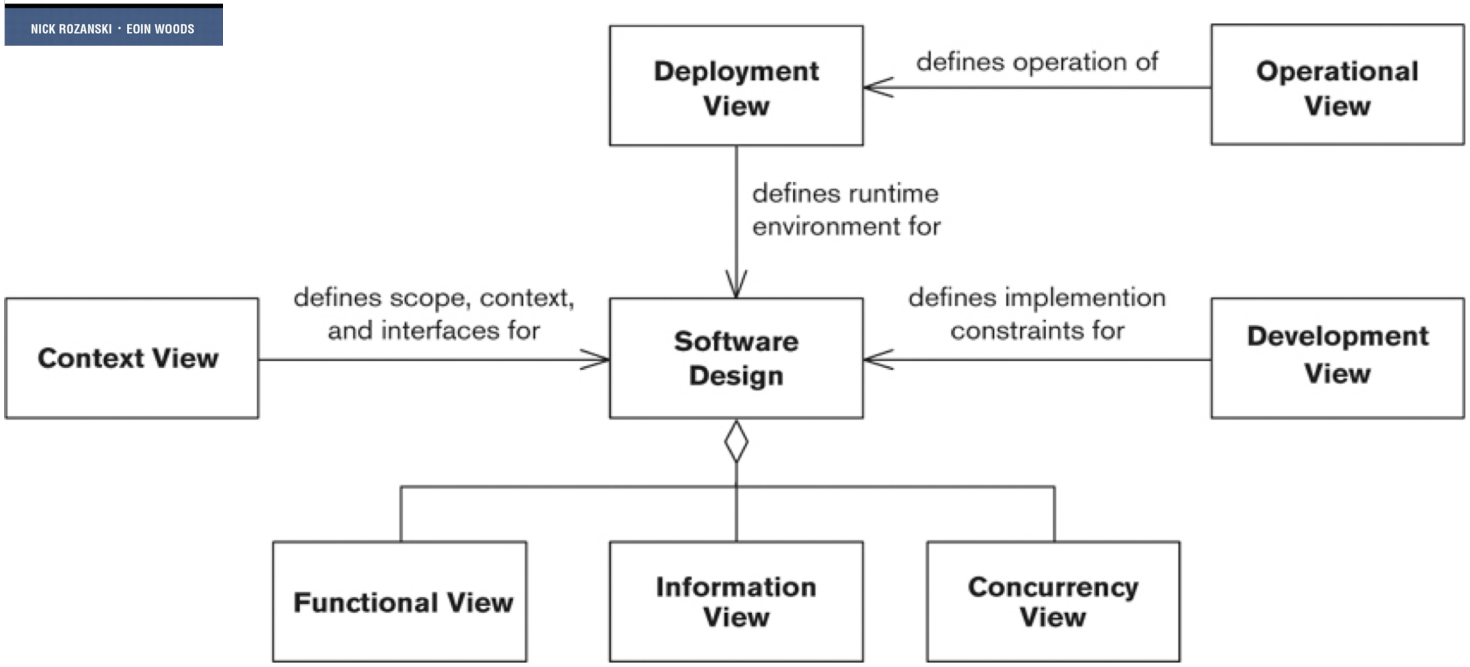

Ch.3

- A collection of patterns, templates, and conventions for constructing one type of view.

- Defines the
  - stakeholders whose concerns are reflected in the viewpoint
  - and the guidelines, principles, and template models for constructing its views.



A Viewpoint Taxonomy



Context View

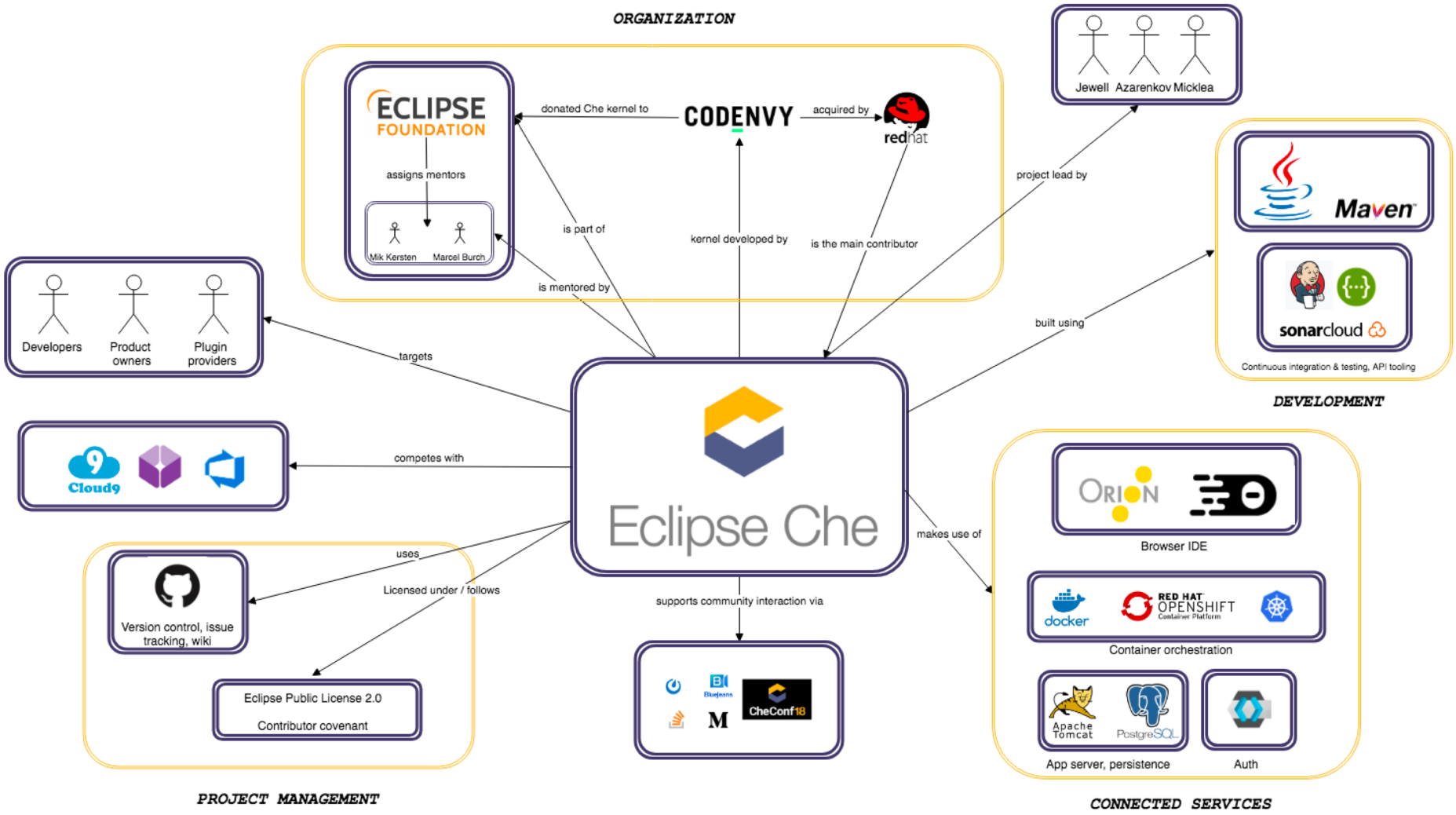

Ch.16

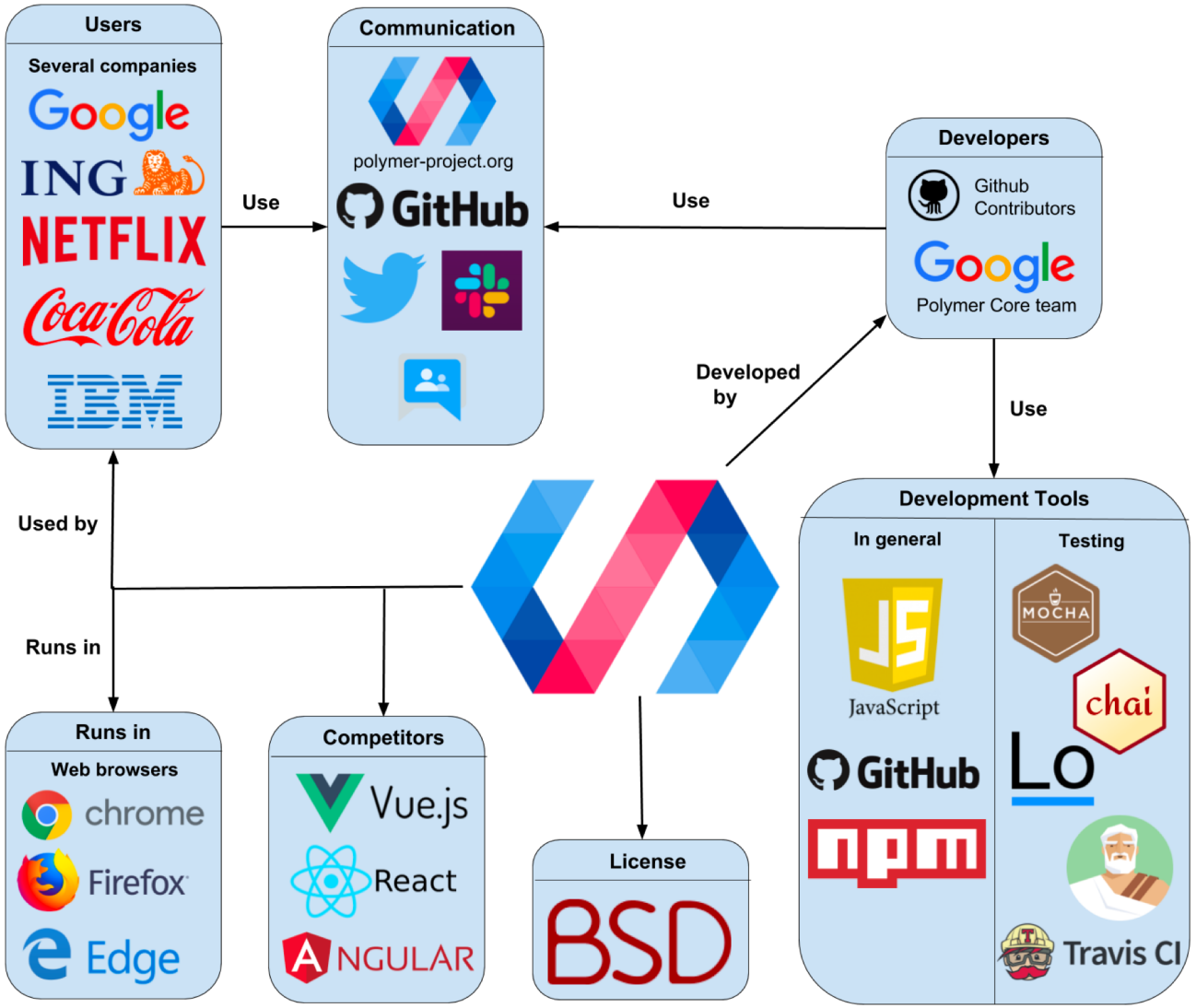
Describes the

relationships, dependencies, and interactions

between the system and its environment

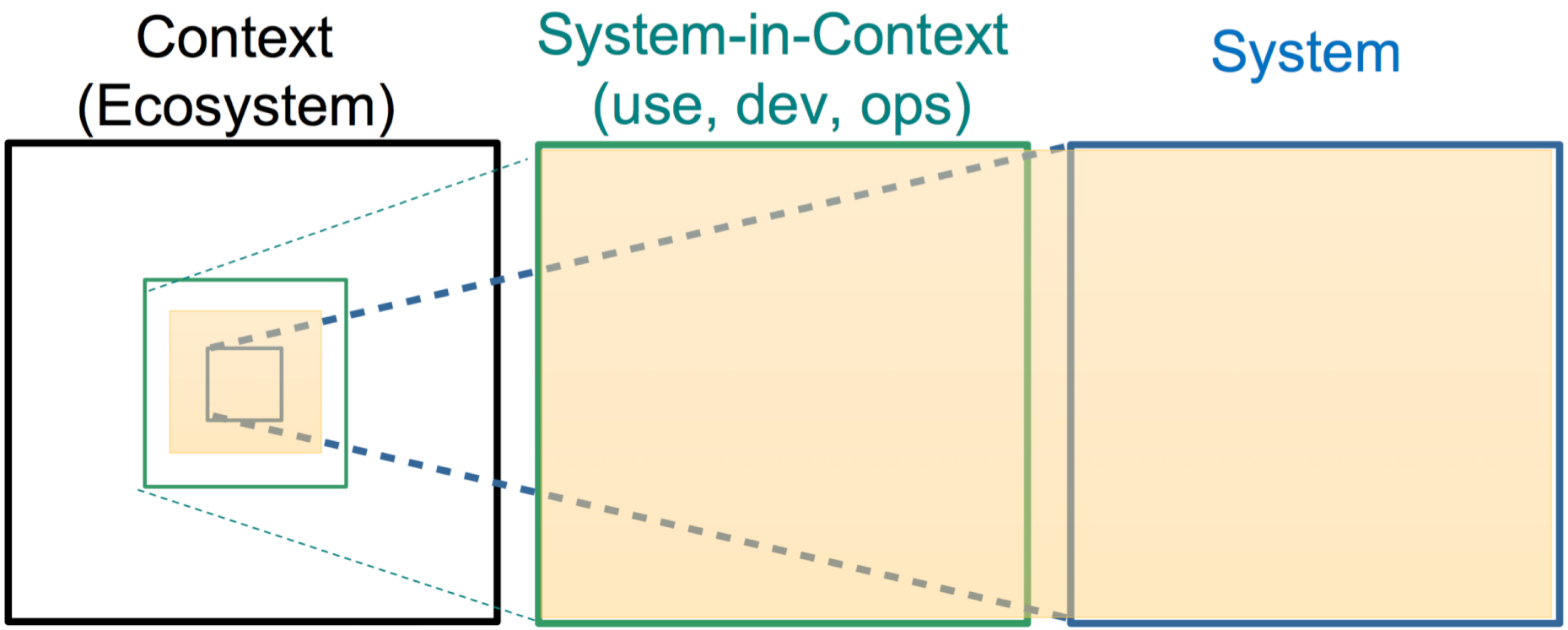
Environment: the people, systems, and external entities with which it interacts


ORGANIZATION
ECLIPSE FOUNDATION
assigns mentors
Mik Kersten
Marcel Burch
donated Che kernel to
CODENVY
acquired by
redhat
is part of
kernel developed by
is the main contributor
is mentored by
project lead by
Jewell Azarenkov Micklea
Developers
Product owners
Plugin providers
targets
built using
Maven
sonarcloud
Continuous integration & testing, API tooling
DEVELOPMENT
Eclipse Che
Cloud9
competes with
uses
Licensed under / follows
Version control, issue tracking, wiki
Eclipse Public License 2.0
Contributor covenant
PROJECT MANAGEMENT
supports community interaction via
CheConf18
makes use of
ORION
Browser IDE
docker
RED HAT OPENSHIFT
Container orchestration
Apache Tomcat
PostgreSQL
App server, persistence
Auth
CONNECTED SERVICES


Users
Several companies
Google
ING
NETFLIX
Coca-Cola
IBM
Use
Communication
polymer-project.org
GitHub
Use
Developers
Github Contributors
Google
Polymer Core team
Developed by
Use
Used by
Runs in
Development Tools
In general
Testing
JavaScript
GitHub
npm
MOCHA
chai
Lo
Travis CI
Runs in
Web browsers
chrome
Firefox
Edge
Competitors
Vue.js
React
ANGULAR
License
BSD

"Always design a thing by considering it in its next larger context"

—Eliel Saarinen



Strategy
ecosystem interventions

"Requirements"
design of system capabilities

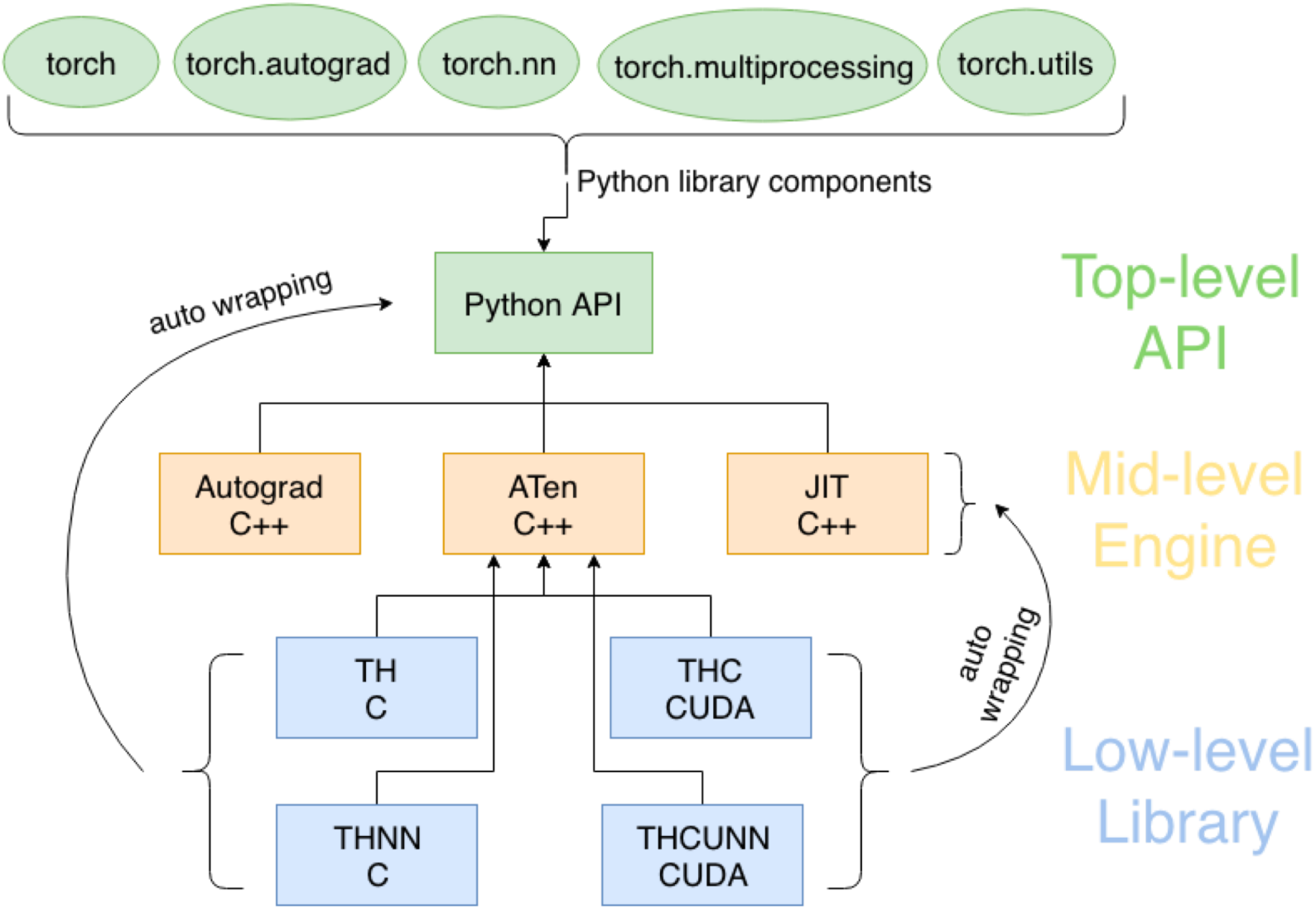
Architecture
structure and mechanisms

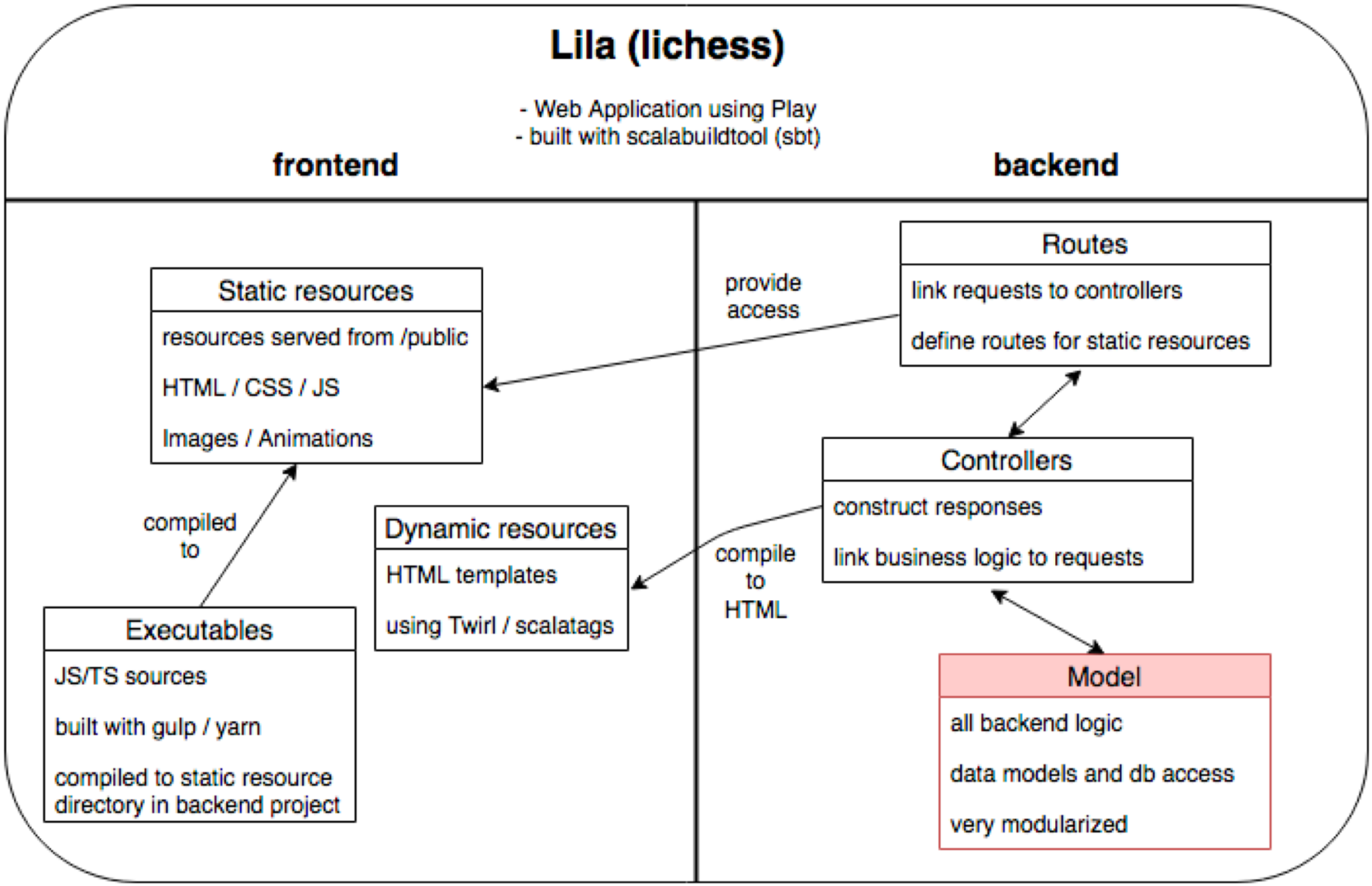
Development View

Ch.20

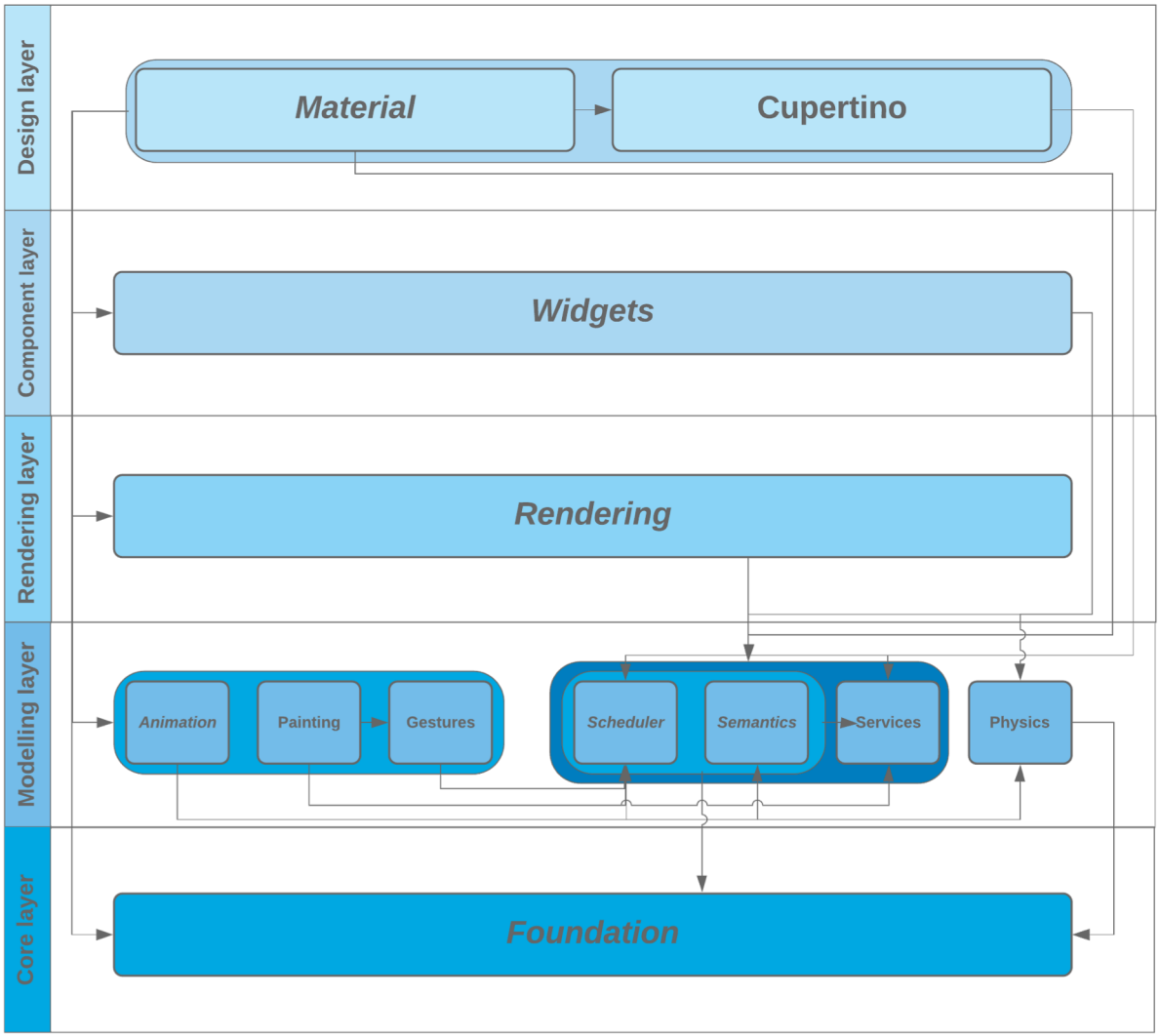
Describes the architecture that supports the software development process.

Communicates the aspects of the architecture of interest to stakeholders involved in building, testing, maintaining, and enhancing the system.


torch
torch.autograd
torch.nn
torch.multiprocessing
torch.utils
Python library components
auto wrapping
Python API
Autograd
C++
ATen
C++
JIT
C++
TH
C
THC
CUDA
THNN
C
THCUNN
CUDA
auto
wrapping
Top-level
API
Mid-level
Engine
Low-level
Library


Lila (lichess)
- Web Application using Play
- built with scalabuildtool (sbt)
frontend
backend
Static resources
resources served from /public
HTML / CSS / JS
Images / Animations
provide access
Routes
link requests to controllers
define routes for static resources
compiled to
Dynamic resources
HTML templates
using Twirl / scalatags
compile to HTML
Controllers
construct responses
link business logic to requests
Executables
JS/TS sources
built with gulp / yarn
compiled to static resource directory in backend project
Model
all backend logic
data models and db access
very modularized

Flutter


Design layer
Material
Cupertino
Component layer
Widgets
Rendering layer
Rendering
Modelling layer
Animation
Painting
Gestures
Scheduler
Semantics
Services
Physics
Core layer
Foundation

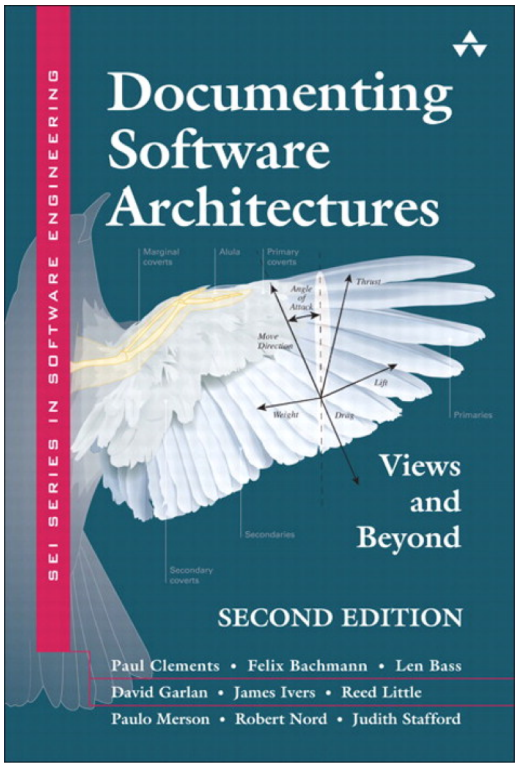
Alternative Catalogs

"View types":

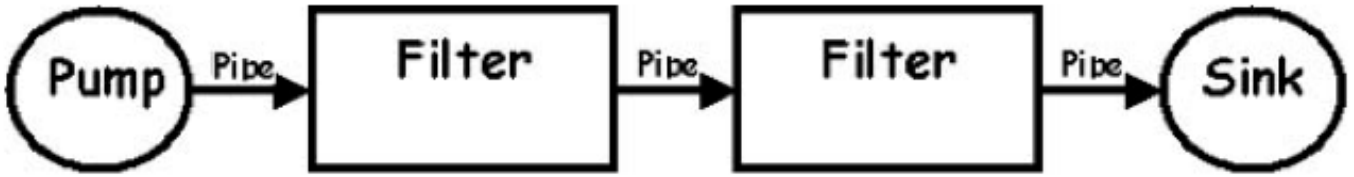
- Module
- Component & Connector
- Allocation

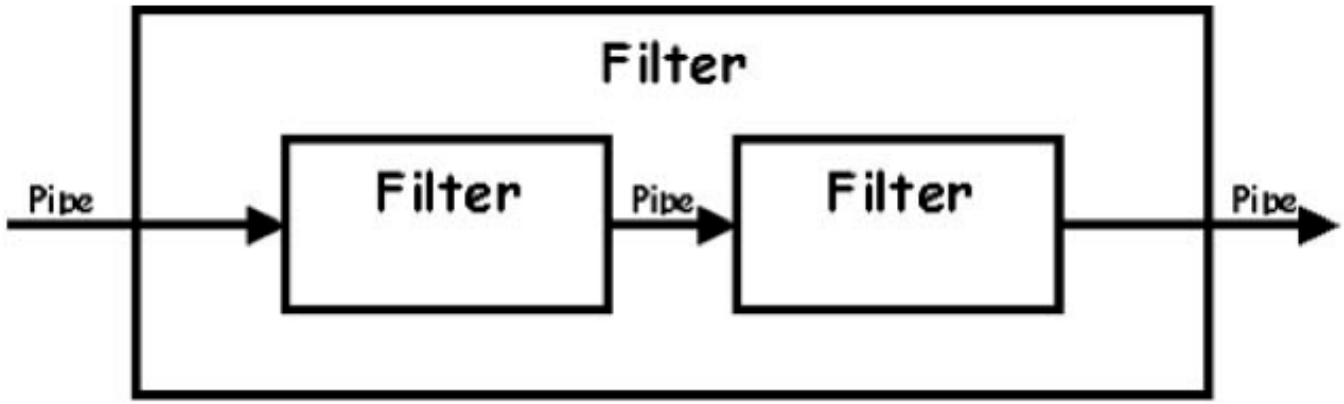
Component & connectors:

- Pipe and filter, shared data, publish subscribe, client-server, p2p, …

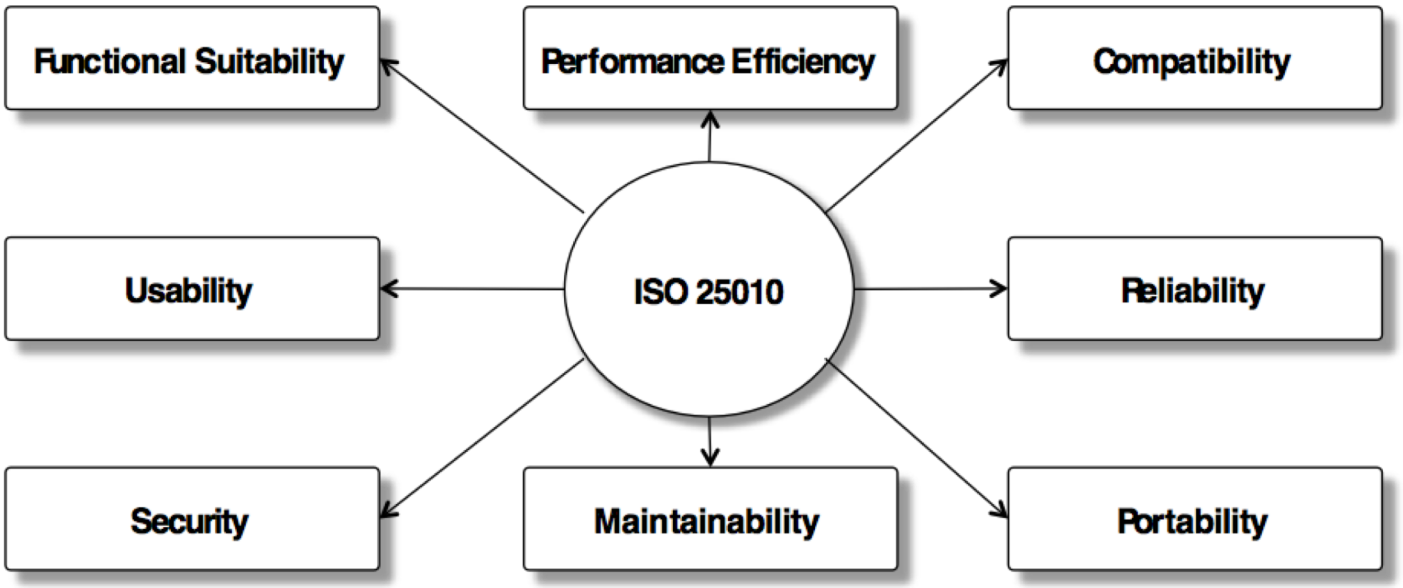


Example: Pipes & Filter

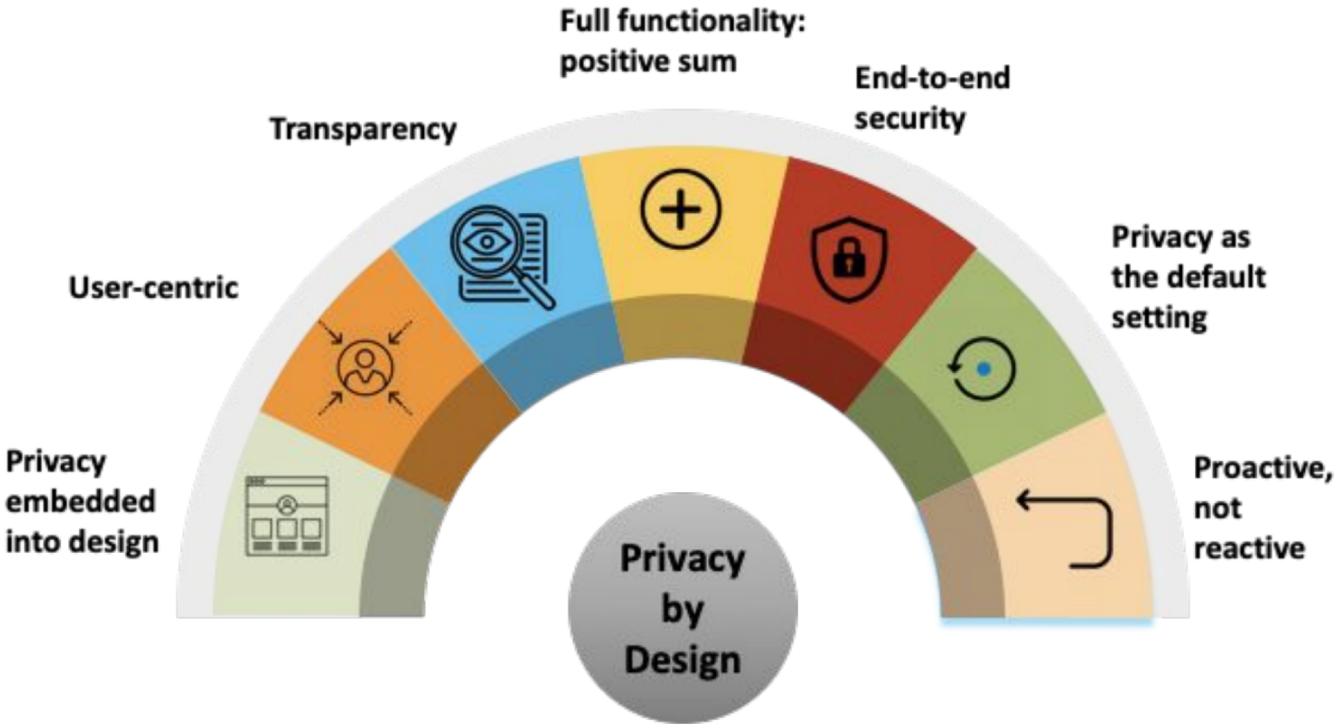


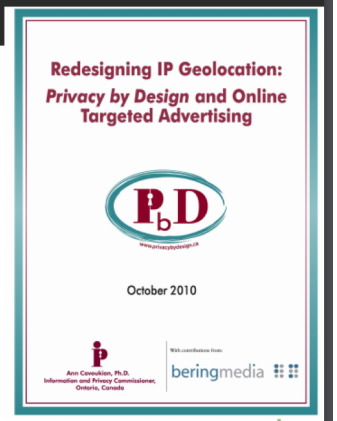


ISO Software Quality Characteristics



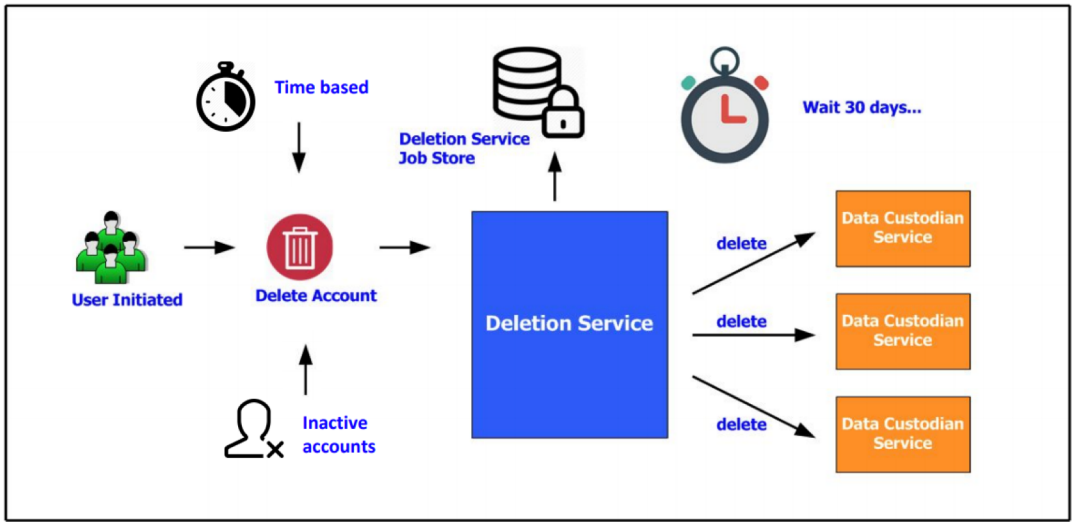
Privacy by Design (Since 1995)





Uber's Approach to Data Deletion

- Support scale of data, data stores, and microservices
- Privacy Impact Assessment and Technical Privacy Reviews
- Vetting process combines legal and technical privacy
- Automate onboarding process for new services



Realizing Quality Attributes

- An architecture must realize the required quality attributes
- Models required that permit reasoning over quality attributes
- Architectural decisions may have to make tradeoff between conflicting quality attributes

Architectural *Perspectives*

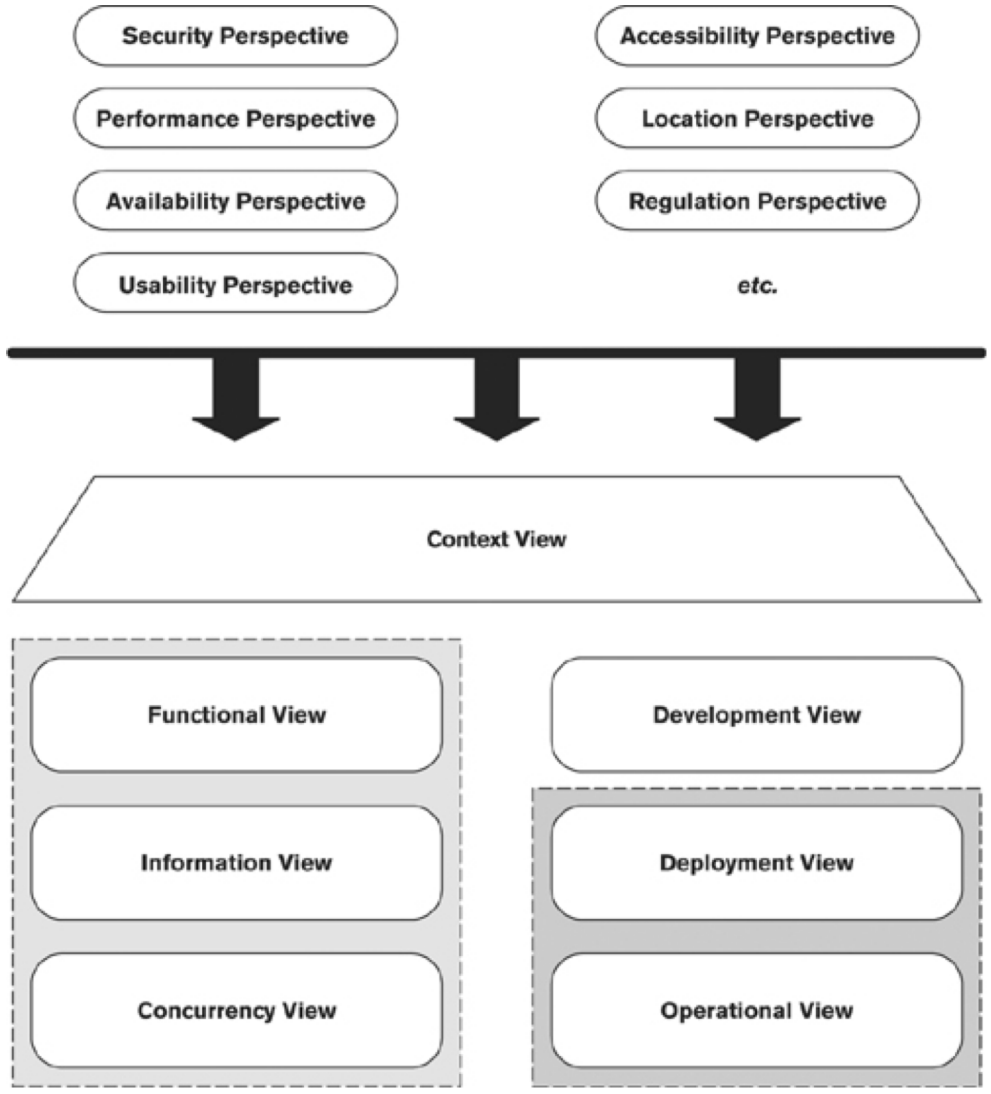

Ch.4

An architectural perspective is a

collection of architectural activities, tactics, and guidelines

that are used to ensure that a system exhibits a particular set of related quality properties

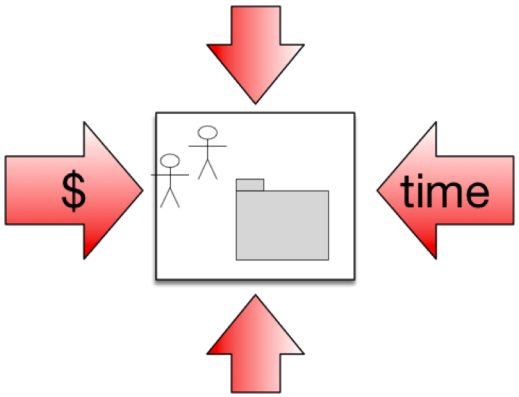
that require consideration across a number of the system's architectural views.


Security Perspective
Accessibility Perspective
Performance Perspective
Location Perspective
Availability Perspective
Regulation Perspective
Usability Perspective
etc.
Context View
Functional View
Development View
Information View
Deployment View
Concurrency View
Operational View

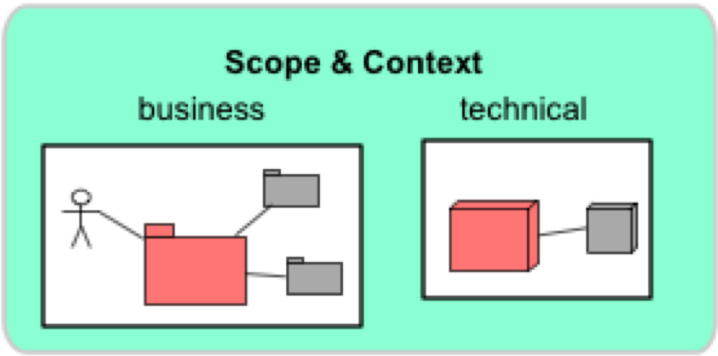
The arc42.org Template for Architecture Communication and Documentation

1. Introduction and Goals

2. Constraints

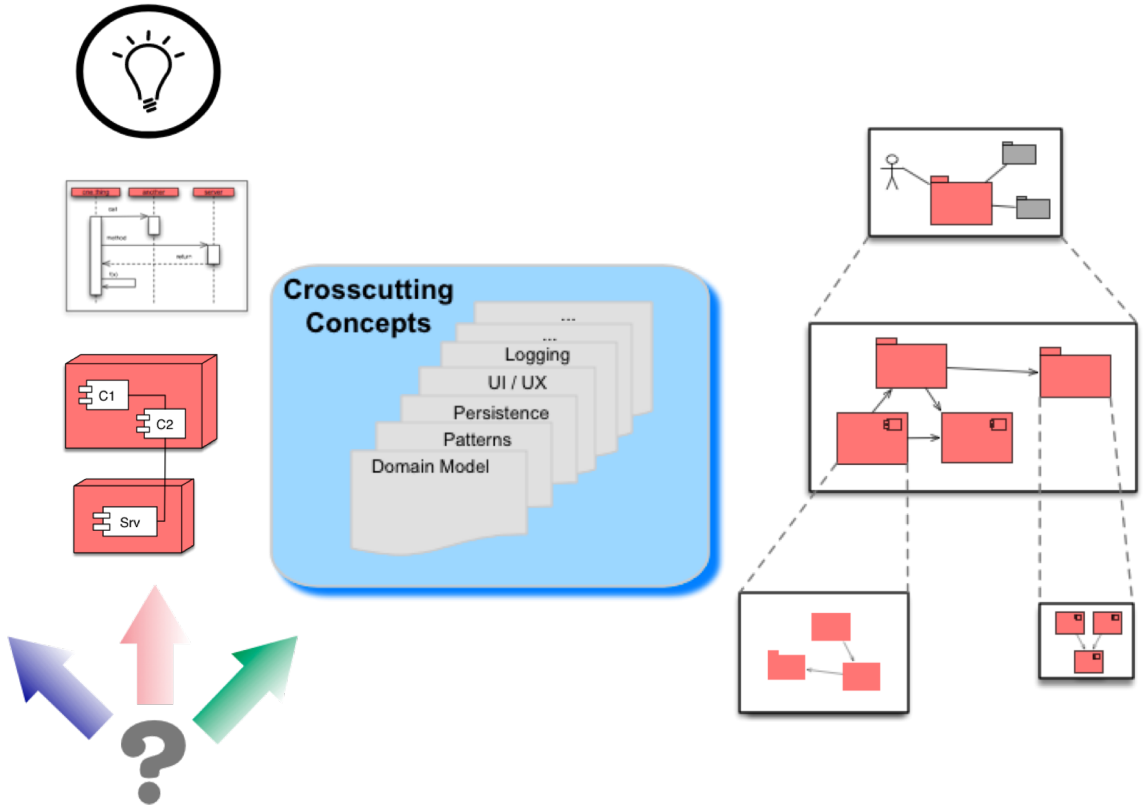


3. Context and Scope



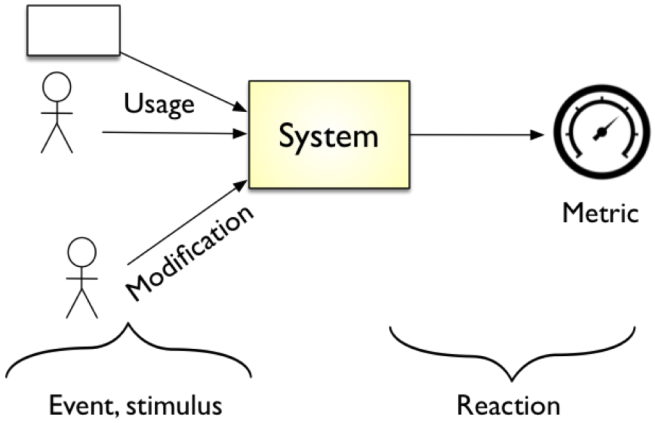
The arc42.org Template for Architecture Communication and Documentation

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

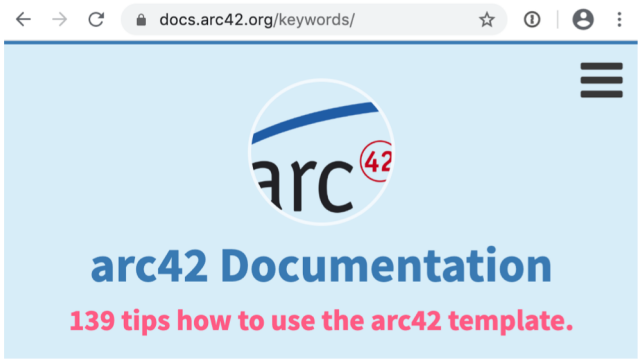


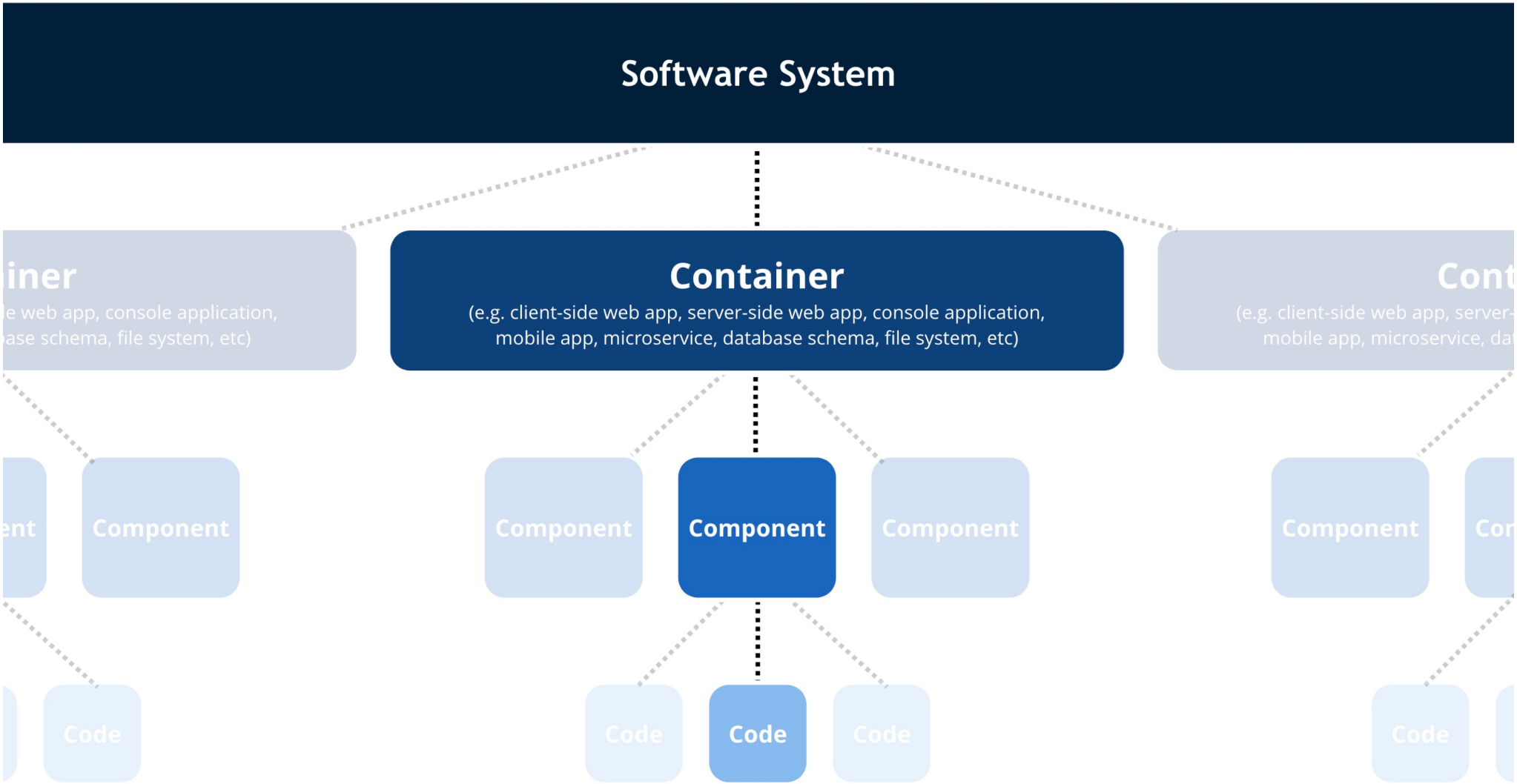
The arc42.org Template for Architecture Communication and Documentation

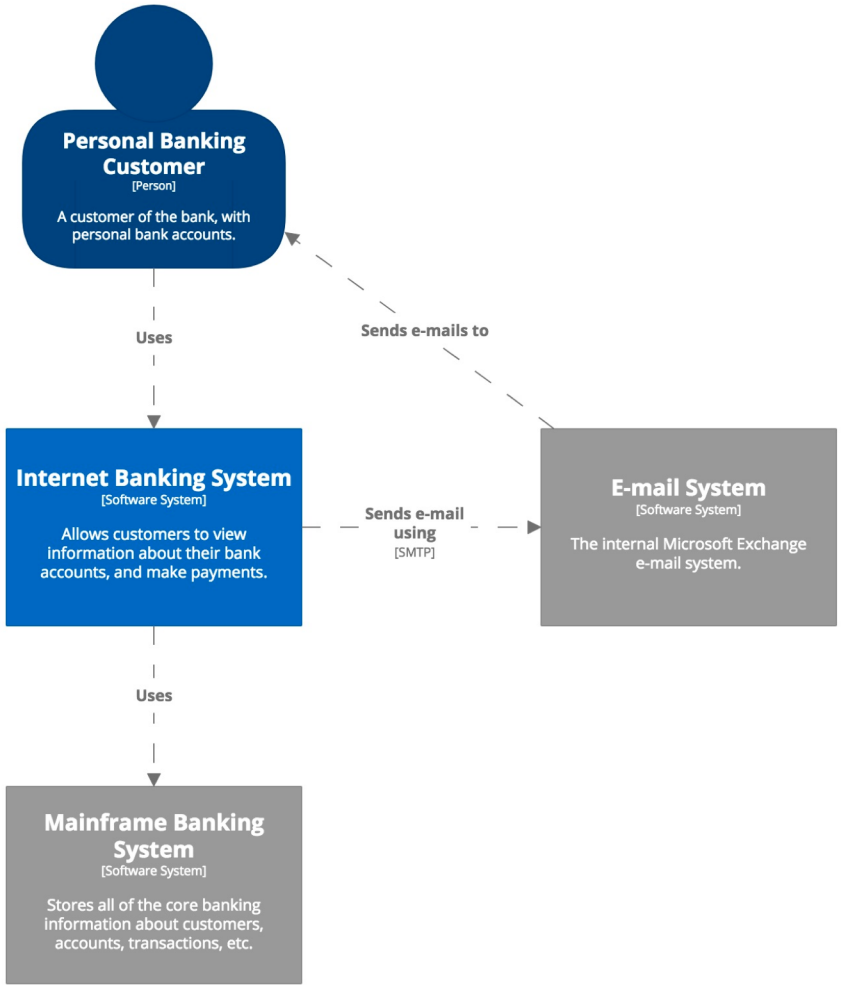
10. Quality Requirements



11. Risks and Technical Debt

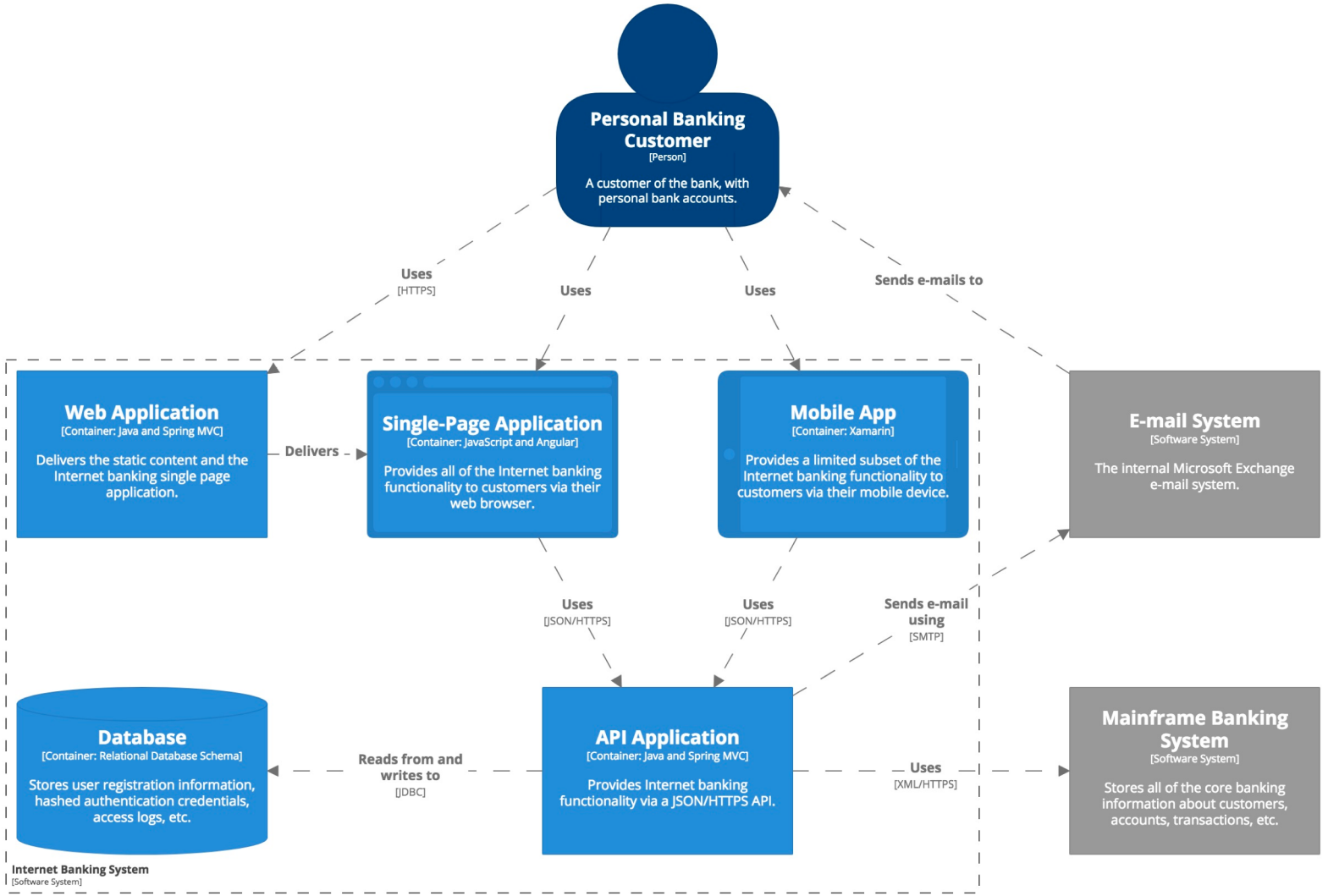



Software System
Container
(e.g. client-side web app, server-side web app, console application, mobile app, microservice, database schema, file system, etc)
Component
Component
Component
Component
Code
Code
Code
Code



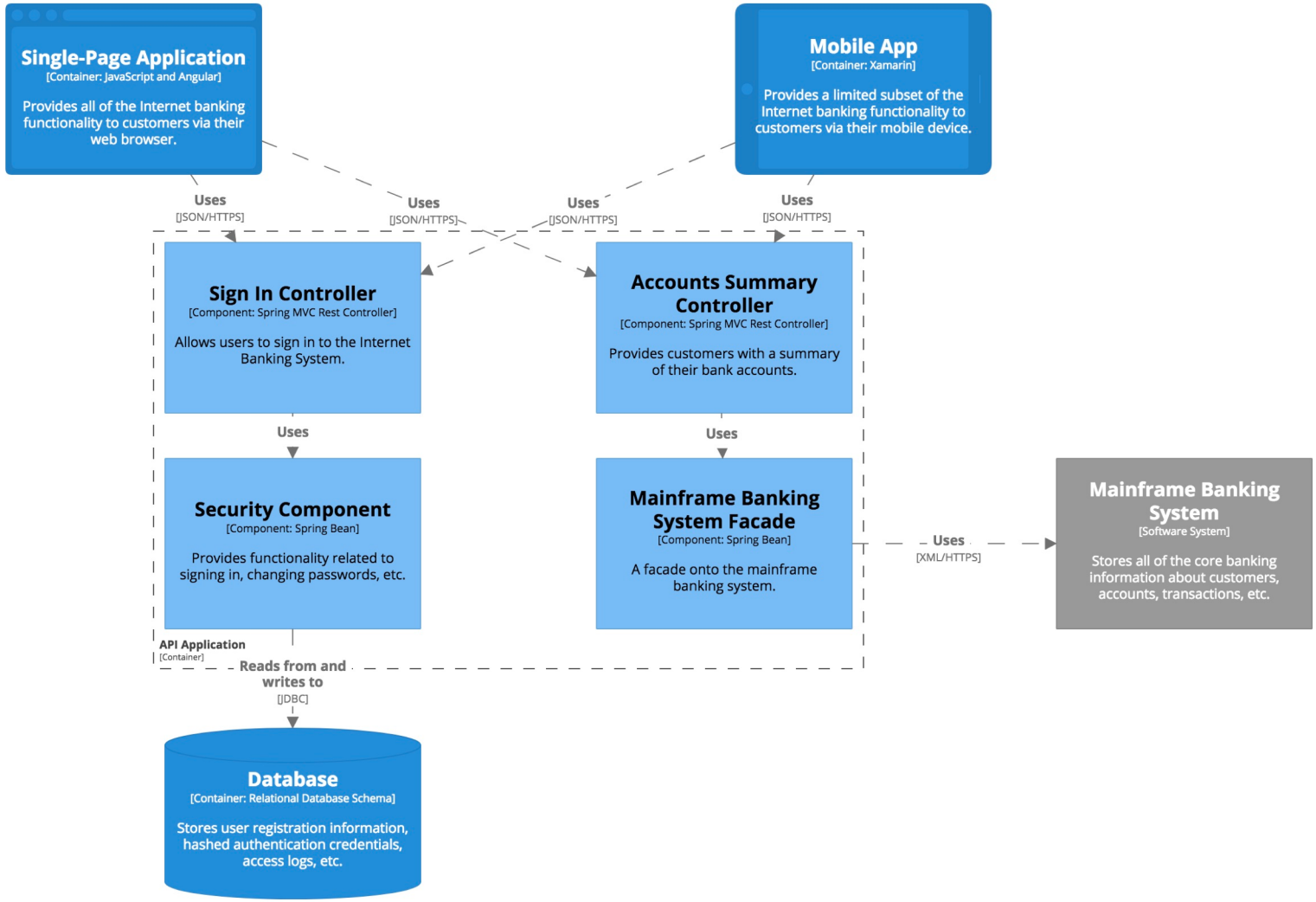
System Context diagram for Internet Banking System

The system context diagram for the Internet Banking System.
Last modified: Wednesday 02 May 2018 13:46 UTC



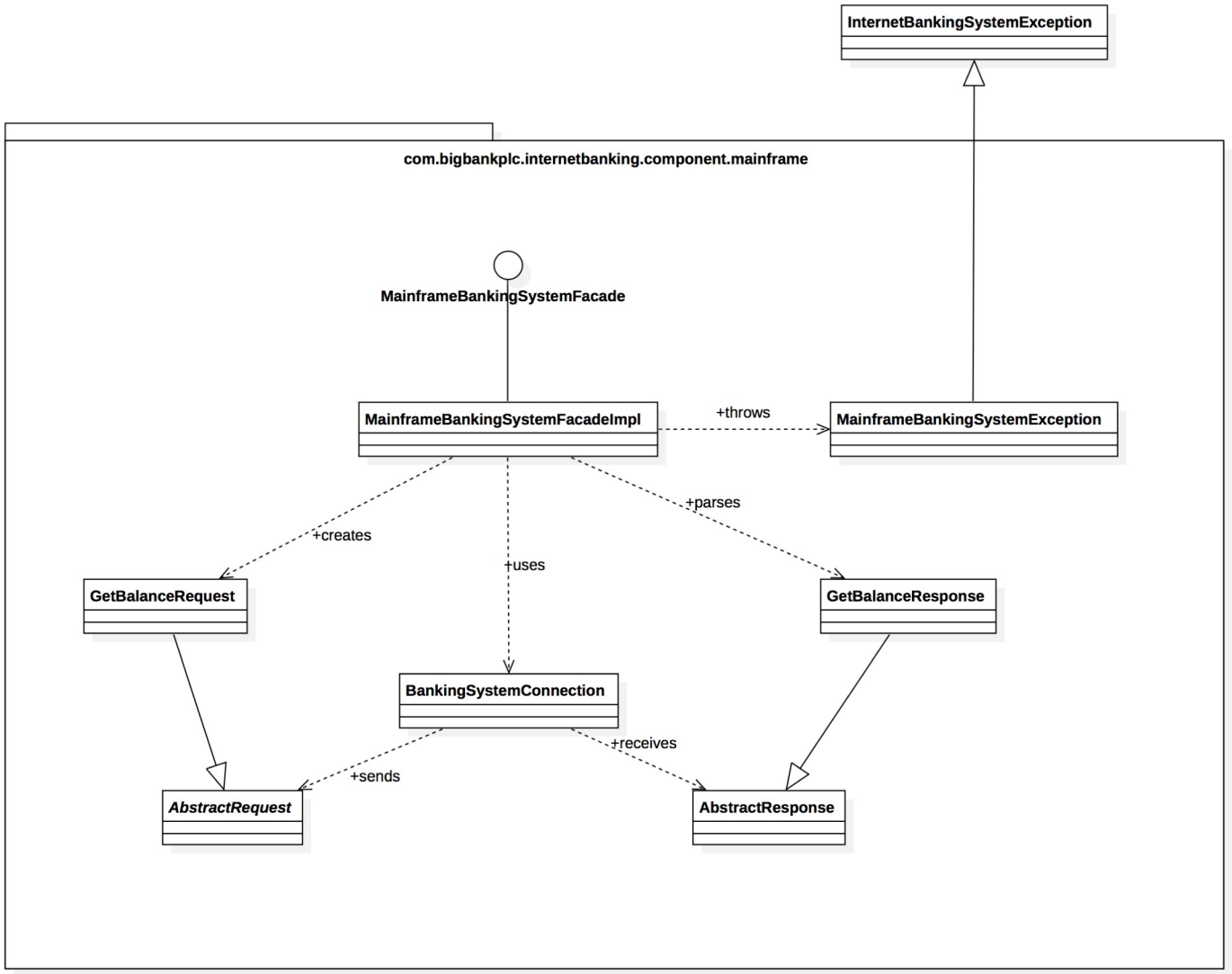
Container diagram for Internet Banking System

The container diagram for the Internet Banking System.
Last modified: Wednesday 02 May 2018 13:46 UTC



Component diagram for Internet Banking System - API Application

The component diagram for the API Application.
Last modified: Wednesday 02 May 2018 13:46 UTC


InternetBankingSystemException
com.bigbankplc.internetbanking.component.mainframe
MainframeBankingSystemFacade
MainframeBankingSystemFacadeImpl
+throws
MainframeBankingSystemException
+creates
+uses
+parses
GetBalanceRequest
GetBalanceResponse
BankingSystemConnection
+receives
+sends
AbstractRequest
AbstractResponse

Why doesn't the C4 model cover business processes, workflows, state machines, domain models, data models, etc?

The focus of the C4 model is the static structures that make up a software system, at different levels of abstraction. If you need to describe other aspects, feel free to supplement the C4 diagrams with UML diagrams, BPML diagrams, ArchiMate diagrams, entity relationship diagrams, etc.

The C4 model vs UML, ArchiMate and SysML?

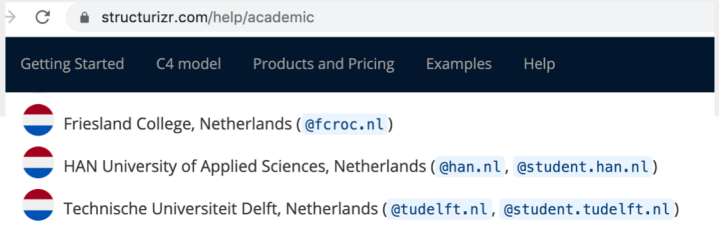
Although existing notations such as UML, ArchiMate and SysML already exist, many software development teams don't seem to use them. Often this is because teams don't know these notations well enough, perceive them to be too complicated, think they are not compatible with agile approaches or don't have the required tooling.

If you are already successfully using one of these notations to communicate software architecture and it's working, stick with it. If not, try the C4 model. And don't be afraid to supplement the C4 diagrams with UML state diagrams, timing diagrams, etc if you need to.

Can we combine C4 and arc42?

Yes, many teams do, and the C4 model is compatible with the arc42 documentation template as follows.

- Context and Scope => System Context diagram
- Building Block View (level 1) => Container diagram
- Building Block View (level 2) => Component diagram
- Building Block View (level 3) => Class diagram



What Is Technical Debt?

- Ward Cunningham:
  - “I coined the debt metaphor to explain the refactoring that we were doing.”

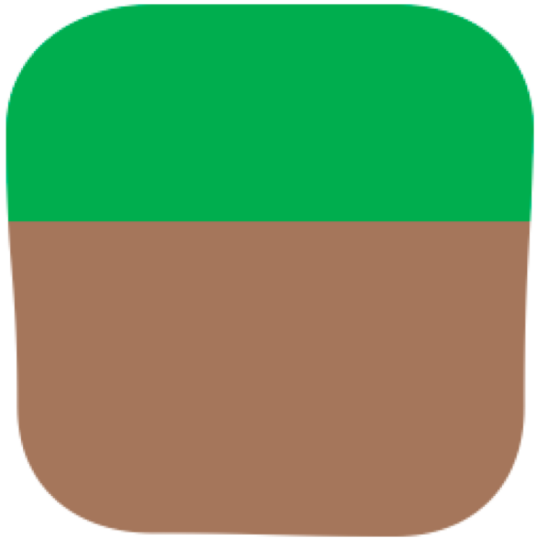
http://c2.com/cgi/wiki?WardExplainsDebtMetaphor

- Michael Feathers:
  - “The refactoring effort needed to add a feature non invasively”

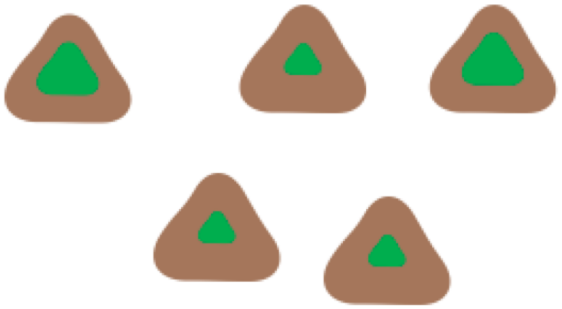
https://www.youtube.com/watch?v=7hL6g1aTGvo

*Any software system has a certain amount of **essential** complexity required to do its job...*

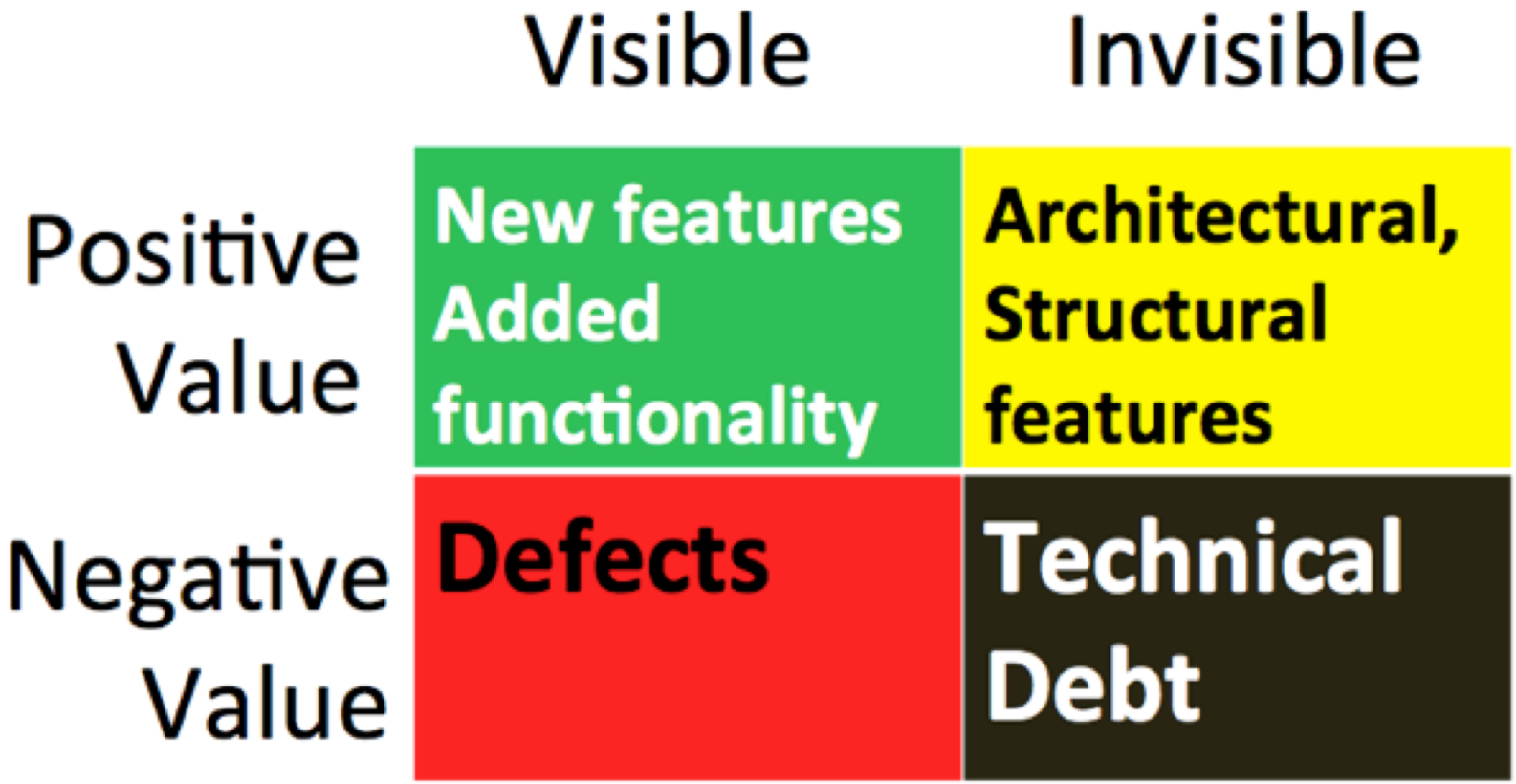
*... but most systems contain **cruft** that makes it harder to understand.*



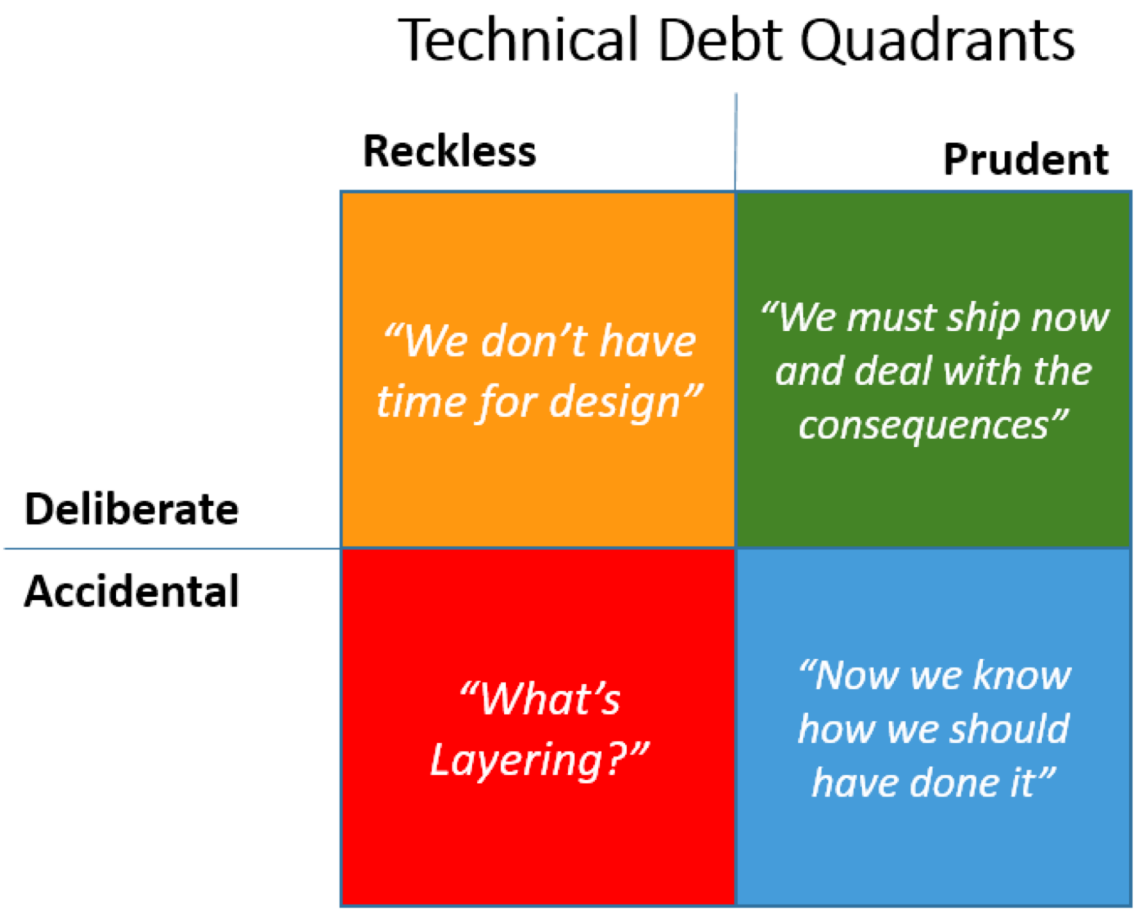
*Cruft causes changes to take **more effort***



The technical debt metaphor treats the cruft as a debt, whose interest payments are the extra effort these changes require.



Kruchten, 2013:
The (missing) value of software architecture


Technical Debt Quadrants
Reckless
Prudent
Deliberate
Accidental
"We don't have time for design"
"We must ship now and deal with the consequences"
"What's Layering?"
"Now we know how we should have done it"

Learning how to do it

I am in favor of writing code to reflect
your current understanding of a problem
even if that understanding is partial

http://c2.com/cgi/wiki?WardExplainsDebtMetaphor

Assessing Technical Debt?

- https://www.sonarqube.org/
- https://www.jarchitect.com/Metrics
- https://github.com/tsantalis/JDeodorant

Beware: Debt is Relative

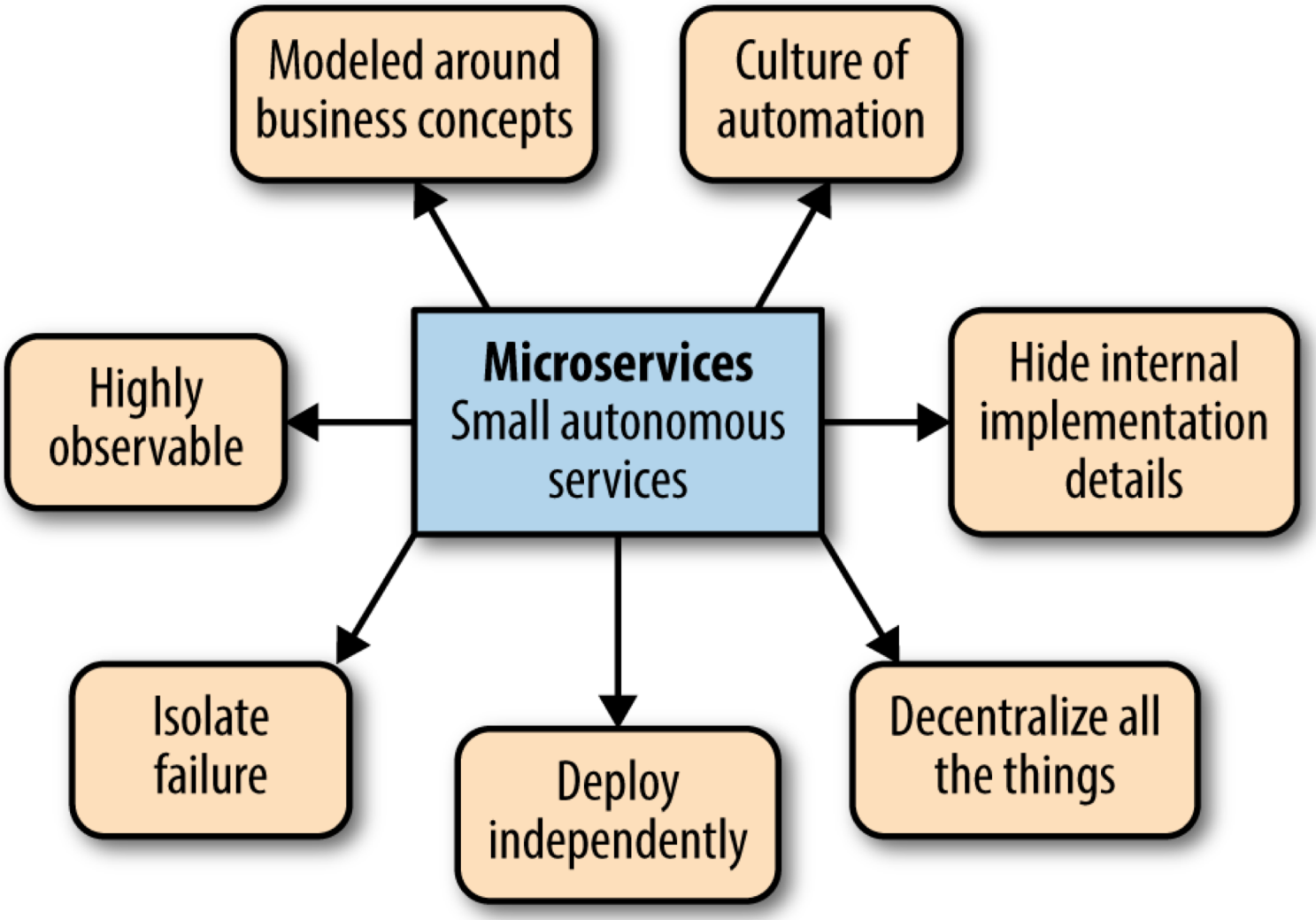
- *The refactoring effort needed to add a feature (resolve an issue) non invasively*
  - Debt depends on features and issues to solve

- Systems are used and society progresses
  - New libraries and versions come available
  - Actual usage affects our understanding of what matters

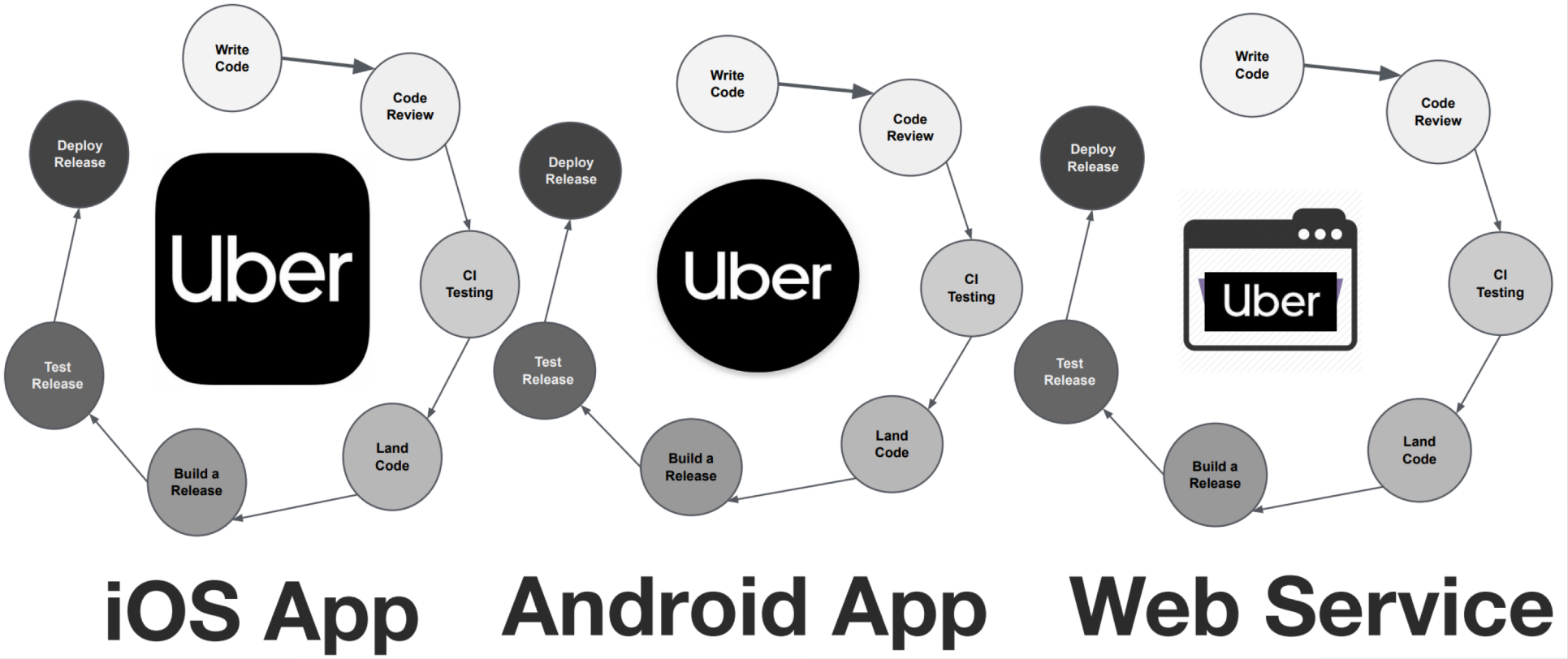
- Debt quantifications / visualizations are only useful when they lead to *action.* Avoid ranting; propose rational action instead.

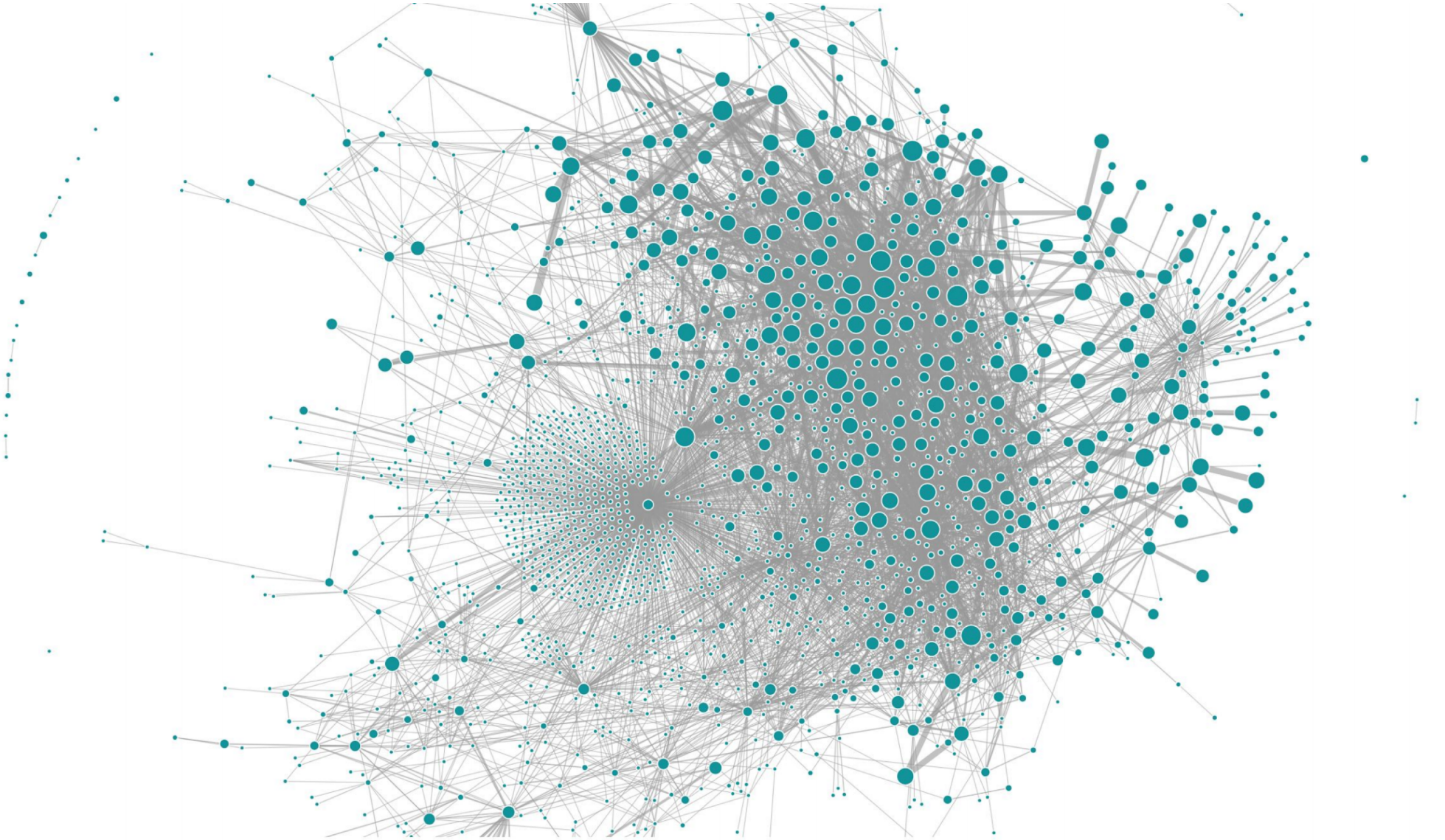
Microservices

- Small, autonomous services that work together
- Single Responsibility Principle:
  - Gather together those things that change for the same reason
- Strong cohesion within the service
- Loose coupling among services


Modeled around business concepts
Culture of automation
Highly observable
Microservices
Small autonomous services
Hide internal implementation details
Isolate failure
Deploy independently
Decentralize all the things

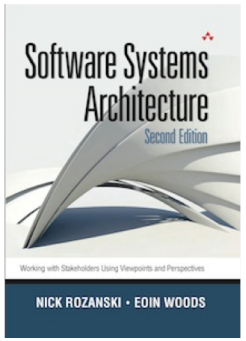
Microservices





The Role of the Architect

The architect is responsible for designing, documenting, and leading the construction of a system that meets the needs of all its stakeholders.

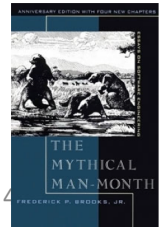


Fred Brooks: *Conceptual Integrity*

The quality of a system where all the concepts and their relationships with each other are applied in a consistent way throughout the system.

Conceptual Integrity is the most important consideration in system design.

It is better to have […] <u>one set of design ideas</u>,
than [...] many good but independent and uncoordinated ideas.



How many Architects?

Conceptual integrity in turn dictates
that the design must proceed from one mind, or from a very small number
of agreeing resonant minds

The Evolutionary Architect

"architects need to shift their thinking away from creating the perfect end product,

and instead focus on helping create a framework in which the right systems can emerge, and continue to grow as we learn more."

Software Architect = Town Planner

- Attempt to optimize the layout of a city
  - to best suit the needs of the citizens today,
  - taking into account future use

- Cannot foresee everything that will happen.
  - Don’t plan for any eventuality,
  - Plan to allow for change

The Coding Architect?

- Architects must ensure that systems are ‘habitable’ for developers too.

- Architects must spend time with the team
- Architects must spend time coding
  - Pair with a developer
  - Beyond code review
- This should be a routine activity