Date	Start	End	Activity	Teacher	Topic
Wed Feb 10	13:45	15:30	Lecture 1	Arie van Deursen	Introduction and Course Structure (slides)
Fri Feb 12	08:45	10:30	Lecture 2	Arie van Deursen	Envisioning the System (slides)
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	Bert Wolters (Adyen)	Architecting for Scalability
Wed Mar 10	13:45	15:30	Lecture 9	Steffan Norberhuis	Architecting for Operations
Fri Mar 12	08:45	10:30	Lecture 10	Xavier Devroey	Architecting for Variability
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

Labwork Q&A (1)

- It is OK to use collaborative editors like overleaf / Google docs
  - Push markdown often and early
  - Use journal to explain who did what

- Being a “guest” in mattermost channels of other teams?
  - Make yourself known and explain why you are present
  - If you wish to learn from other team, ask, and explain what you learned
  - Helping is great (but help should be appreciated)
  - As team, it is ok to ask @all in your channel about their intended role

Labwork Q&A (2)

- Main branch is called `main`, not `master`.
    - You can work on branches and push them
    - Choose branch names that are local to your team (prefix with system, e.g.)
    - You can merge into `main`, via a merge request

Learning from the Architects: Contributing to Open Source

Arie van Deursen

Delft University of Technology

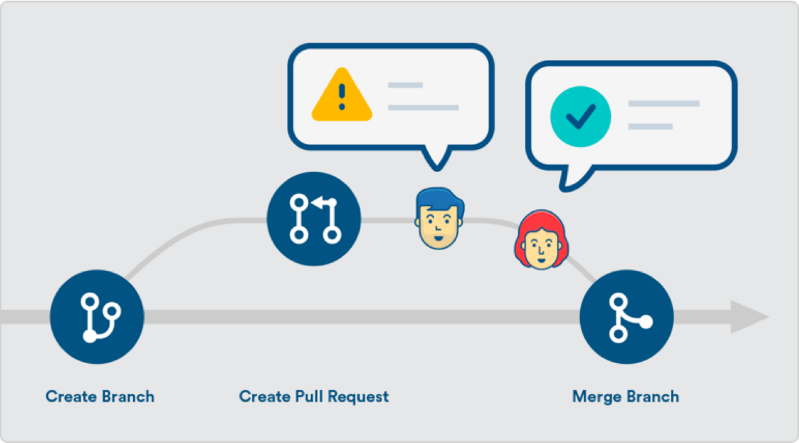
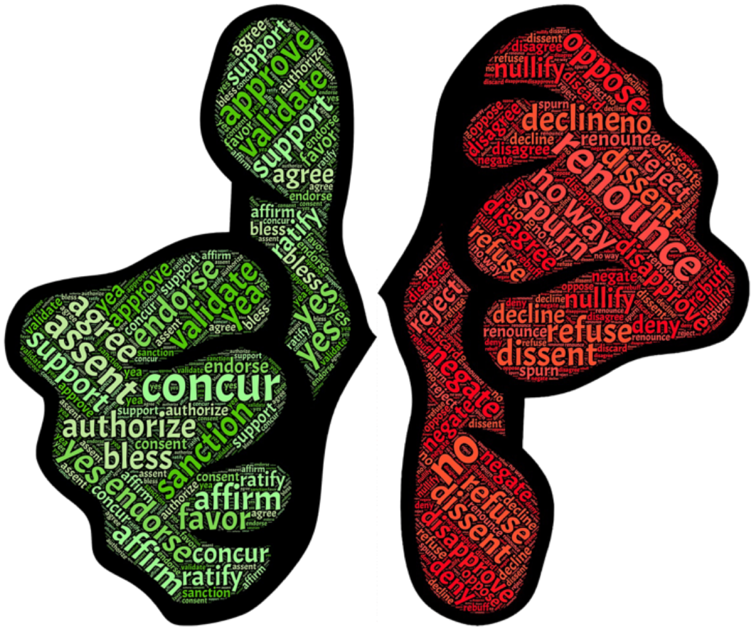


Image credit: Atlassian

The Open Source Architect

- Overall technical decision maker
- Keeper of the vision in times of change:
  - What comes in, what goes out
- Design integrity
  - Design principles guiding changes to code
  - Quality trade-offs
  - Evolution of underlying principles
- Quality assurance: guidelines + control
- Stakeholder management:
  - Listen to the community, prioritize



Learning from Contributing

- Create a meaningful contribution, and request it to be merged (“pulled”)
- Use this to try to understand the full decision making process

- Feel the “hands of the architects”:
  - Trade-offs, prioritization, coding practices, quality control, culture, interaction

- Receive feedback on your own code and way of working
  - Explicit (in comments) or implicit (just a merge / reject)

The Many Shapes of Open Source Contributions

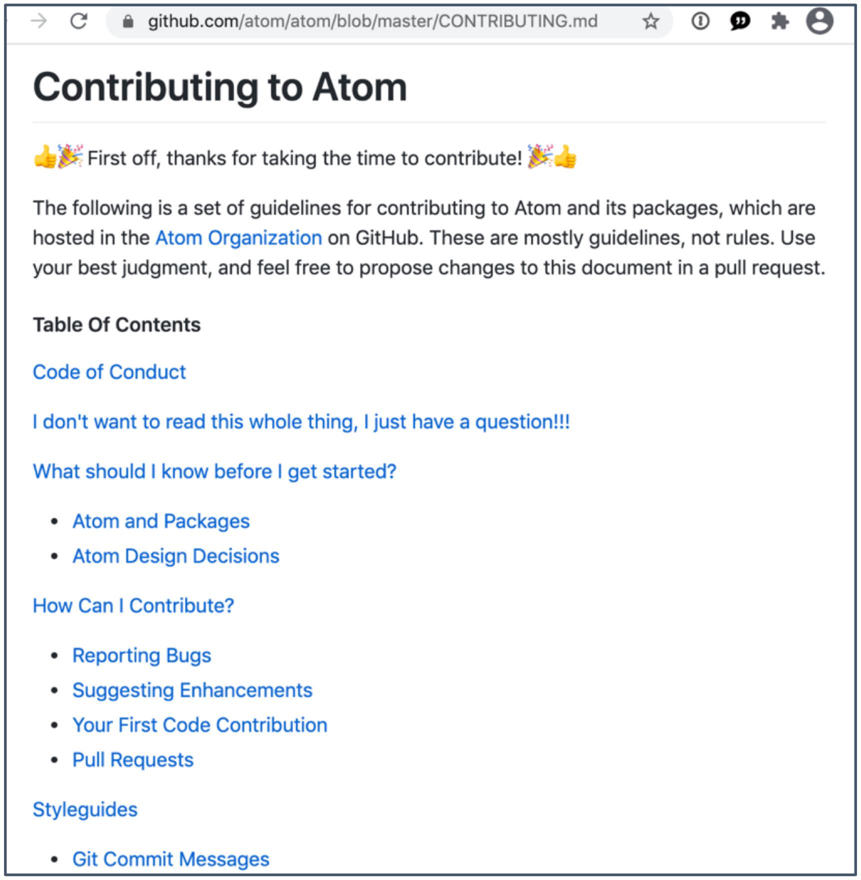
- Documentation
- Internationalization
- Report an issue
- Add some tests (e.g. reproducing a bug)
- Fix a reported bug (with test case)
- Add requested feature (with test case)
- Propose feature (in issue) and build it
- Remove unused or redundant code
- ...

START SIMPLE!

The more interaction with other developers are needed, the more you'll learn about the architecture, and how it guides the decision making process

Getting it Accepted

- Study CONTRIBUTING.md
- Study earlier accepted / rejected pull requests
- Start with simple / starter issues
- Keep it small and simple
- Be clear, concise, and polite
- Know your tools (git, build, ...)



Contributing to Atom

👍🎉 First off, thanks for taking the time to contribute! 🎉👍

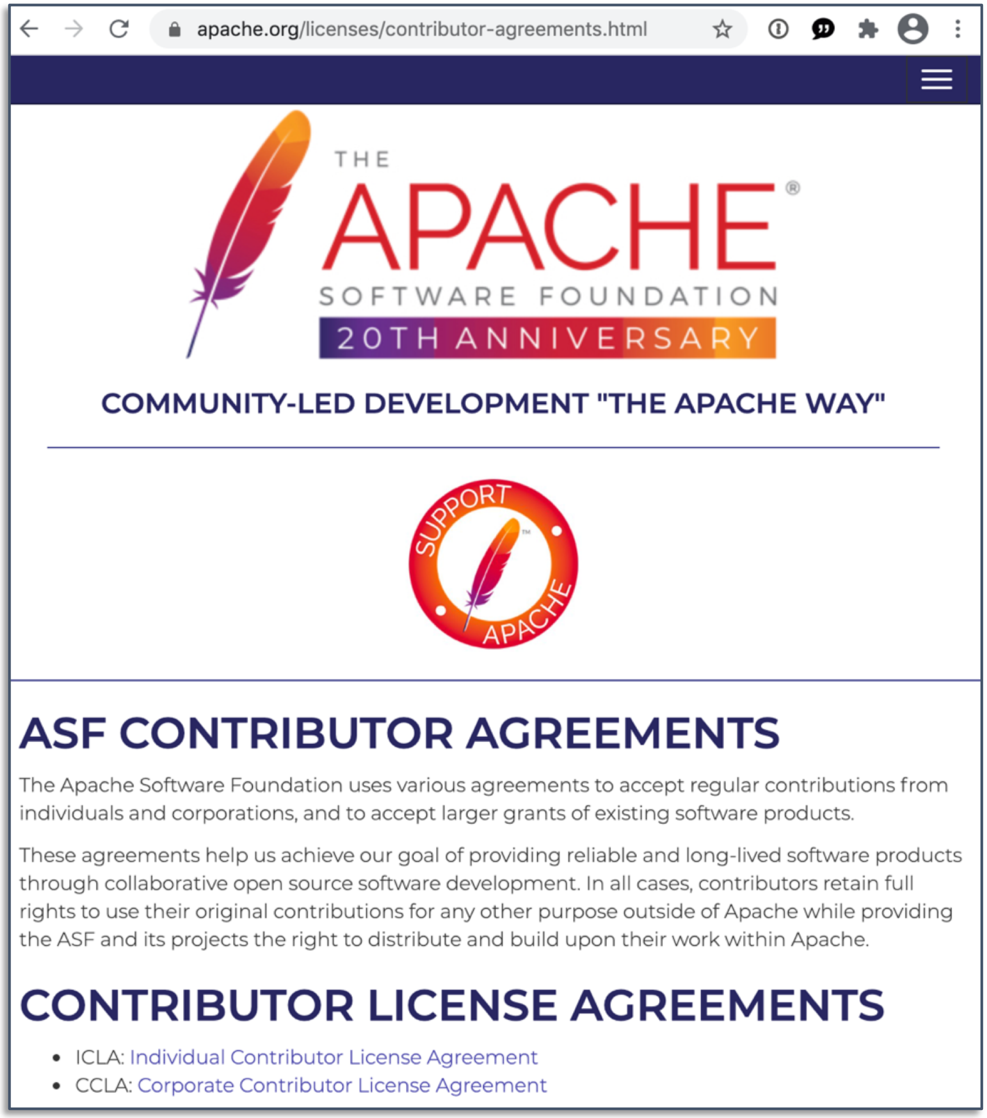
The following is a set of guidelines for contributing to Atom and its packages, which are hosted in the Atom Organization on GitHub. These are mostly guidelines, not rules. Use your best judgment, and feel free to propose changes to this document in a pull request.

Table Of Contents

CLA: The Contributor License Agreement

- Individual license:
  - You contributed in your own time
  - You own your code
  - You can give it away
  - Case for TU Delft students

- Corparate license:
  - You contributed while being paid by a company
  - Company owns your code
  - Company can give it away
  - Case for TU Delft employees


apache.org/licenses/contributor-agreements.html
THE APACHE SOFTWARE FOUNDATION
20TH ANNIVERSARY
SUPPORT APACHE

COMMUNITY-LED DEVELOPMENT "THE APACHE WAY"

ASF CONTRIBUTOR AGREEMENTS

The Apache Software Foundation uses various agreements to accept regular contributions from individuals and corporations, and to accept larger grants of existing software products.

These agreements help us achieve our goal of providing reliable and long-lived software products through collaborative open source software development. In all cases, contributors retain full rights to use their original contributions for any other purpose outside of Apache while providing the ASF and its projects the right to distribute and build upon their work within Apache.

CONTRIBUTOR LICENSE AGREEMENTS

- ICLA: Individual Contributor License Agreement
- CCLA: Corporate Contributor License Agreement

What to Avoid (I)

- One Pull Request doing more than one thing
- PR not addressing an issue (open issue first)
- PR making many small stylistic (subjective) changes
  - Usually these are unpopular (if it ain’t broke don’t fix it)
  - First open issue explaining why you think specific technical debt must be fixed; then offer yourself as volunteer.
- Code not following coding standards / culture (layout, tests, ...)
- Code breaking the automated build

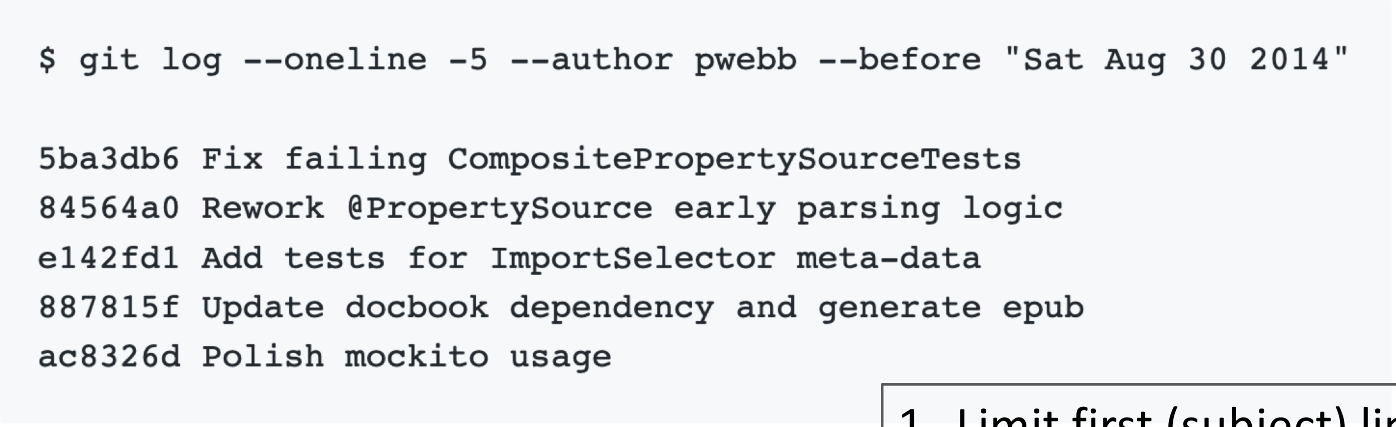
What to Avoid (II)

- Not responding to comments from integrators

- Asking questions without trying to figure them out yourself
  - Better: I searched in A,B,C, but could not find answer to X,Y,Z

- Messy commits in your feature branch
  - Merges from main (master) back into feature branch
  - Unclear commit messages
  - PR on too old main commit
    (rebase feature branch to most recent main commit before creating PR)

Seven Rules of a Great Commit Message



```
$ git log --oneline -5 --author pwebb --before "Sat Aug 30 2014"

5ba3db6 Fix failing CompositePropertySourceTests
84564a0 Rework @PropertySource early parsing logic
e142fd1 Add tests for ImportSelector meta-data
887815f Update docbook dependency and generate epub
ac8326d Polish mockito usage
```

1. Limit first (subject) line to 50 characters
2. Use the imperative mood in subject line
3. Capitalize the subject line
4. Separate subject line from body by new line
5. Do not end subject line with period

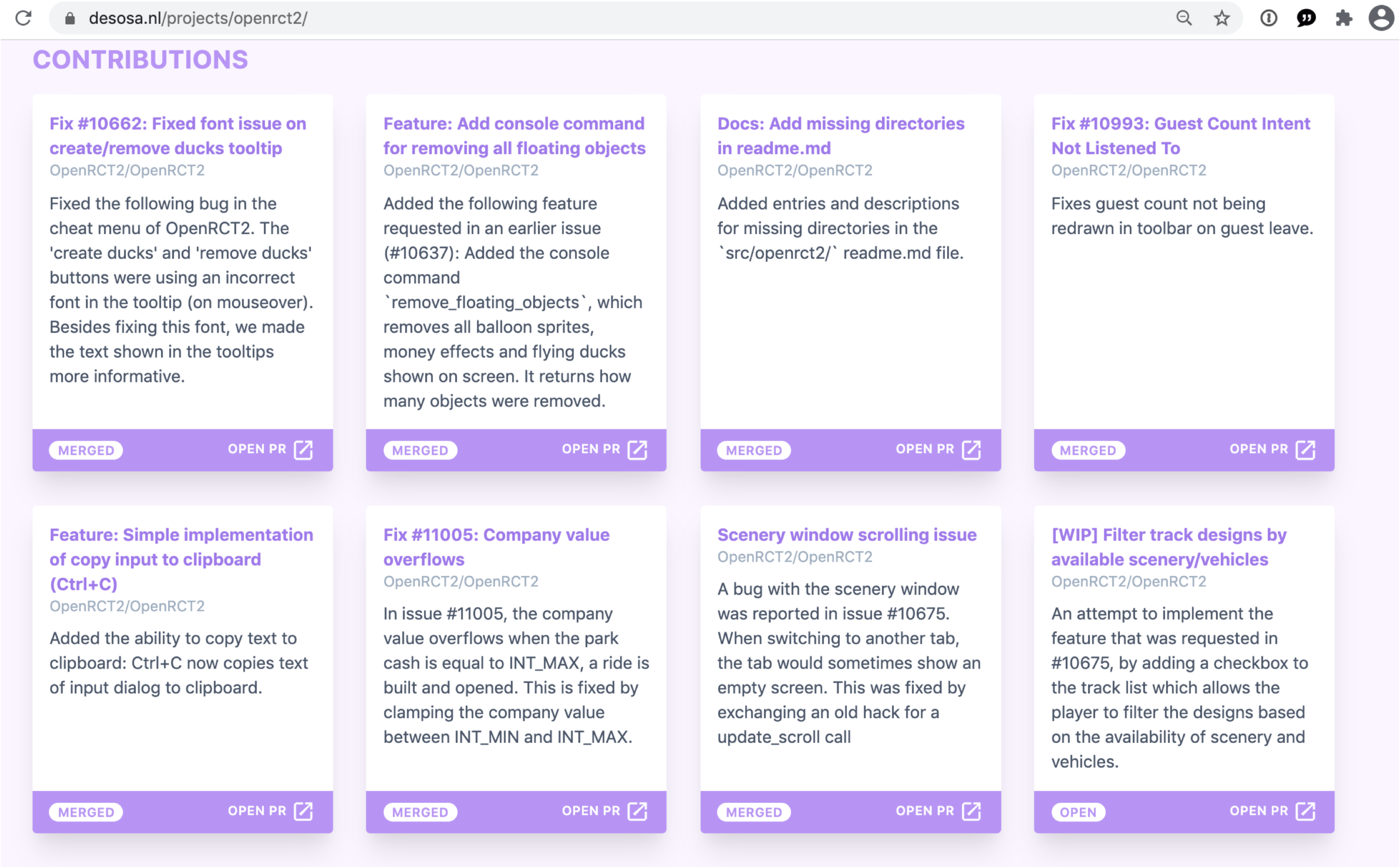
6. Wrap the body at 72 characters
7. Use the body to explain rationale

Contribution done: Reflection Time!

Image credit: wikipedia

- Your own activities:
  - What could you have done better?
  - Who did you interact with?
  - What did you learn?

- The project’s processes and architecture:
  - Did the processes in place help the project achieve its objectives efficiently?
  - Was there friction? What could be improved?
  - Who would you need to convince to make this happen?



CONTRIBUTIONS

Fix #10662: Fixed font issue on create/remove ducks tooltip

OpenRCT2/OpenRCT2

Fixed the following bug in the cheat menu of OpenRCT2. The 'create ducks' and 'remove ducks' buttons were using an incorrect font in the tooltip (on mouseover). Besides fixing this font, we made the text shown in the tooltips more informative.

Feature: Add console command for removing all floating objects

OpenRCT2/OpenRCT2

Added the following feature requested in an earlier issue (#10637): Added the console command `remove_floating_objects`, which removes all balloon sprites, money effects and flying ducks shown on screen. It returns how many objects were removed.

Docs: Add missing directories in readme.md

OpenRCT2/OpenRCT2

Added entries and descriptions for missing directories in the `src/openrct2/` readme.md file.

Fix #10993: Guest Count Intent Not Listened To

OpenRCT2/OpenRCT2

Fixes guest count not being redrawn in toolbar on guest leave.

Feature: Simple implementation of copy input to clipboard (Ctrl+C)

OpenRCT2/OpenRCT2

Added the ability to copy text to clipboard: Ctrl+C now copies text of input dialog to clipboard.

Fix #11005: Company value overflows

OpenRCT2/OpenRCT2

In issue #11005, the company value overflows when the park cash is equal to INT_MAX, a ride is built and opened. This is fixed by clamping the company value between INT_MIN and INT_MAX.

Scenery window scrolling issue

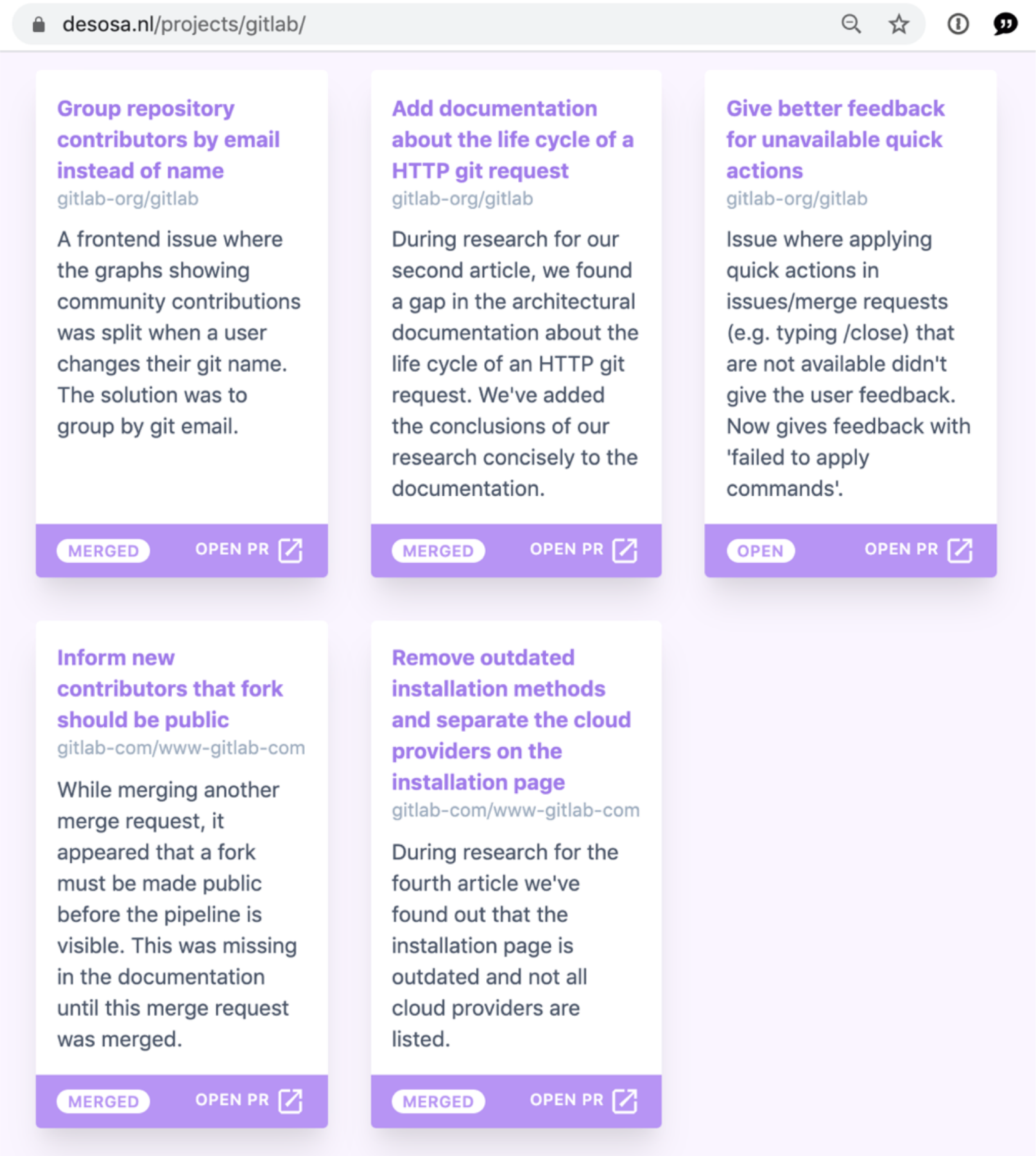
OpenRCT2/OpenRCT2

A bug with the scenery window was reported in issue #10675. When switching to another tab, the tab would sometimes show an empty screen. This was fixed by exchanging an old hack for a update_scroll call

[WIP] Filter track designs by available scenery/vehicles

OpenRCT2/OpenRCT2

An attempt to implement the feature that was requested in #10675, by adding a checkbox to the track list which allows the player to filter the designs based on the availability of scenery and vehicles.



Group repository contributors by email instead of name
gitlab-org/gitlab

A frontend issue where the graphs showing community contributions was split when a user changes their git name. The solution was to group by git email.

Add documentation about the life cycle of a HTTP git request
gitlab-org/gitlab

During research for our second article, we found a gap in the architectural documentation about the life cycle of an HTTP git request. We've added the conclusions of our research concisely to the documentation.

Give better feedback for unavailable quick actions
gitlab-org/gitlab

Issue where applying quick actions in issues/merge requests (e.g. typing /close) that are not available didn't give the user feedback. Now gives feedback with 'failed to apply commands'.

Inform new contributors that fork should be public
gitlab-com/www-gitlab-com

While merging another merge request, it appeared that a fork must be made public before the pipeline is visible. This was missing in the documentation until this merge request was merged.

Remove outdated installation methods and separate the cloud providers on the installation page
gitlab-com/www-gitlab-com

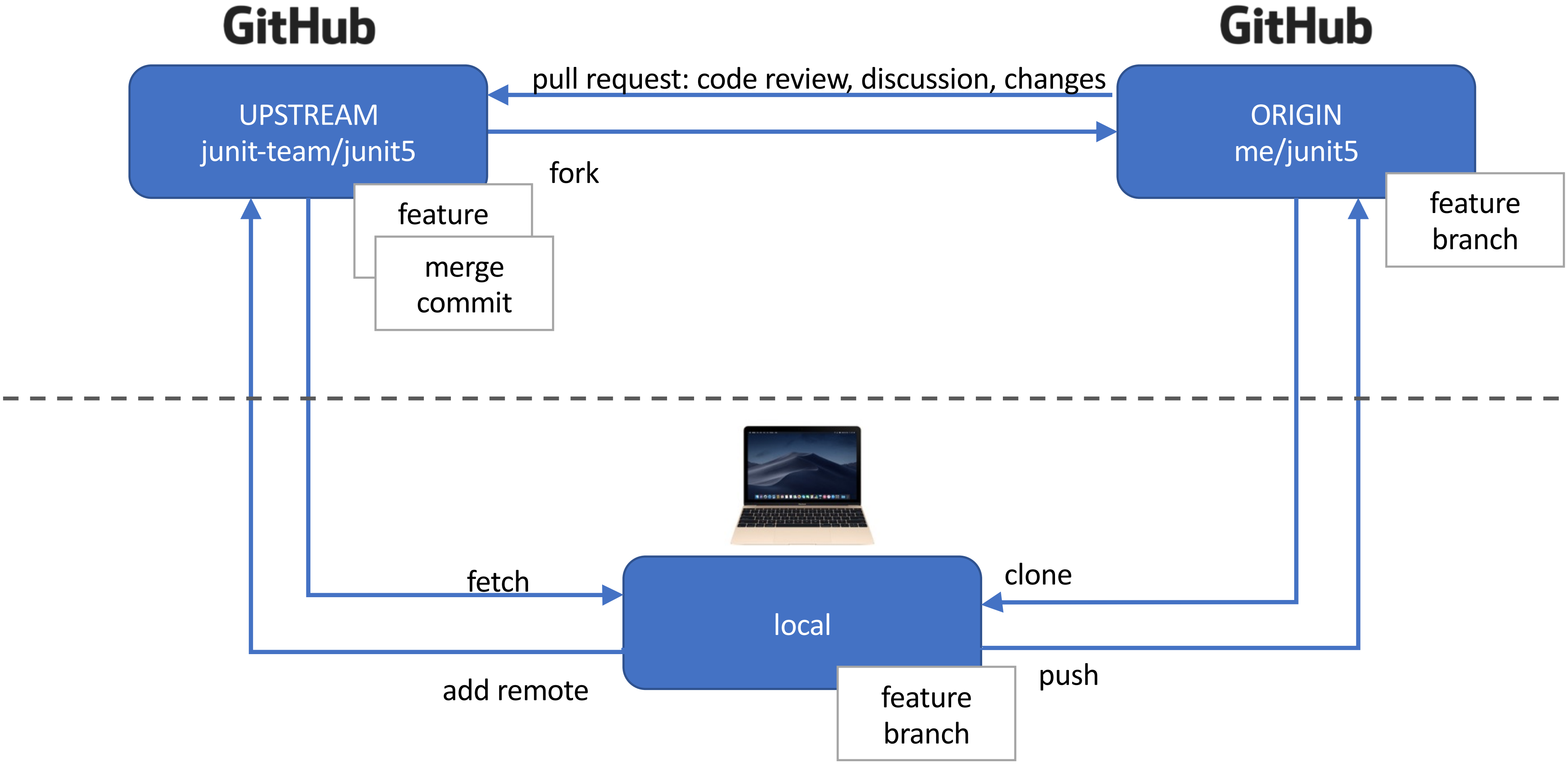
During research for the fourth article we've found out that the installation page is outdated and not all cloud providers are listed.

Further Resources

- How to Contribute to Open Source
  https://opensource.guide/how-to-contribute/

- The Beginner's Guide to Open Source
  https://blog.newrelic.com/tag/open-source-best-practices

- How to Write a Git Commit Message
  https://chris.beams.io/posts/git-commit/



GitHub

UPSTREAM
junit-team/junit5

GitHub

ORIGIN
me/junit5

pull request: code review, discussion, changes

fork

feature

merge
commit

feature
branch

local

fetch

clone

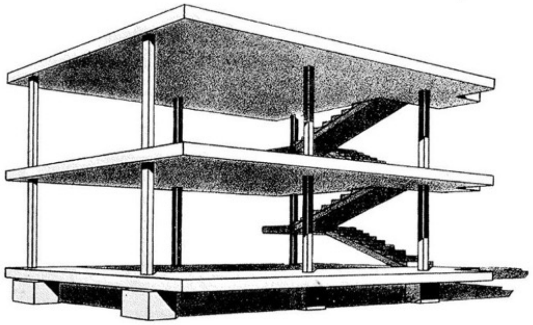
add remote

push

feature
branch

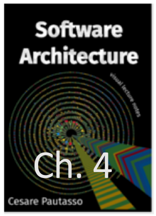
Software Architecture: Views and Models

Arie van Deursen



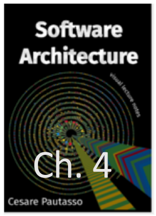
Wikipedia, Dom-Ino House, Corbusier

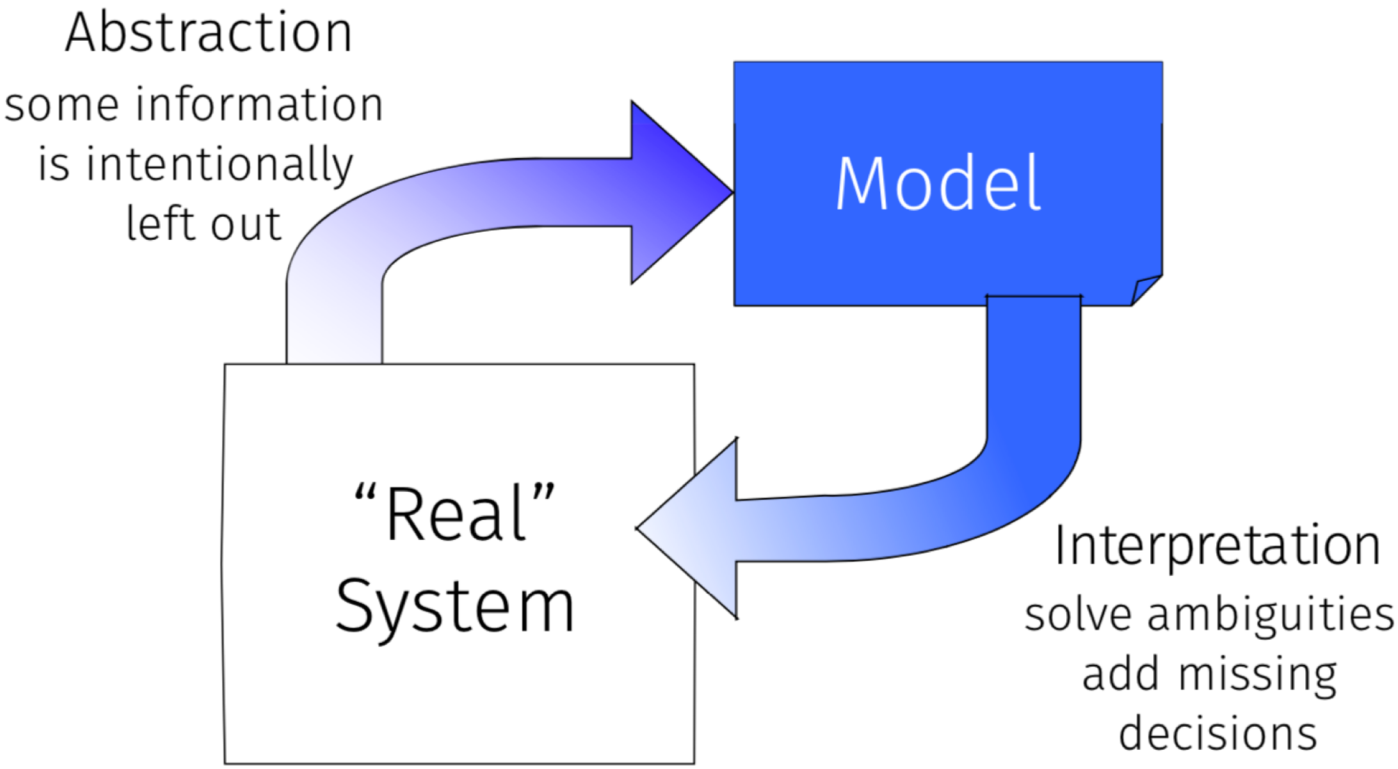
Capturing the Architecture



- Every system has an architecture
  - Some architectures are manifest and visible, many others are not
- 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.

Abstraction and Interpretation





Abstraction

some information is intentionally left out

Model

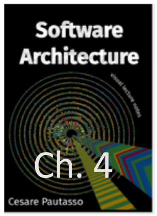
"Real" System

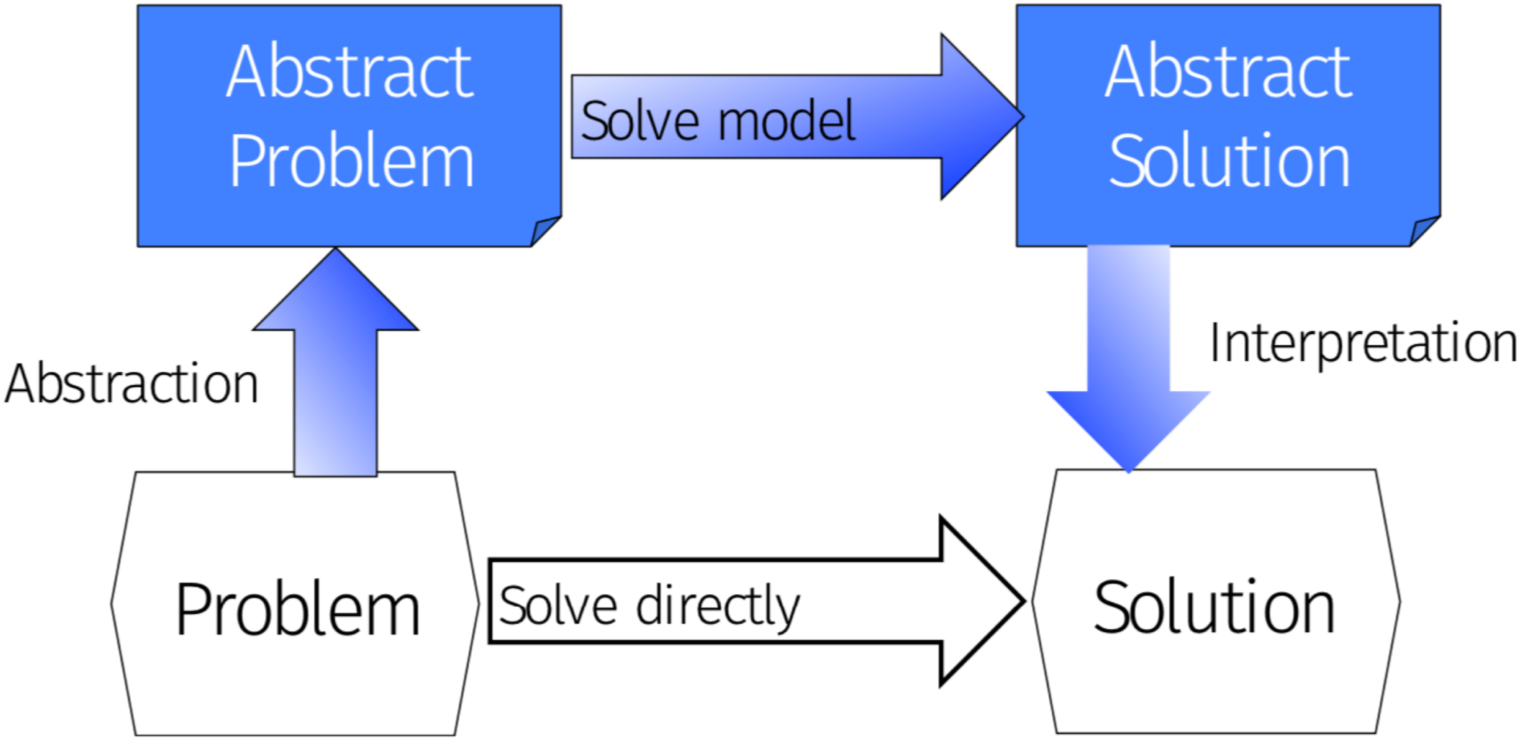
Interpretation

solve ambiguities add missing decisions

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

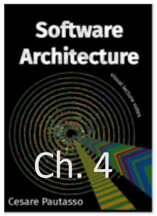
Solving Problems with Models





Mary Shaw

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



Question First, Model Second

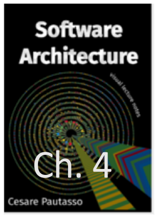
- Different models have different purposes
- Know what questions you want the model to answer before you build it

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

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

http://www.codingthearchitecture.com/2015/01/08/shneidermans_mantra.html

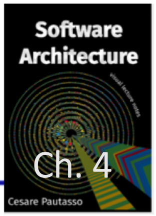
The “Domain Model”



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

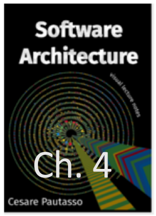
- Problem domain description:
    - Information (invariants, navigation, snapshots)
- Functionality (use-case scenarios, feature models)
- Define shared vocabulary and understanding towards your customer, domain expert

Design Model



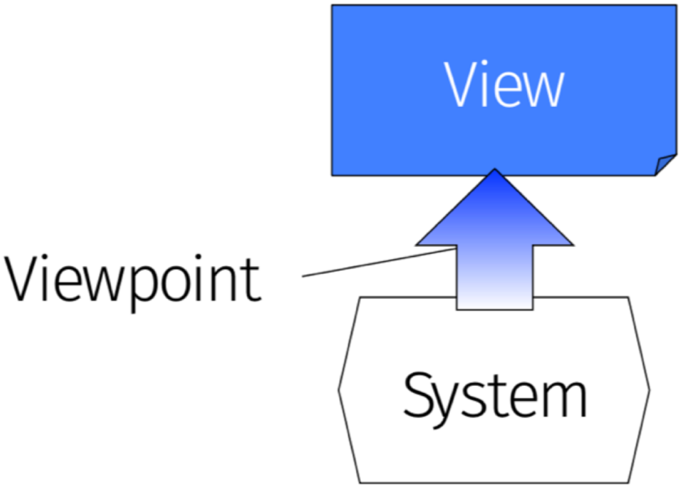
- Refutable truths about your system
- Within your control
- Prescriptive: Your system will be built based on it
- Descriptive: Your system is represented by it
- Interfaces (externally visible behavior, data interchange)
- Quality Attributes (how to achieve them)
- Structural decomposition, component assembly
- Define shared vocabulary and understanding within the development team

What is a view?

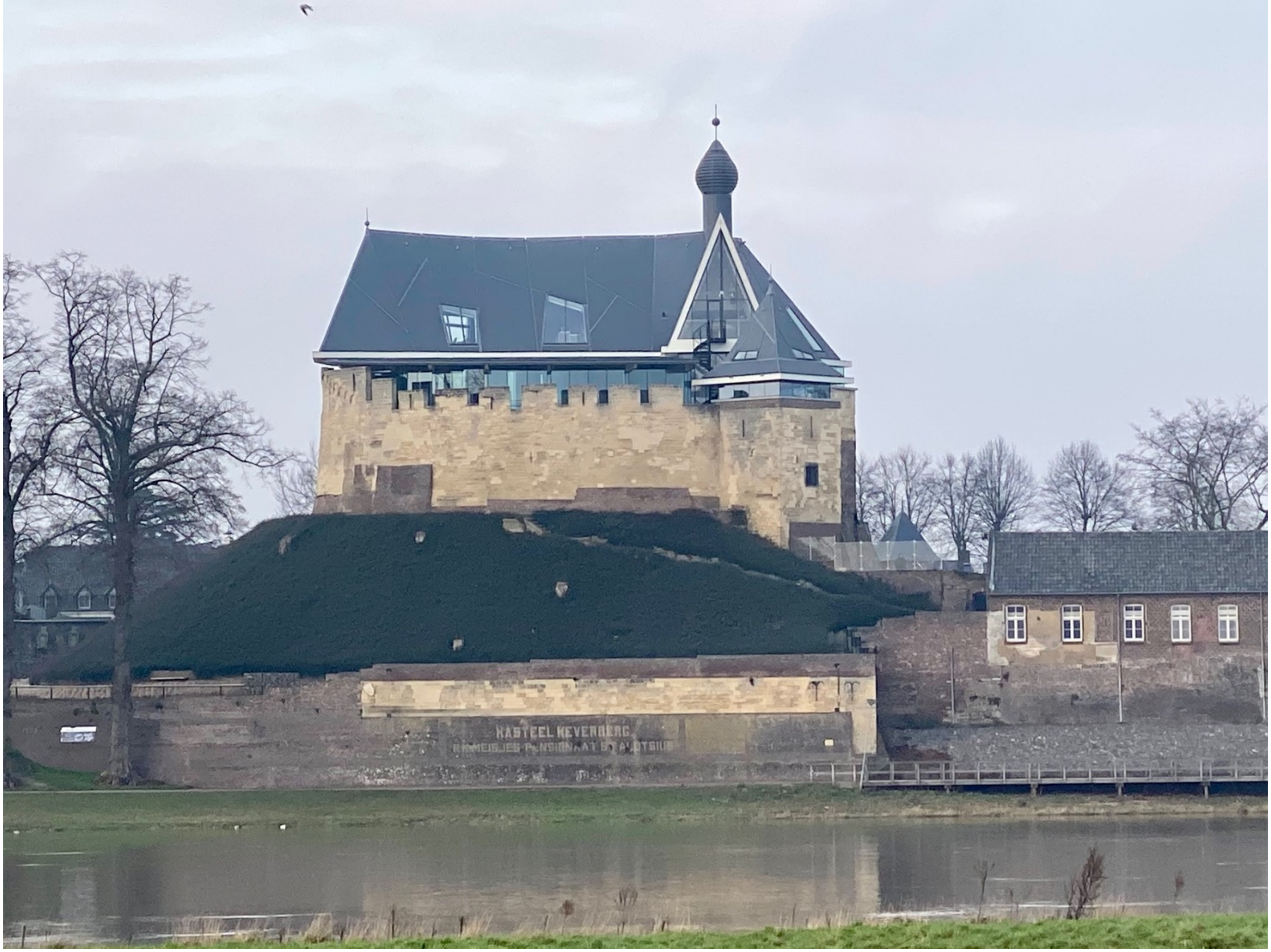


- No single modeling approach can capture the entire complexity of a software architecture
- Various parts of the architecture (or views) may have to be modeled with a different:
  - Notation
  - Level of detail
  - Target Audience

- A **view** is a set of design decisions related by common concerns (the viewpoint)

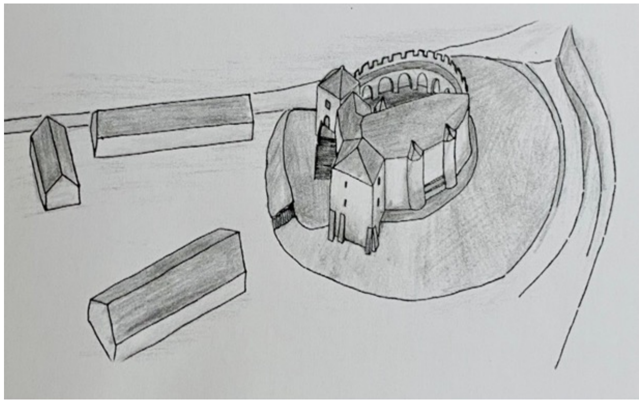


Views on Kessel Castle Keverberg

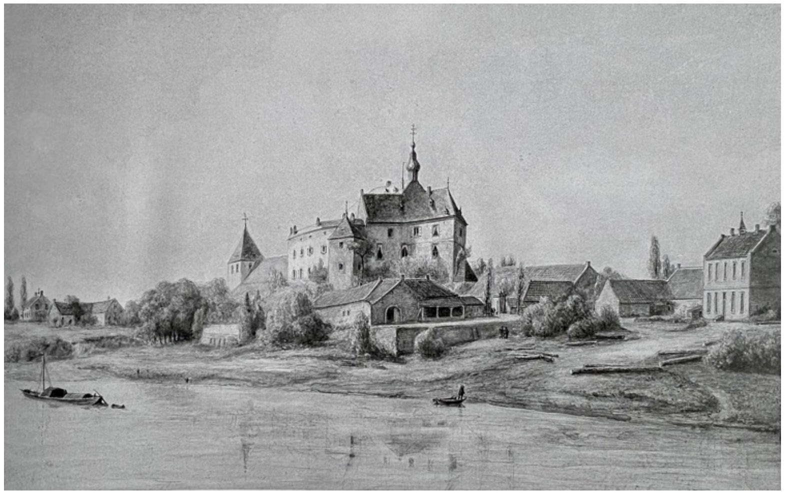


The legacy view

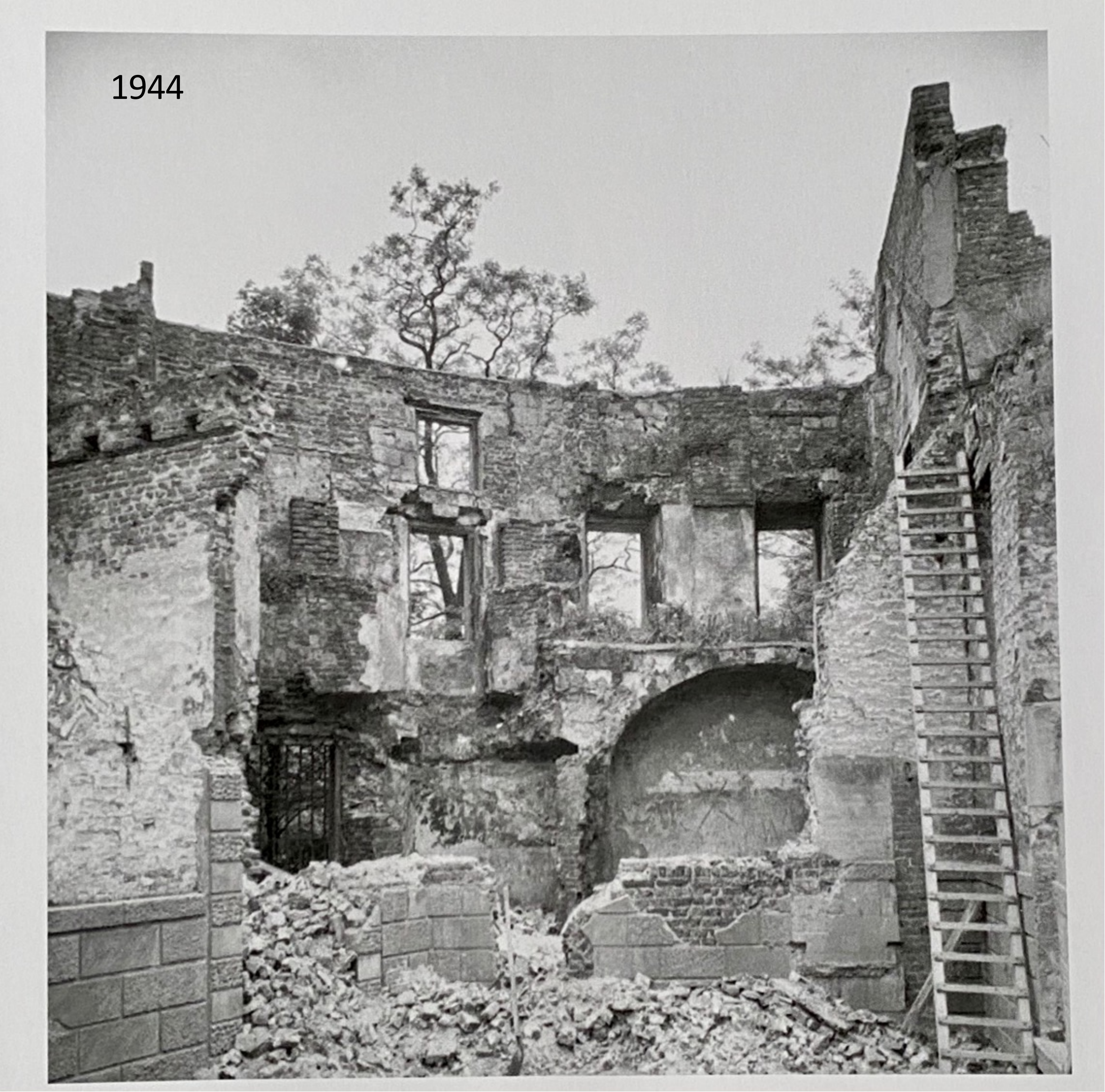
1400
(motte)



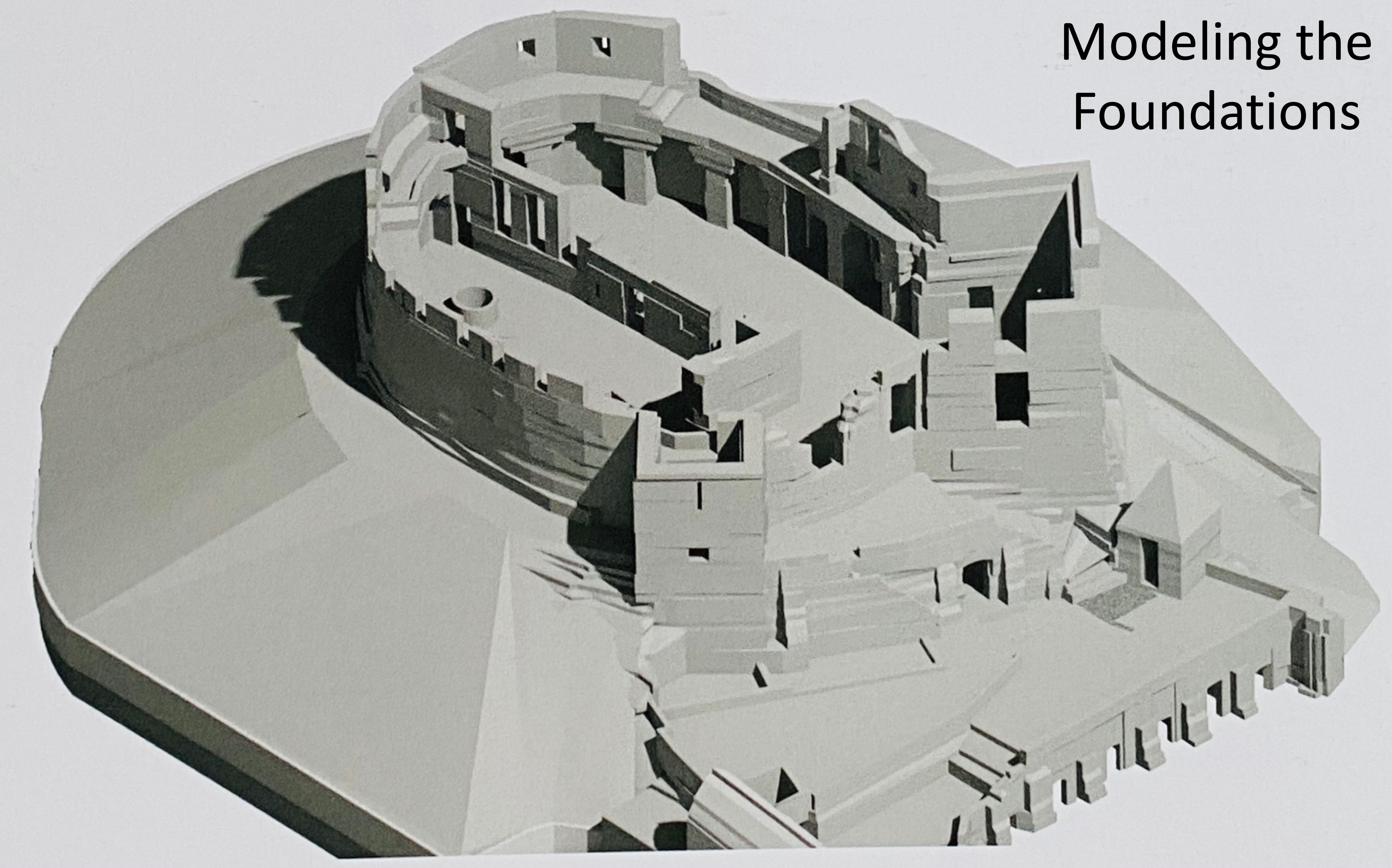
1850

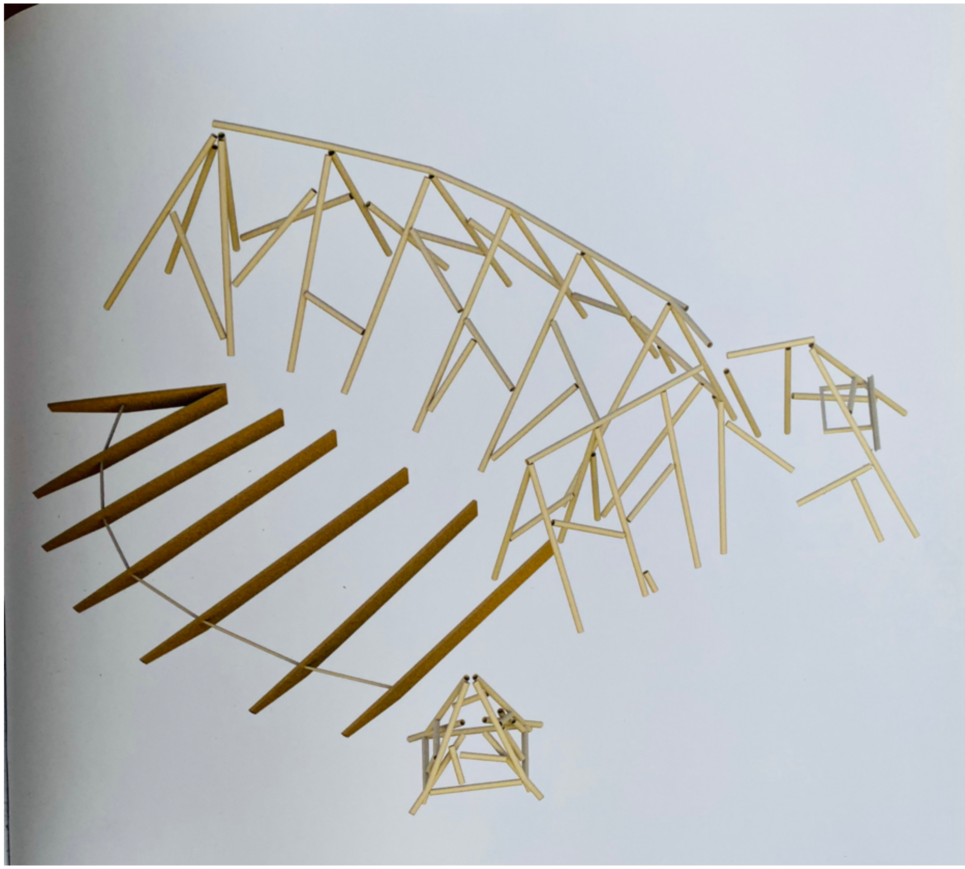


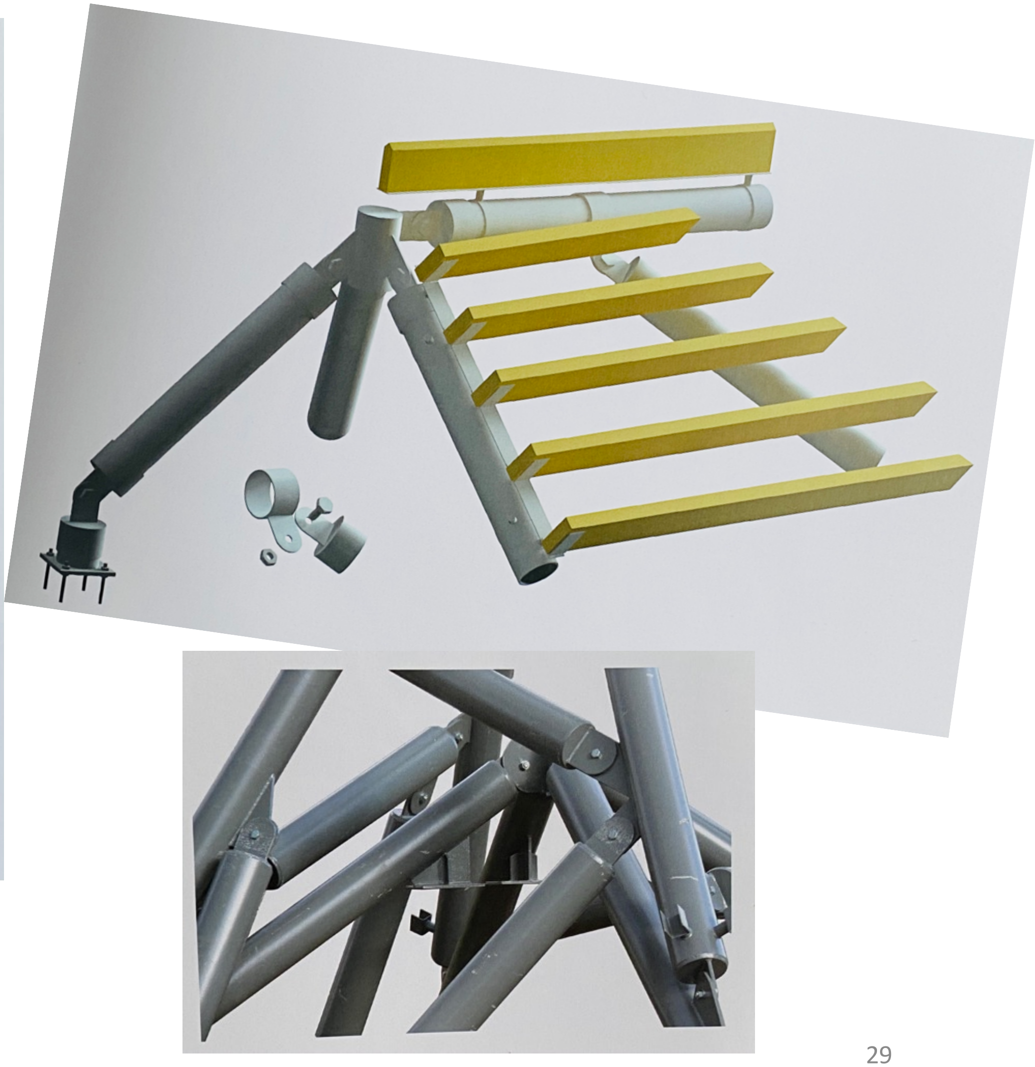
1944



Modeling the Foundations

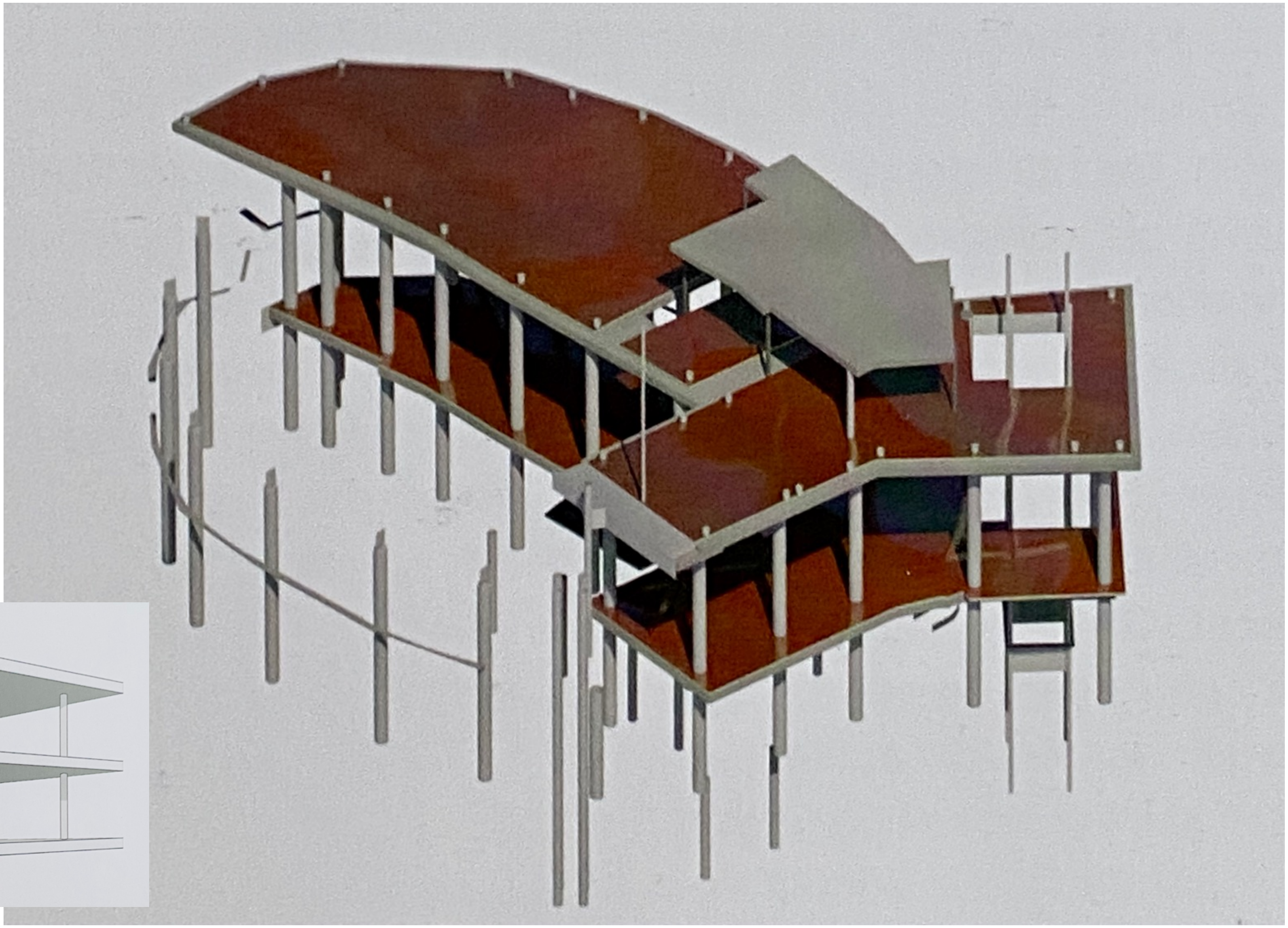




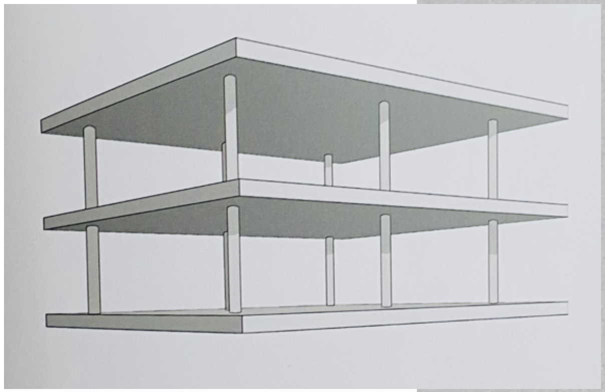


A view on the roof

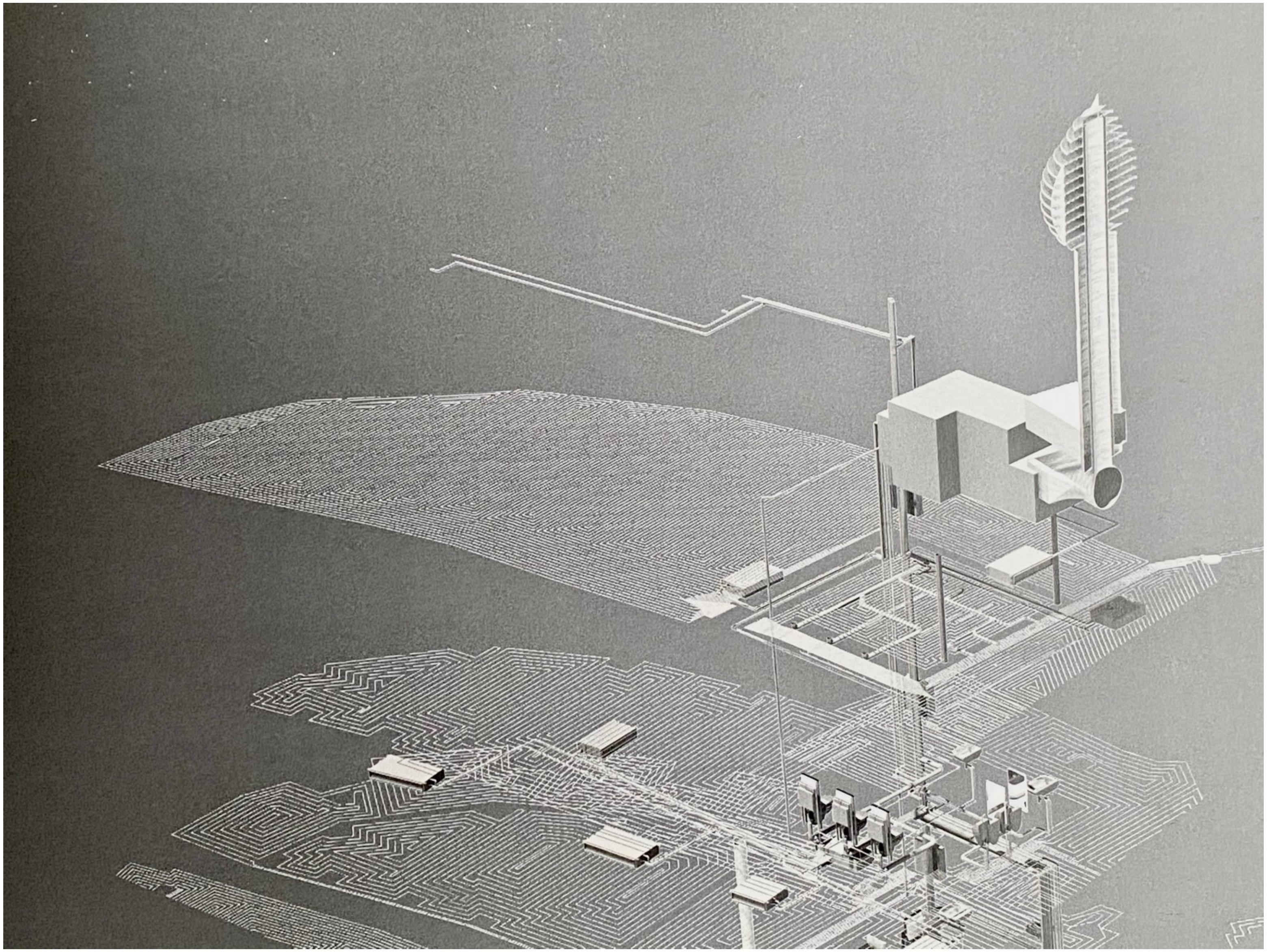
A view on the floors



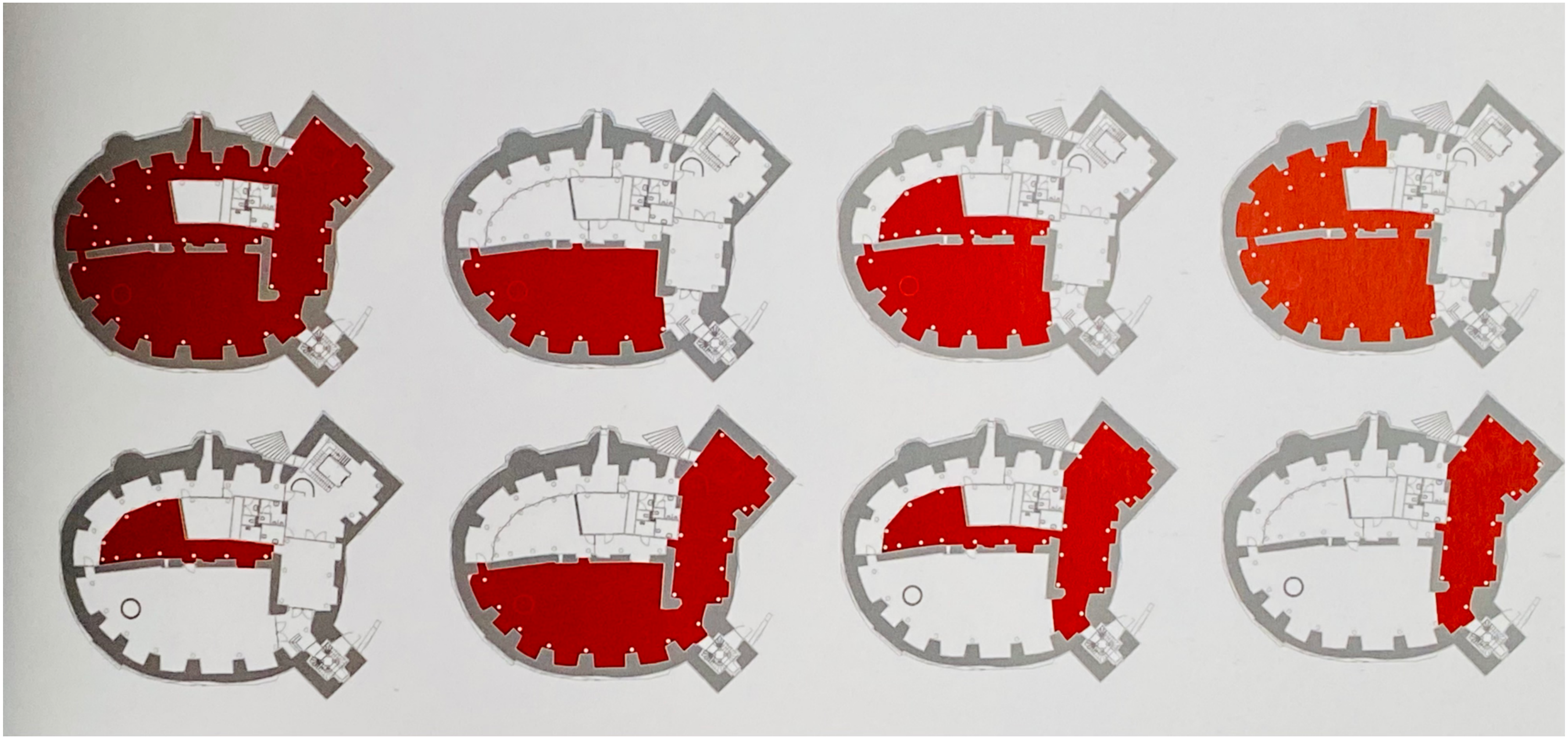
Design pattern from Le Corbusier



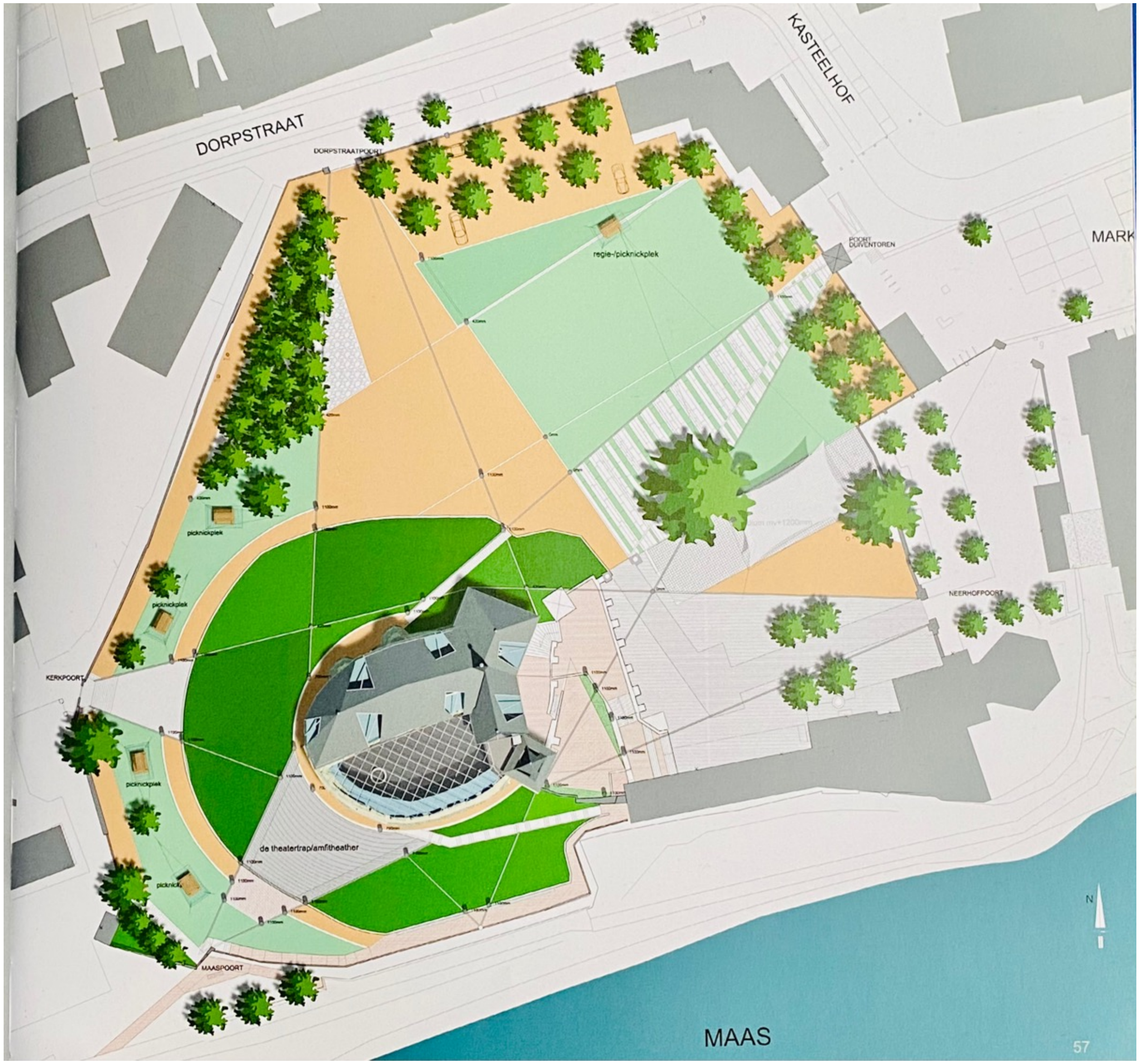
A view
on the
air flow



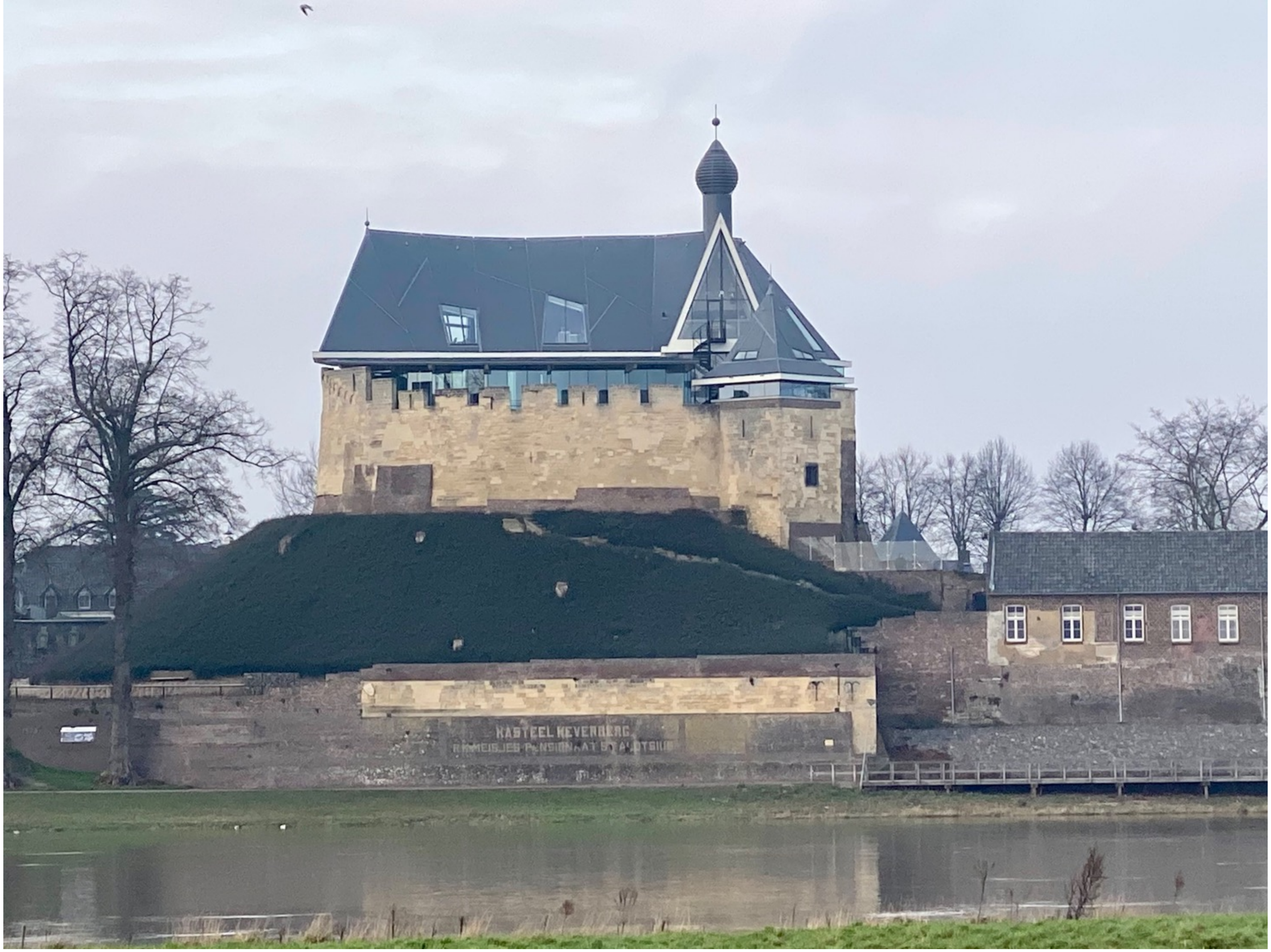
The Room Configuration View



A view on the context

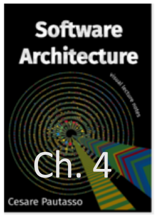


Views on Kessel Castle Keverberg



Reconstruction 2015

How many views?



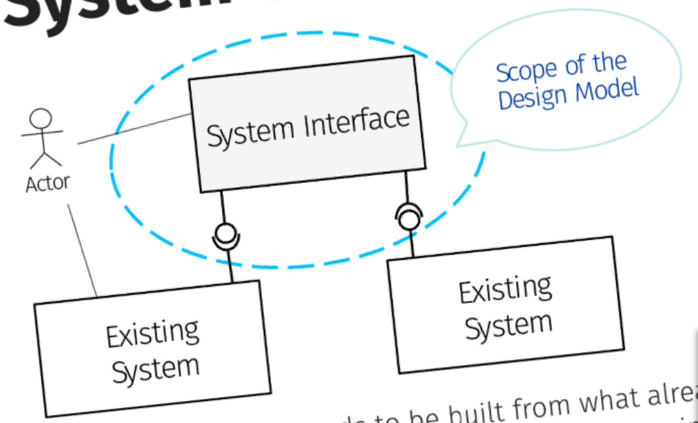
Ch. 4

- System Context
- Functional
- Logical
- Physical
- Deployment
- Development
- Information
- Process
- Concurrency
- Operational

- Security
- Performance and Scalability
- Availability and Reliability
- Evolution
- Teachability ("Welcome to the team")
- Regulatory
- Marketing
- Business Impact

Rozanski & Woods

System Context View

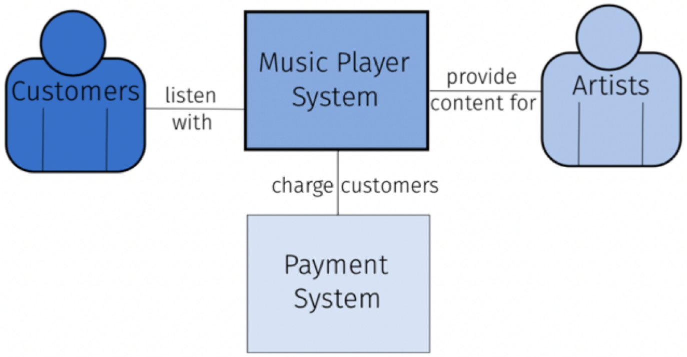


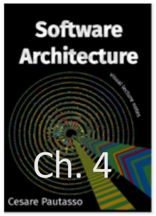
- Distinguish what needs to be built from what alrea
define the dependencies and the integration poin

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





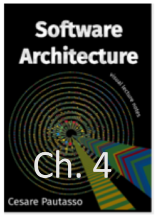
Containers View

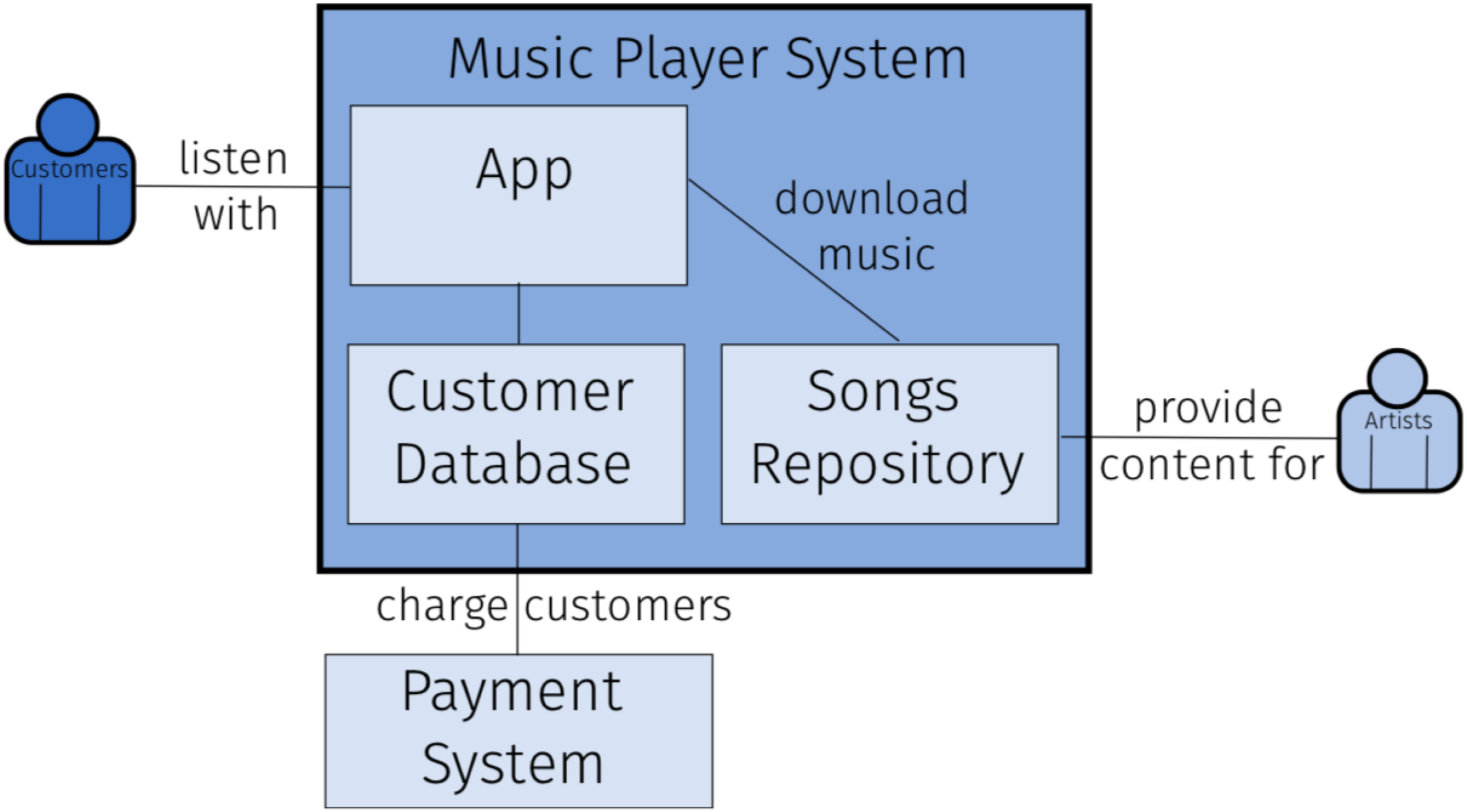
- What are the main logical execution environments in which the system can run?
- Containers can be deployed separately and independently evolved
- Container: architectural abstraction (beyond Docker)

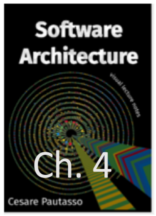
Examples:

- 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

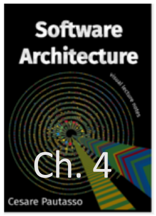




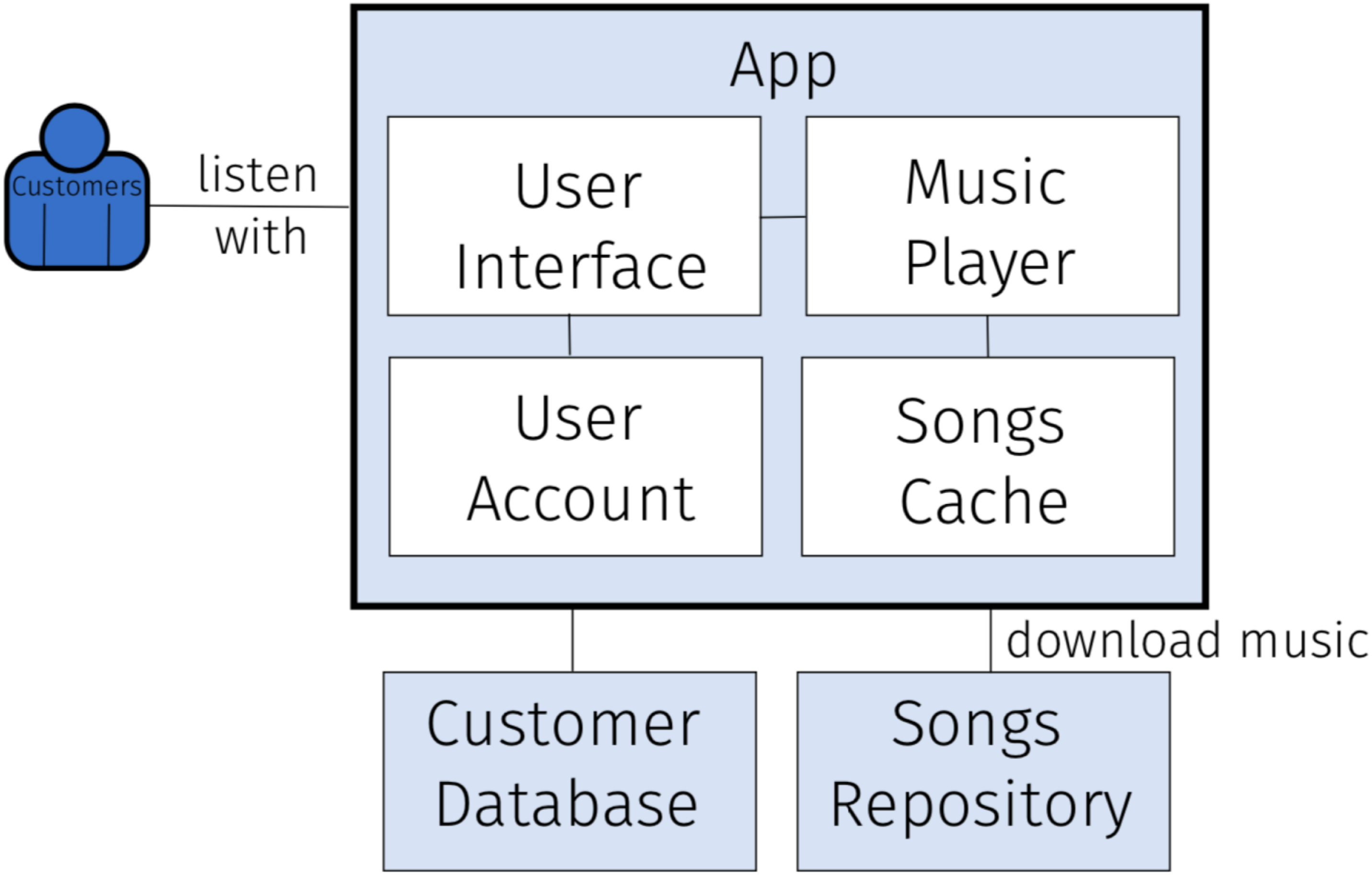


Components View

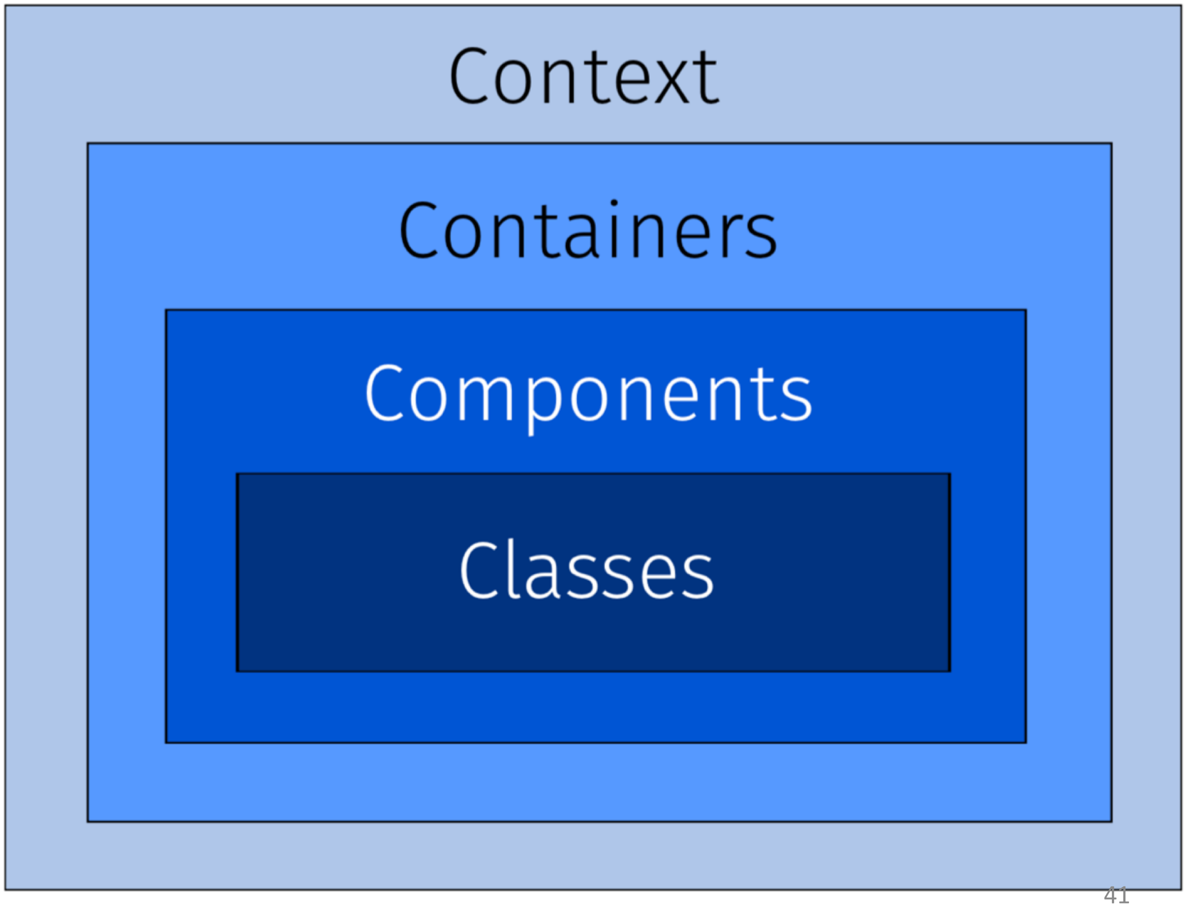
- What is the structural decomposition of the software with related functionality encapsulated behind a well-defined interface?
- What are the dependencies between components?
- Are there shared components that will be deployed in multiple containers?
- What is the technology used to build the components?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



Components View Example



C4



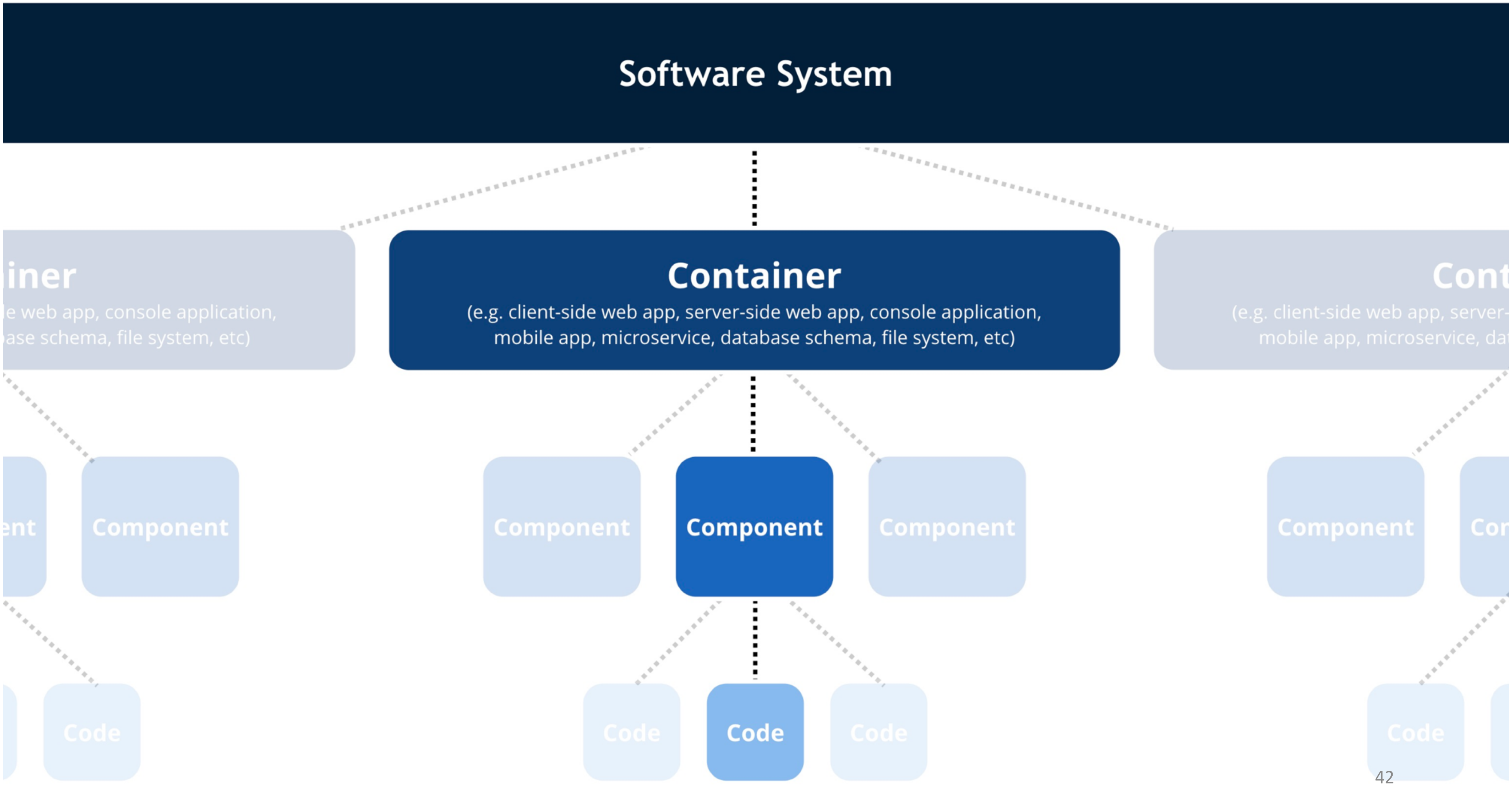
Software Architecture
for Developers

Volume 2

Visualise, document and explore your software architecture

Simon Brown

https://c4model.com/



Software System

iner

le web app, console application,

base schema, file system, etc)

Container

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

Cont

(e.g. client-side web app, server-

mobile app, microservice, dat

ent

Component

Component

Component

Component

Component

Con

Code

Code

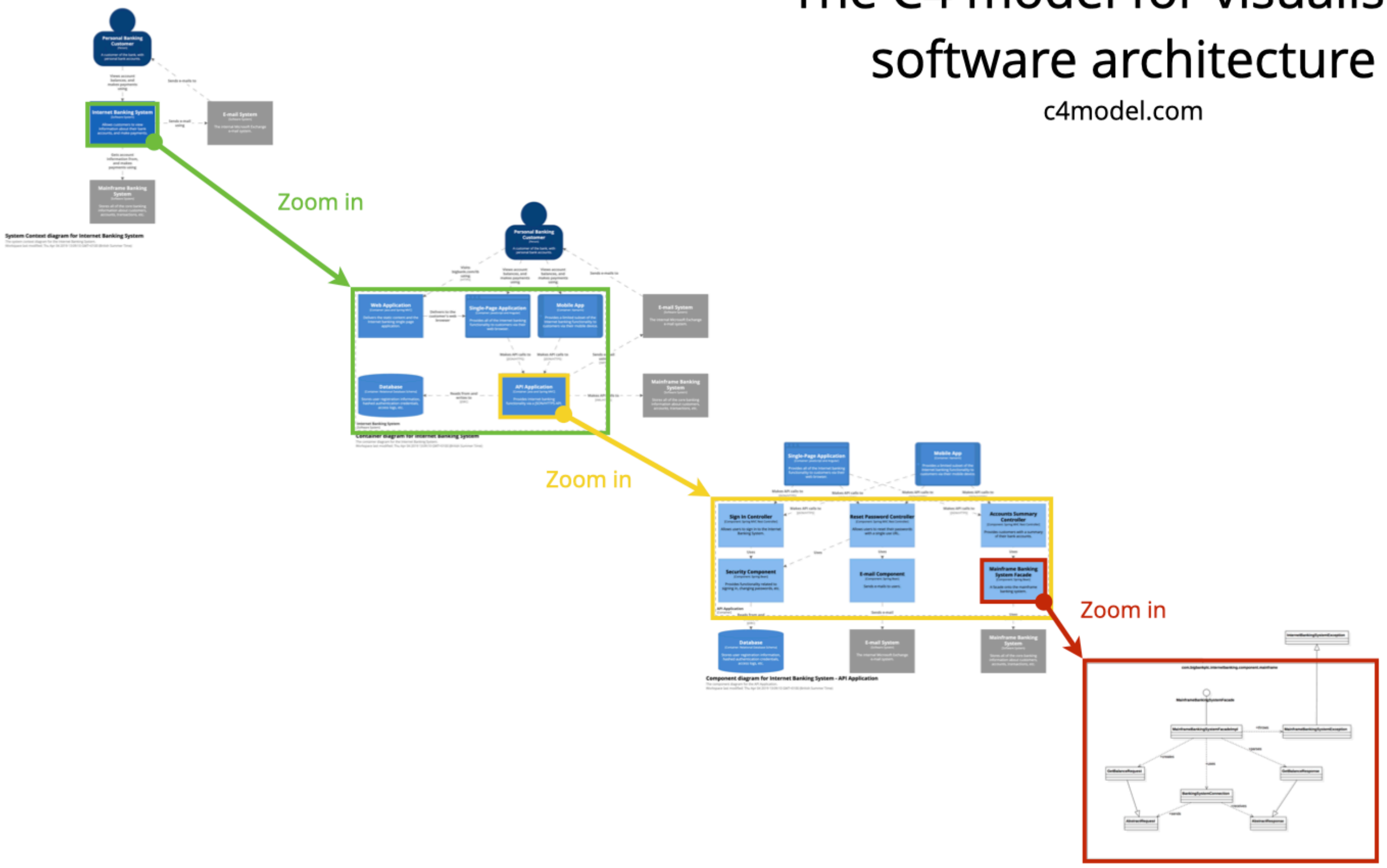
Code

Code

Code

The C4 model for visualising software architecture

c4model.com

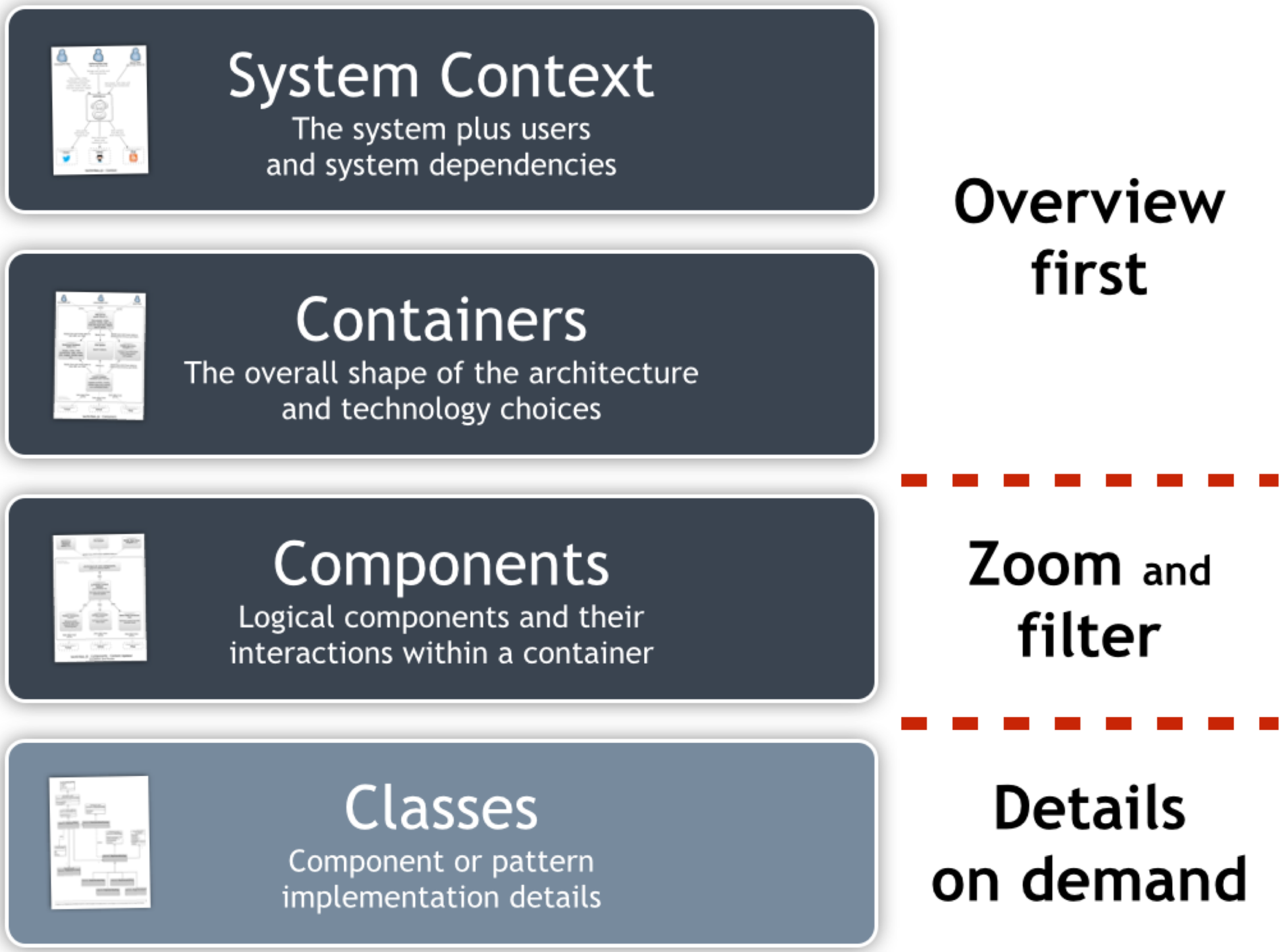

Personal Banking Customer
Internet Banking System
E-mail System
Mainframe Banking System
System Context diagram for Internet Banking System
Zoom in
Web Application
Single-Page Application
Mobile App
Database
API Application
Container diagram for Internet Banking System
Zoom in
Sign In Controller
Reset Password Controller
Accounts Summary Controller
Security Component
E-mail Component
Mainframe Banking System Facade
Component diagram for Internet Banking System - API Application
Zoom in

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

Overview
first

Components

Logical components and their
interactions within a container

Zoom and
filter

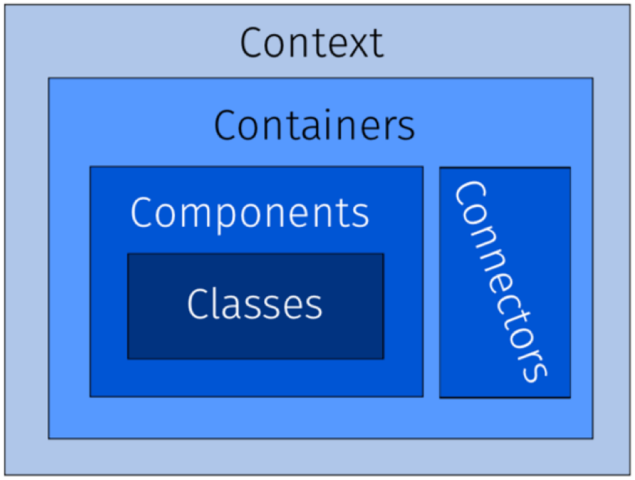
Classes

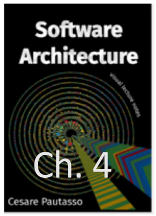
Component or pattern
implementation details

Details
on demand

http://www.codingthearchitecture.com/2015/01/08/shneidermans_mantra.html

C5

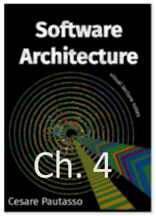




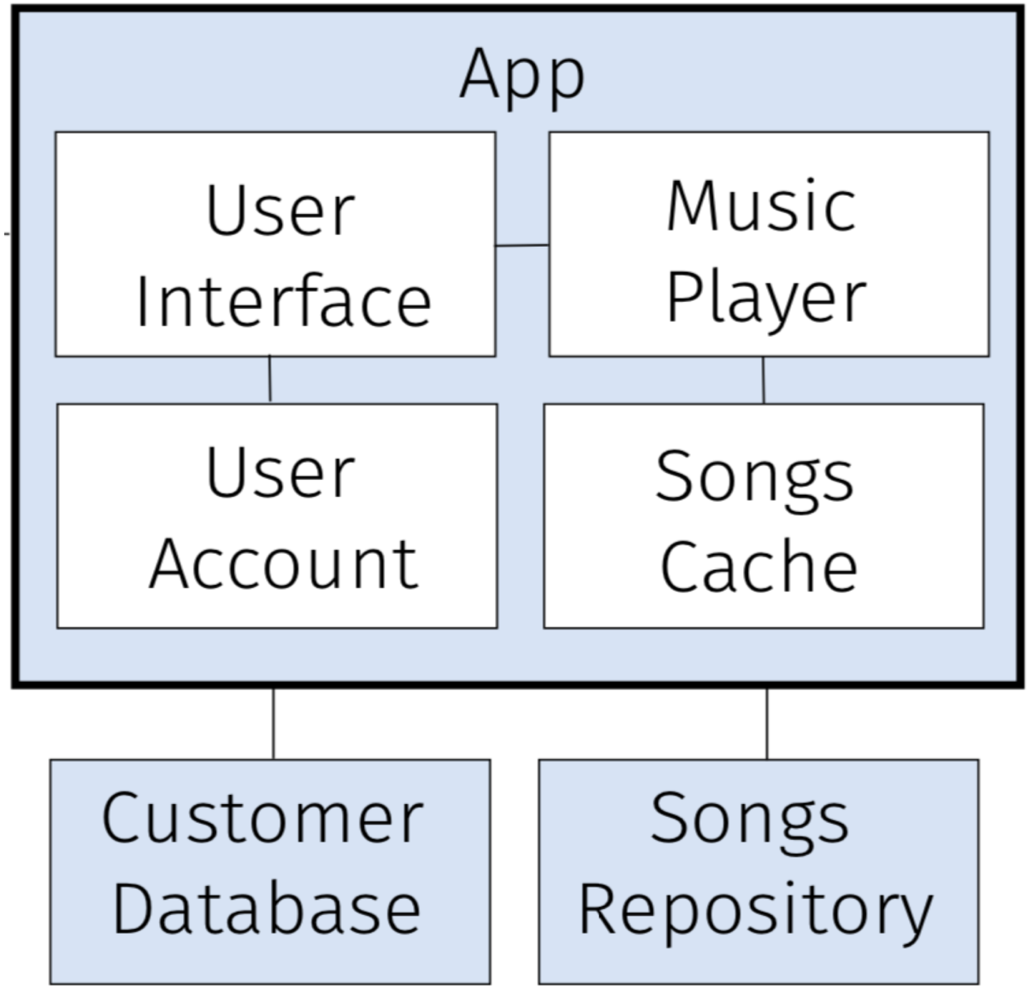
Connectors View

- How are component interfaces interconnected?
- What kind of connector(s) are chosen?
- What is the amount of coupling between components?

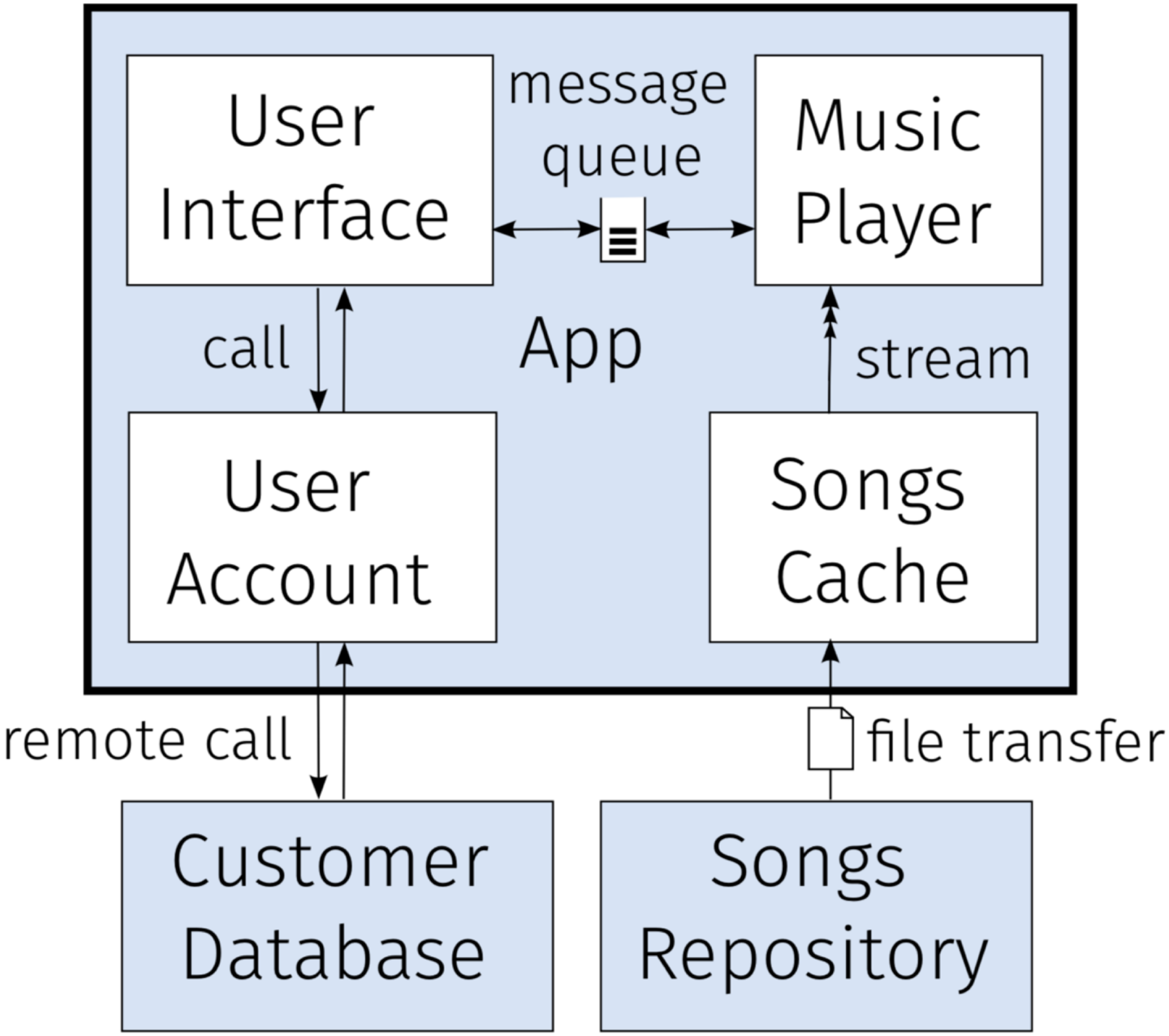
These decisions may depend on the deployment configuration

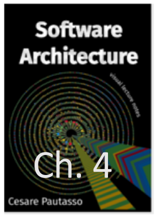


Can you think of a (different) type of connector for each line between two components?

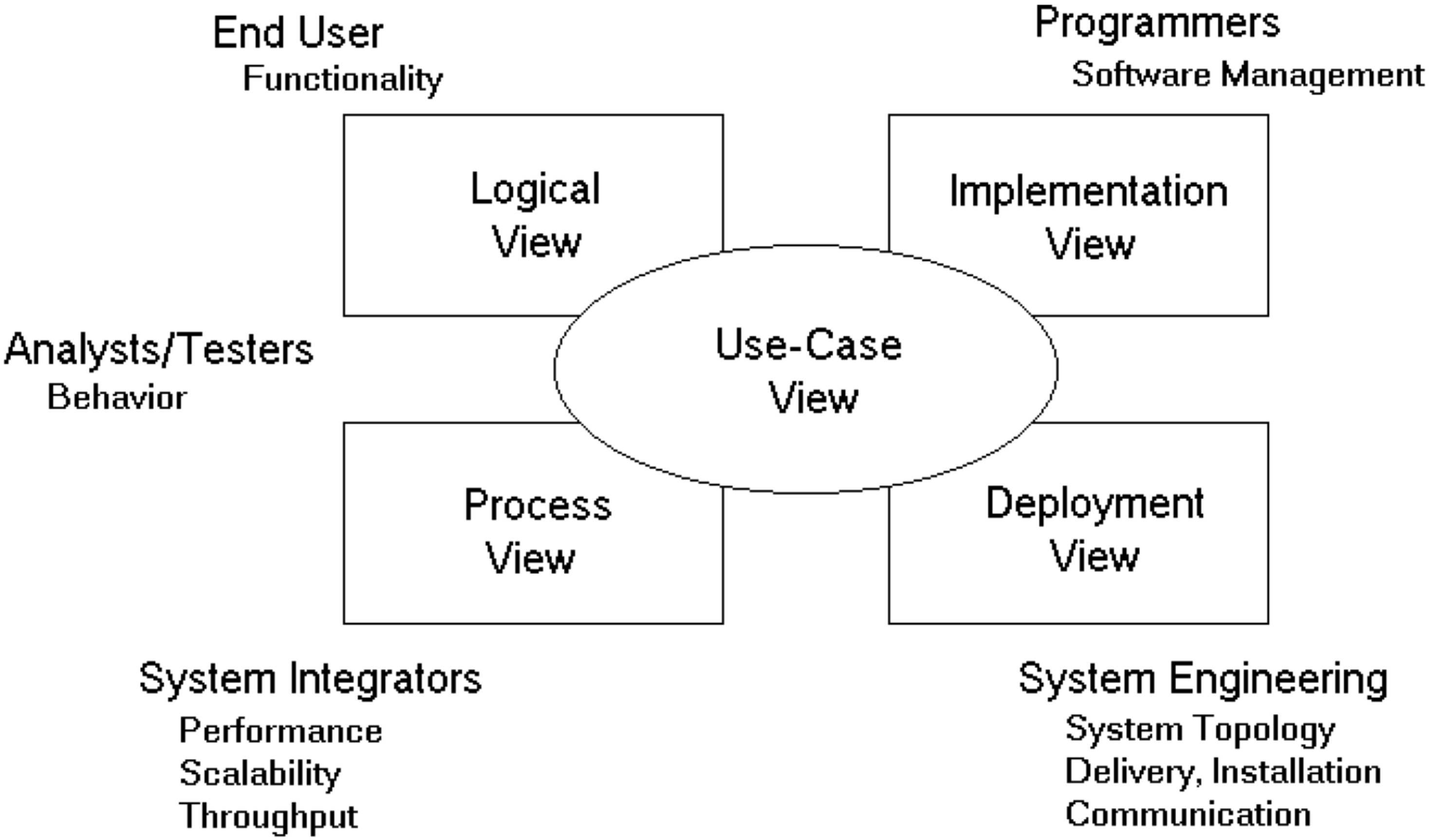


Connectors View Example





Philippe Kruchten’s “4+1 Views”



End User
Functionality

Programmers
Software Management

Logical View

Implementation View

Use-Case View

Analysts/Testers
Behavior

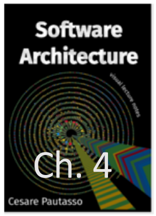
Process View

Deployment View

System Integrators
Performance
Scalability
Throughput

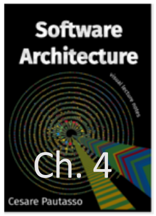
System Engineering
System Topology
Delivery, Installation
Communication

IEEE Software, November 1995

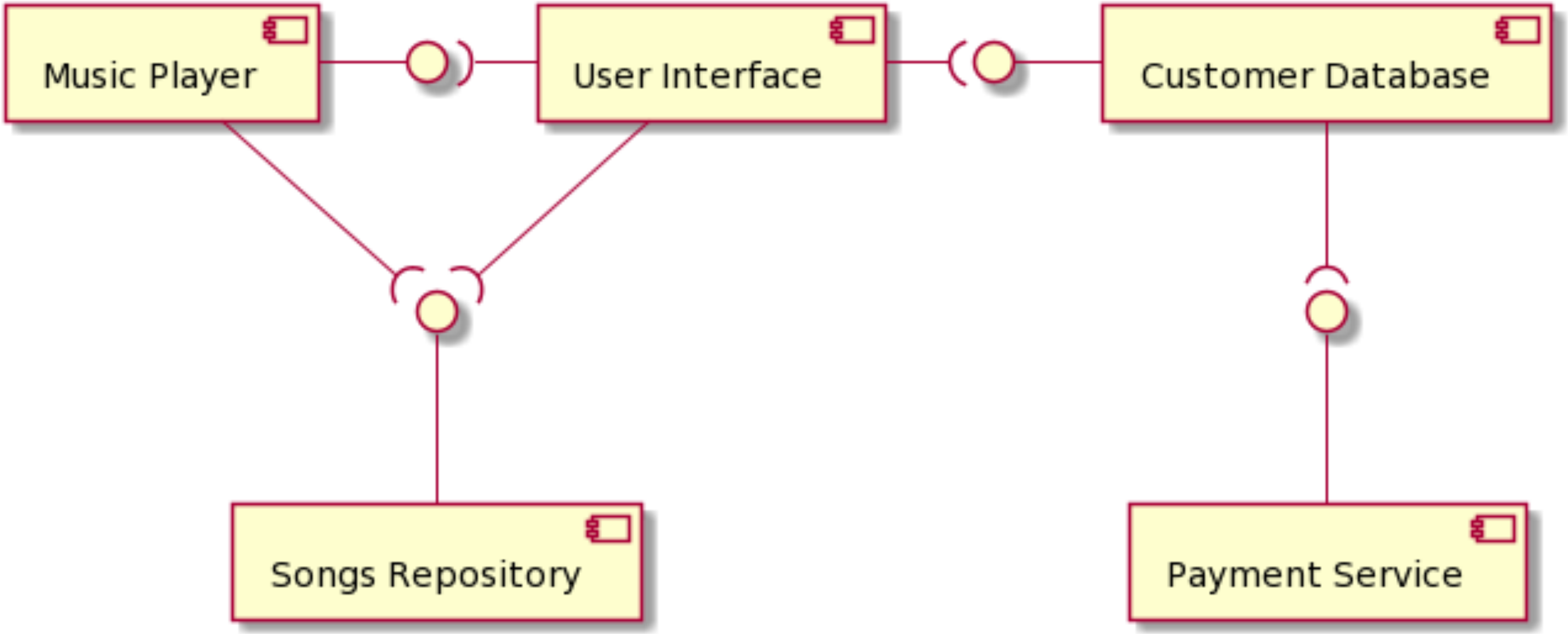


Kruchten’s “Logical View”

- Similar to C4 component view
- Decompose the system structure into software components and connectors
- Map functionality/requirements/use cases onto the components
- Concern: Functionality
- Target Audience: Developers and Users

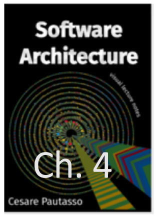


Example Logical View

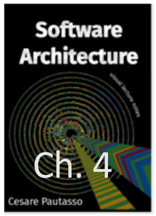


www.plantuml.com

Kruchten’s “Process View”

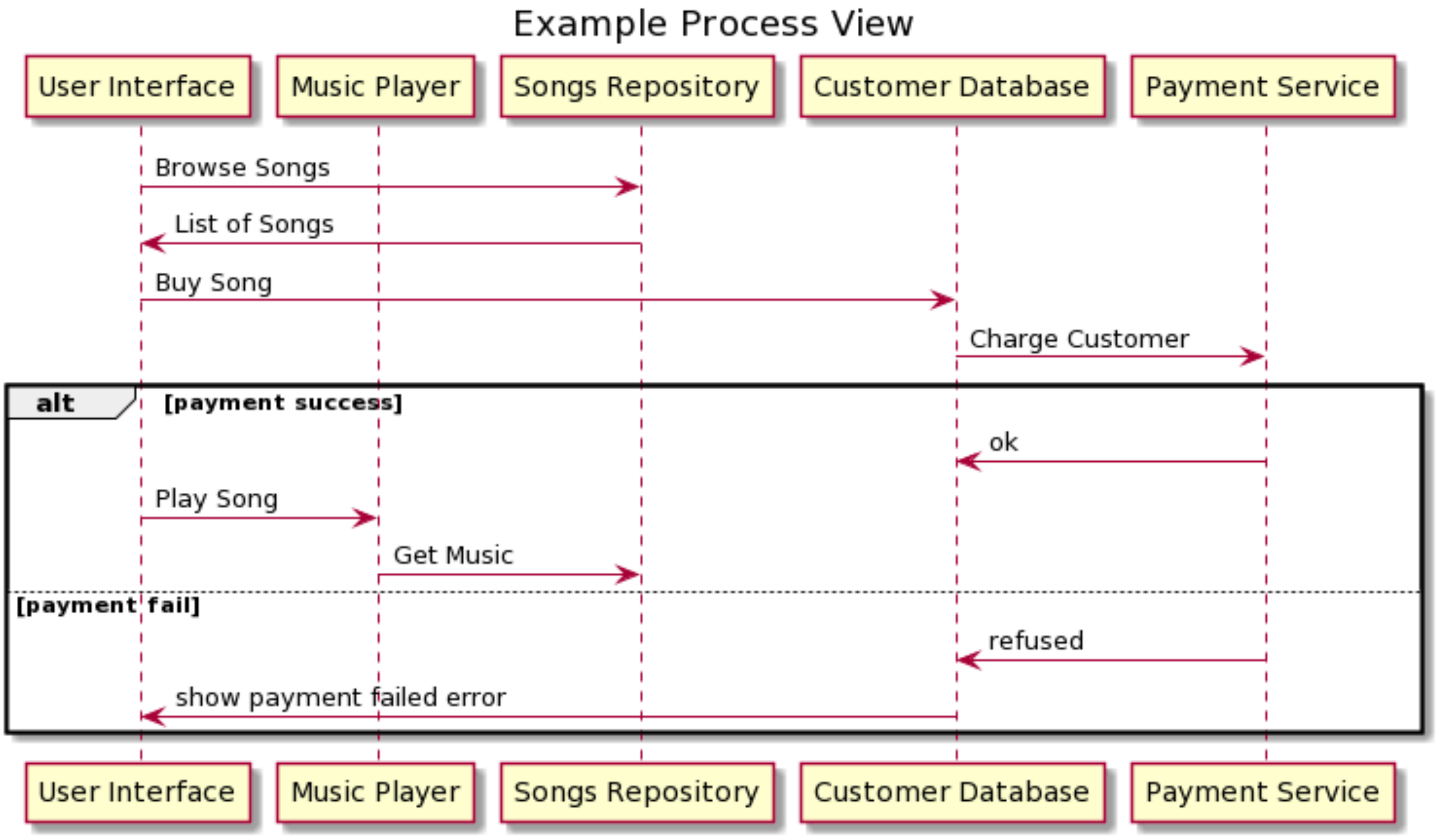


- Model the dynamic aspects of the architecture:
  - Which are the active components?
  - Are there concurrent threads of control?
  - Are there multiple distributed processes in the system?
  - What is the behavior of (parts of) the system?
- Describe how processes/threads communicate (e.g., remote procedure call, messaging connectors)
- Concern: Functionality, Performance
- Target Audience: Developers


Software Architecture

Ch. 4

Cesare Pautasso


Example Process View

User Interface

Music Player

Songs Repository

Customer Database

Payment Service

Browse Songs

List of Songs

Buy Song

Charge Customer

alt [payment success]

ok

Play Song

Get Music

[payment fail]

refused

show payment failed error

User Interface

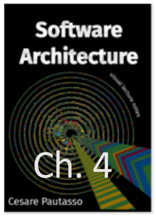
Music Player

Songs Repository

Customer Database

Payment Service

www.plantuml.com



Software Architecture

Ch. 4

Cesare Pautasso

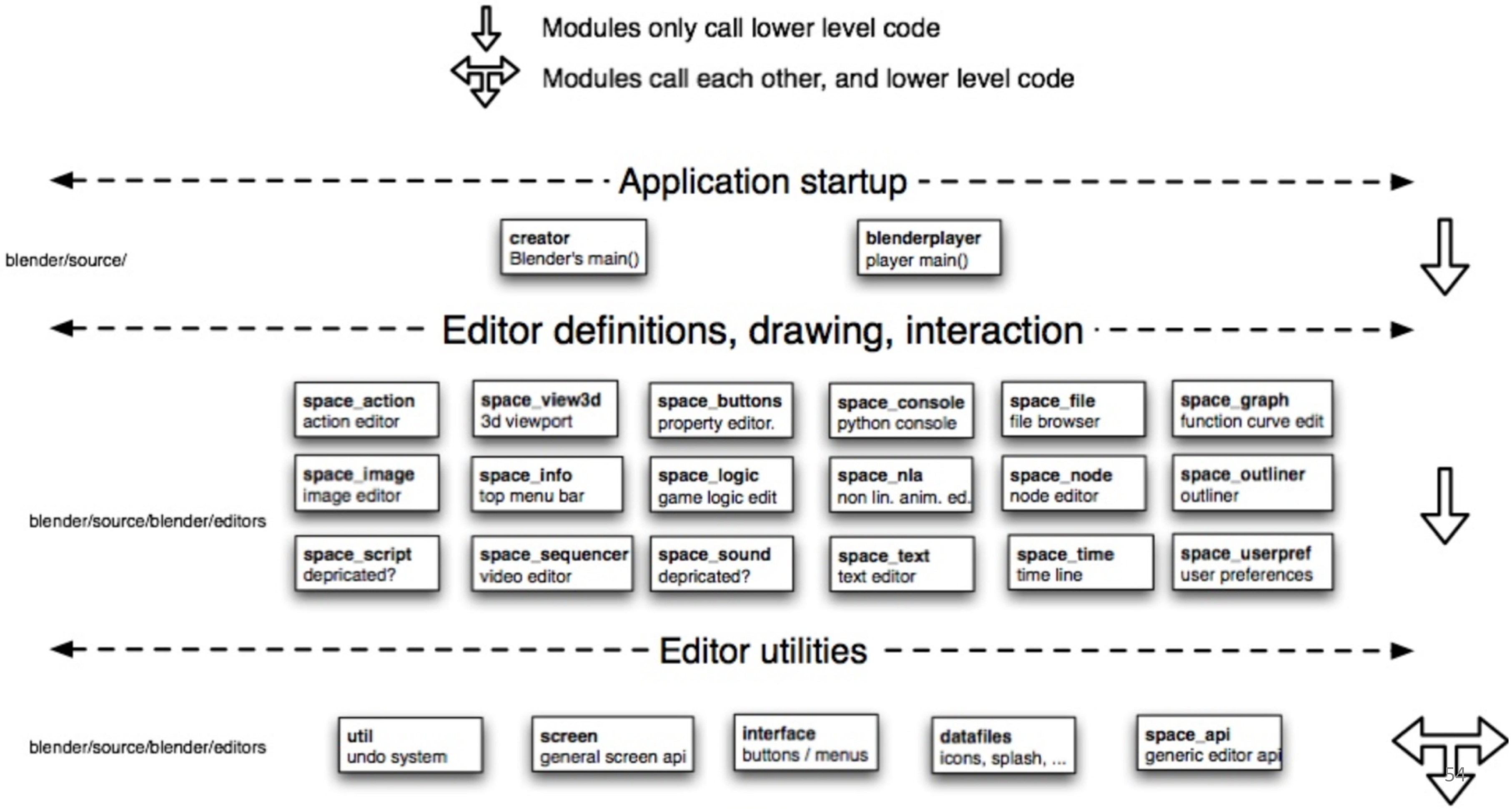
Kruchten’s “Development View”

- Static organization of the software code artifacts (packages, modules, binaries...)
- Map logical view onto code
- Describe code review, contribution, and build process

- Concern: Reuse, Portability, Build
- Target Audience: Developers

First line of thinking for “us, developers”

Blender code layout



Modules only call lower level code

Modules call each other, and lower level code

Application startup

blender/source/

creator
Blender's main()

blenderplayer
player main()

Editor definitions, drawing, interaction

blender/source/blender/editors

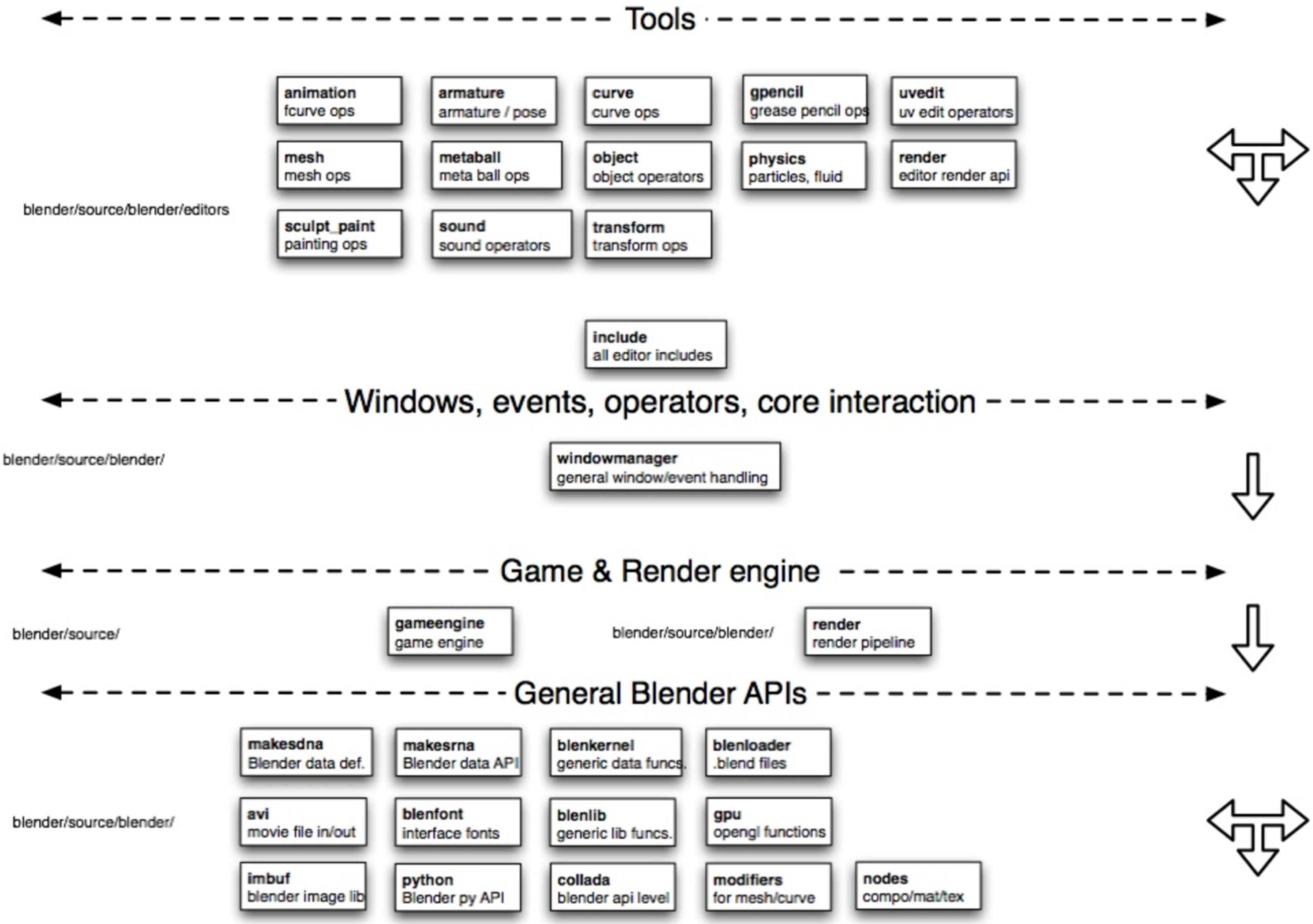
space_action action editor	**space_view3d** 3d viewport	**space_buttons** property editor.	**space_console** python console	**space_file** file browser	**space_graph** function curve edit
space_image image editor	**space_info** top menu bar	**space_logic** game logic edit	**space_nla** non lin. anim. ed.	**space_node** node editor	**space_outliner** outliner
space_script depricated?	**space_sequencer** video editor	**space_sound** depricated?	**space_text** text editor	**space_time** time line	**space_userpref** user preferences

Editor utilities

blender/source/blender/editors

util undo system	**screen** general screen api	**interface** buttons / menus	**datafiles** icons, splash, ...	**space_api** generic editor api

https://www.blender.org/bf/codelayout.jpg



Tools

blender/source/blender/editors

animation fcurve ops

armature armature / pose

curve curve ops

gpencil grease pencil ops

uvedit uv edit operators

mesh mesh ops

metaball meta ball ops

object object operators

physics particles, fluid

render editor render api

sculpt_paint painting ops

sound sound operators

transform transform ops

include all editor includes

Windows, events, operators, core interaction

blender/source/blender/

windowmanager general window/event handling

Game & Render engine

blender/source/

gameengine game engine

blender/source/blender/

render render pipeline

General Blender APIs

blender/source/blender/

makesdna Blender data def.

makesrna Blender data API

blenkernel generic data funcs.

blenloader .blend files

avi movie file in/out

blenfont interface fonts

blenlib generic lib funcs.

gpu opengl functions

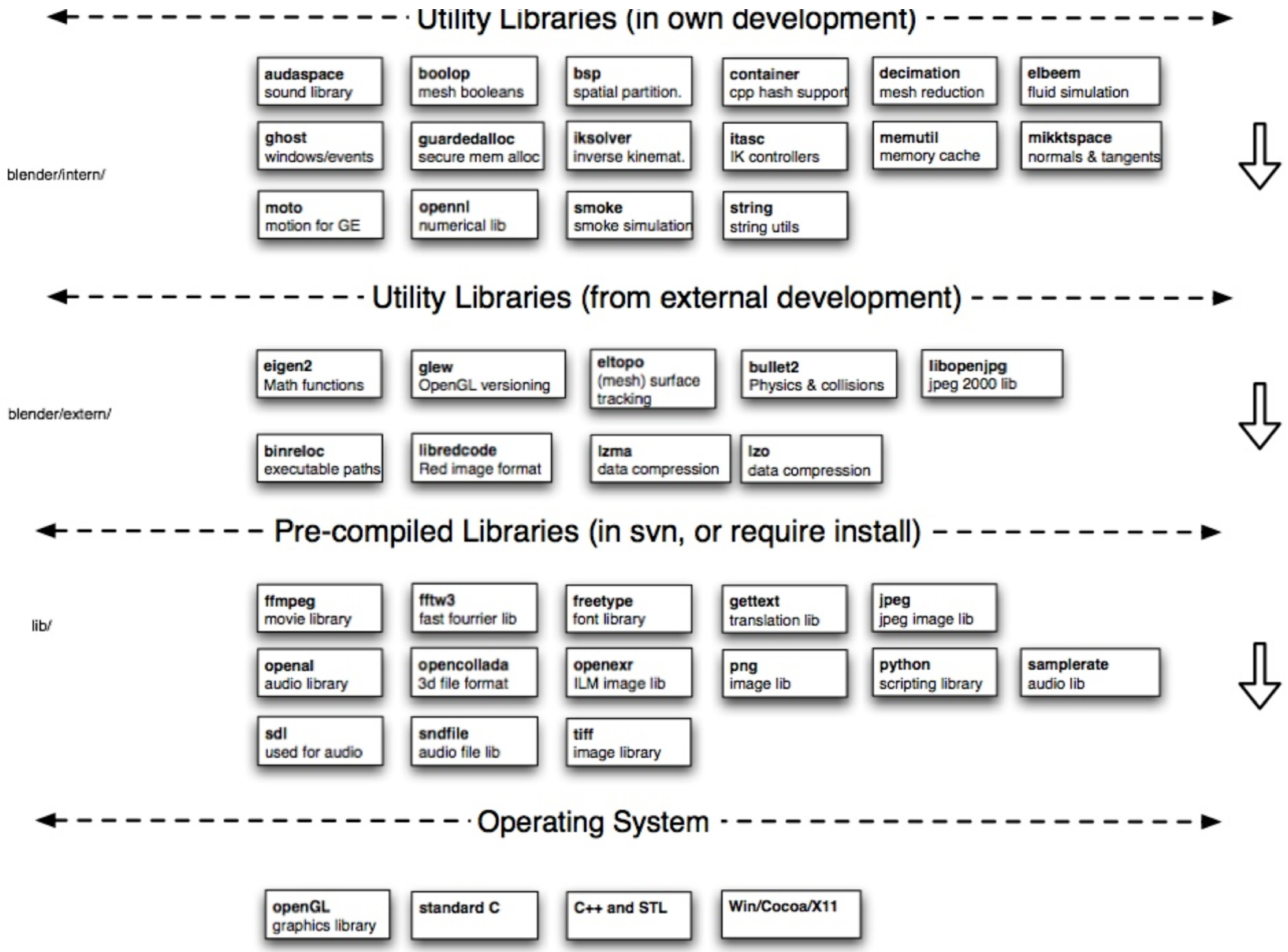
imbuf blender image lib

python Blender py API

collada blender api level

modifiers for mesh/curve

nodes compo/mat/tex



Utility Libraries (in own development)

blender/intern/

audaspace
sound library

boolop
mesh booleans

bsp
spatial partition.

container
cpp hash support

decimation
mesh reduction

elbeem
fluid simulation

ghost
windows/events

guardedalloc
secure mem alloc

iksolver
inverse kinemat.

itasc
IK controllers

memutil
memory cache

mikktspace
normals & tangents

moto
motion for GE

opennl
numerical lib

smoke
smoke simulation

string
string utils

Utility Libraries (from external development)

blender/extern/

eigen2
Math functions

glew
OpenGL versioning

eltopo
(mesh) surface tracking

bullet2
Physics & collisions

libopenjpg
jpeg 2000 lib

binreloc
executable paths

libredcode
Red image format

lzma
data compression

lzo
data compression

Pre-compiled Libraries (in svn, or require install)

lib/

ffmpeg
movie library

fftw3
fast fourrier lib

freetype
font library

gettext
translation lib

jpeg
jpeg image lib

openal
audio library

opencollada
3d file format

openexr
ILM image lib

png
image lib

python
scripting library

samplerate
audio lib

sdl
used for audio

sndfile
audio file lib

tiff
image library

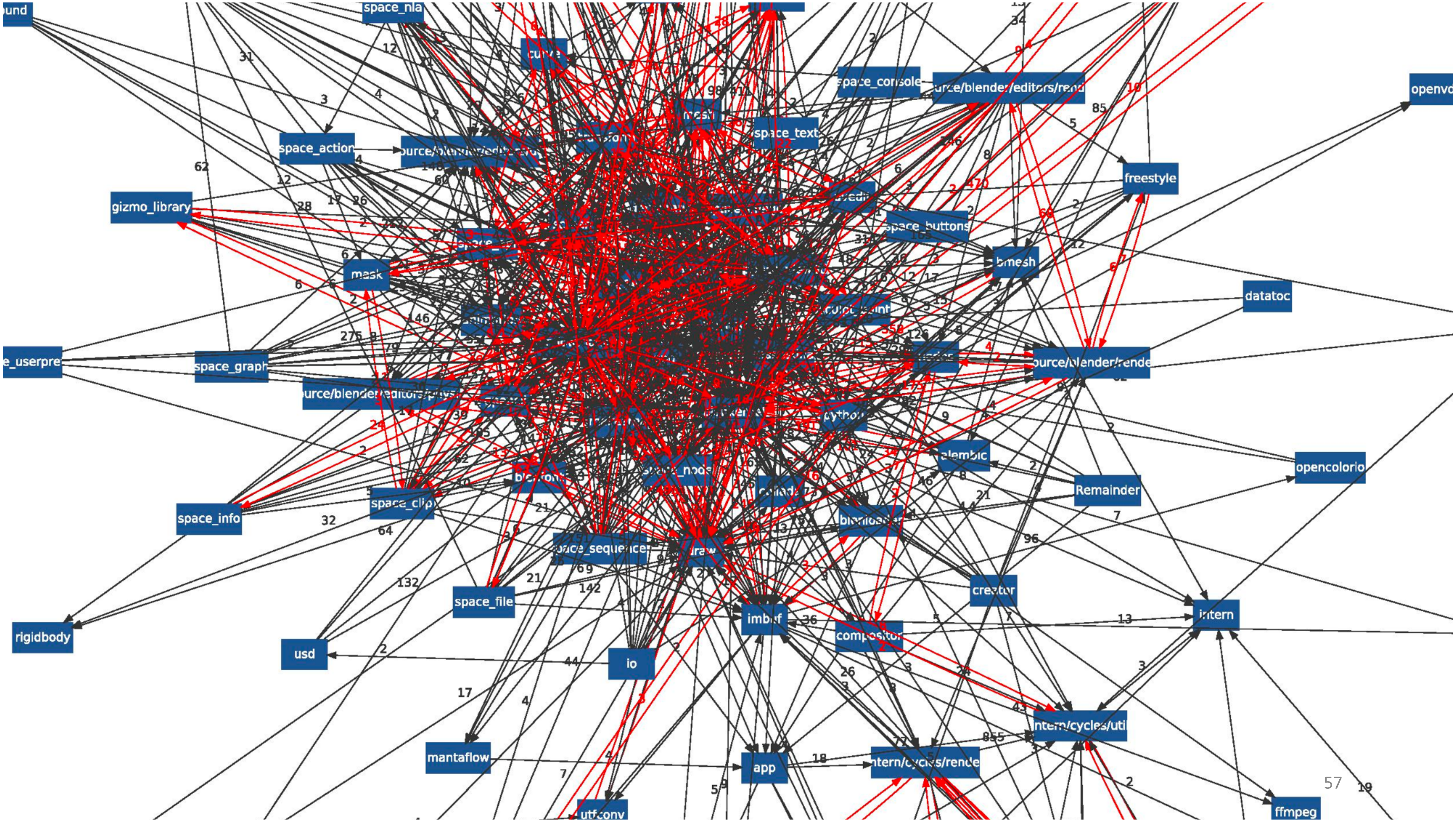
Operating System

openGL
graphics library

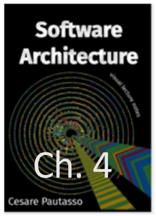
standard C

C++ and STL

Win/Cocoa/X11


ound
space_nla
curve
space_console
urce/blender/editors/rend
openvo
space_action
ource/blender/editors/
mesh
space_text
freestyle
gizmo_library
space_buttons
bmesh
mask
datatoc
e_userpre
space_graph
ource/blender/rende
ource/blender/editors/phys
python
alembic
opencolorio
Remainder
space_info
space_clip
space_node
collada
blenloader
space_sequencer
draw
creator
space_file
imbuf
compositor
intern
rigidbody
usd
io
intern/cycles/util
mantaflow
app
intern/cycles/rende
ffmpeg
utfconv

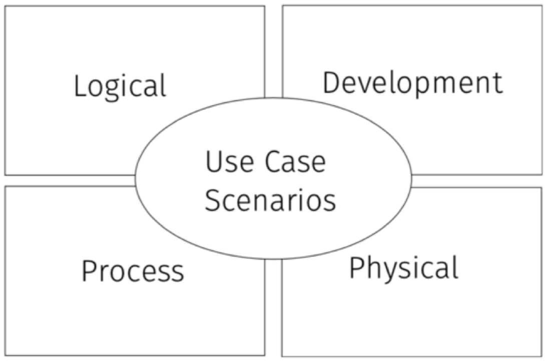
Kruchten’s “Physical View”

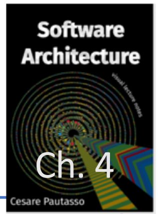


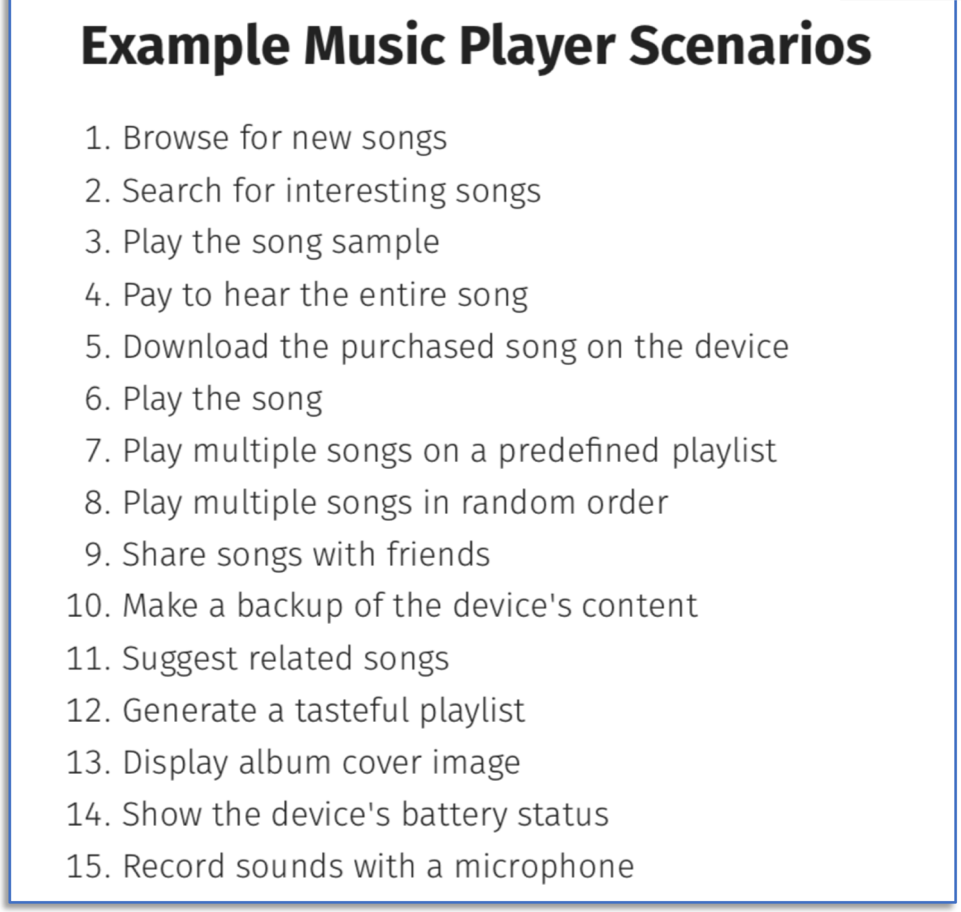
- Define the hardware environment (hosts, networks, storage, etc.) where the software will be deployed
- Different hardware configurations for providing different qualities
- **Deployment View**: Mapping between logical and physical entities
- Virtual is the new physical
  - Amazon’s “AWS Well-Architected Framework”
- Concern: Performance, Scalability, Availability, Reliability, Security
- Target Audience: Operations

4+1: Connecting Kruchten’s Views with Use Cases

- Views should not contradict each other
- Use cases can be “executed” in each view



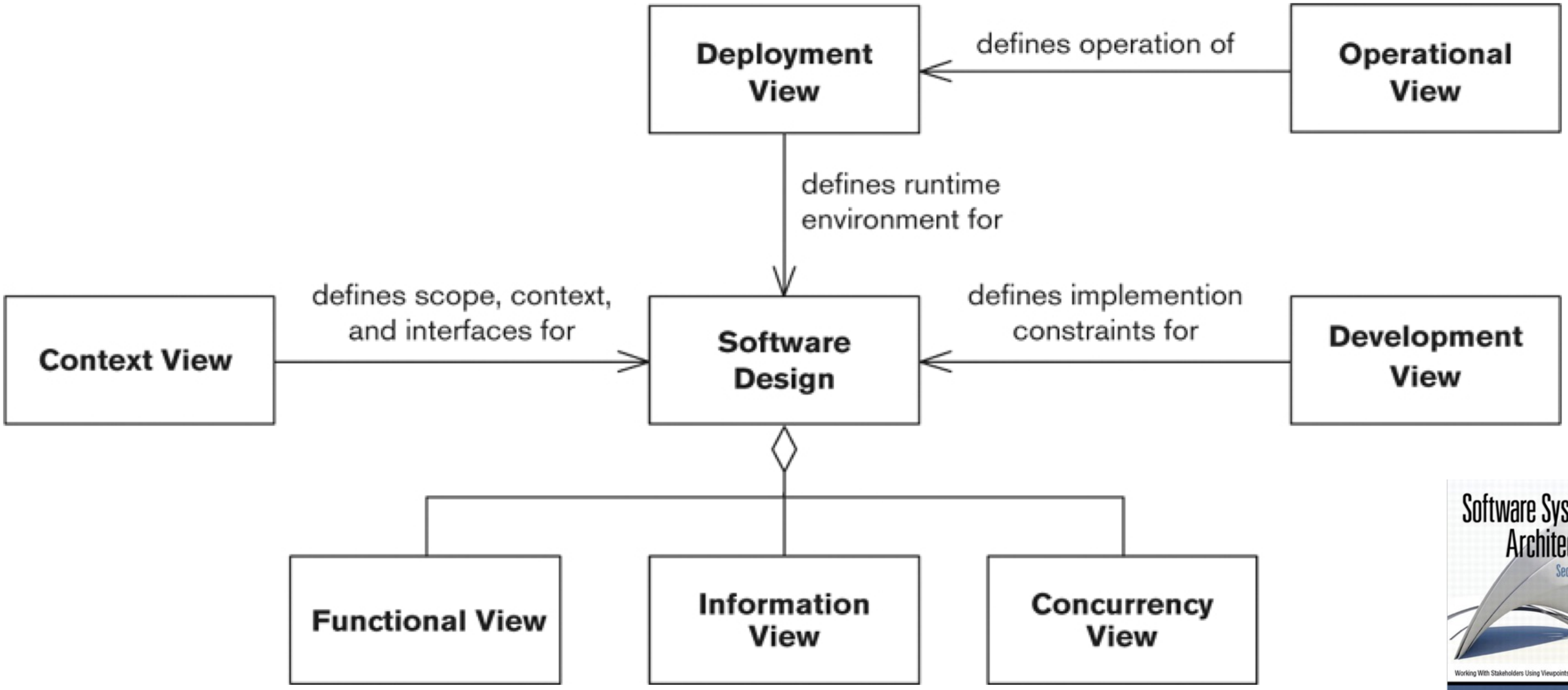


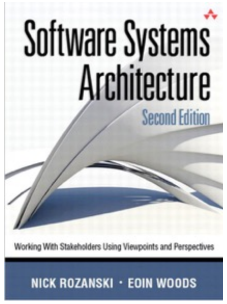


Example Music Player Scenarios

1. Browse for new songs
2. Search for interesting songs
3. Play the song sample
4. Pay to hear the entire song
5. Download the purchased song on the device
6. Play the song
7. Play multiple songs on a predefined playlist
8. Play multiple songs in random order
9. Share songs with friends
10. Make a backup of the device's content
11. Suggest related songs
12. Generate a tasteful playlist
13. Display album cover image
14. Show the device's battery status
15. Record sounds with a microphone

Rozanski & Woods Viewpoint Taxonomy

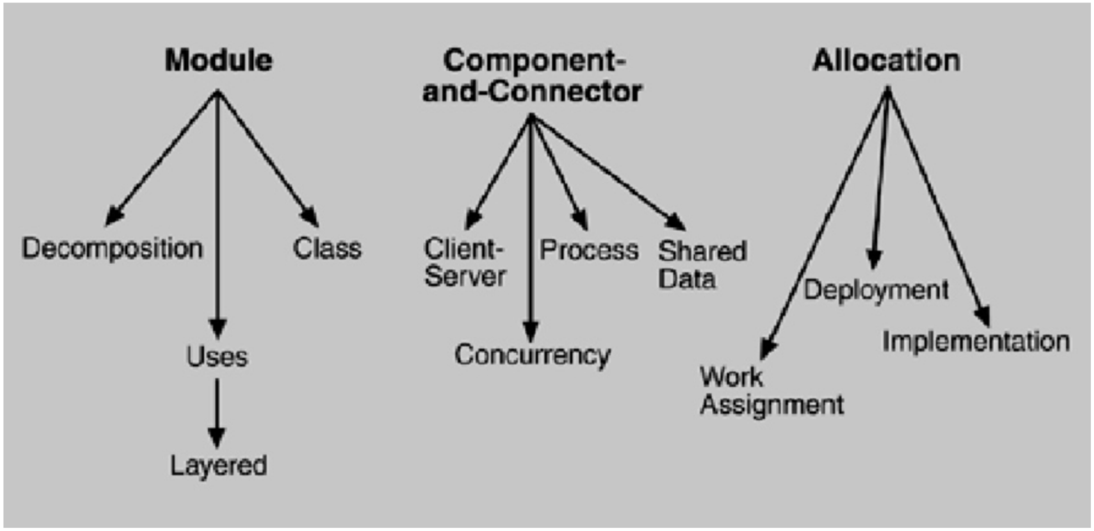


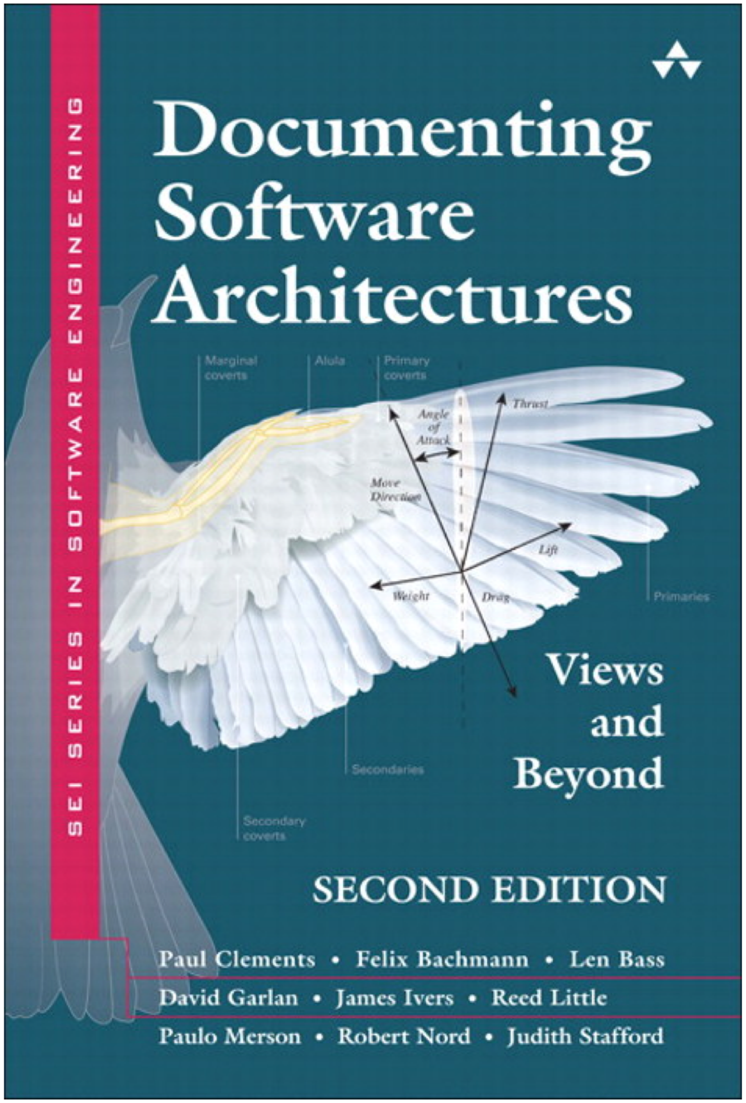

Software Systems Architecture
Second Edition
Working With Stakeholders Using Viewpoints and Perspectives
NICK ROZANSKI · EOIN WOODS

"SEI DSA" Taxonomy

"View types":

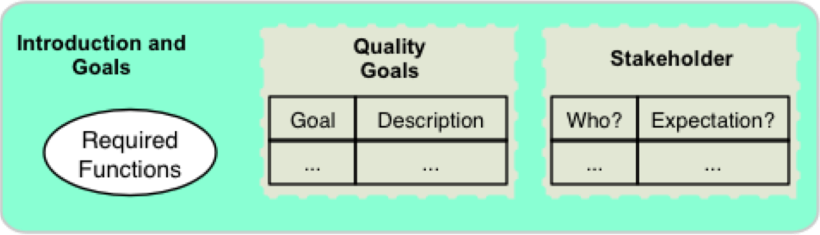
- Module
- Component & Connector
- Allocation




SEI SERIES IN SOFTWARE ENGINEERING
Documenting Software Architectures
Views and Beyond
SECOND EDITION
Paul Clements • Felix Bachmann • Len Bass
David Garlan • James Ivers • Reed Little
Paulo Merson • Robert Nord • Judith Stafford

arc42.org: A Template for Architecture Communication and Documentation

1. Introduction and Goals



Introduction and Goals

Required Functions

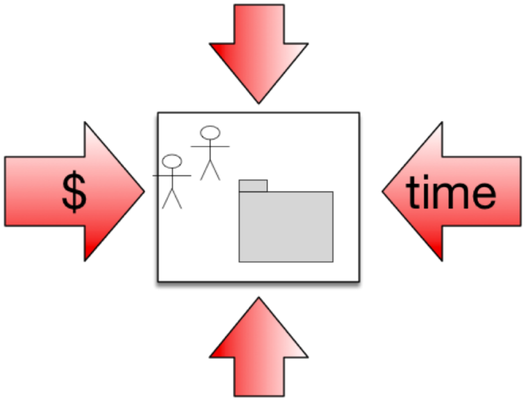
Quality Goals

Goal	Description
...	...

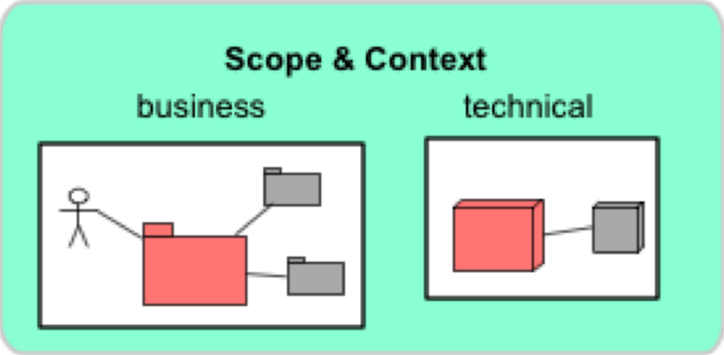
Stakeholder

Who?	Expectation?
...	...

2. Constraints



3. Context and Scope



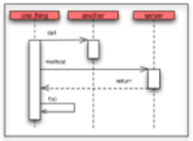
Scope & Context

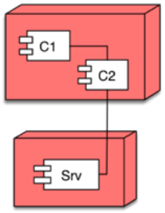
business

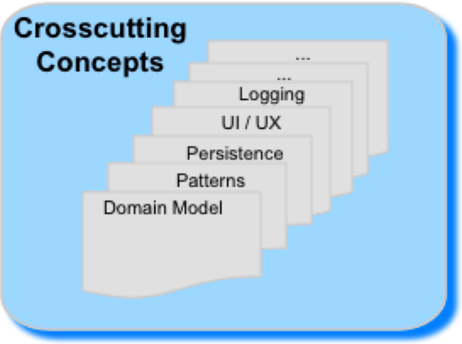
technical

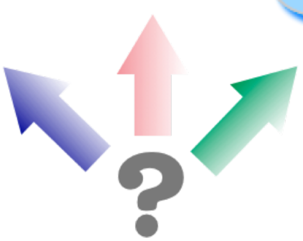
The arc42.org Template for Architecture Communication and Documentation

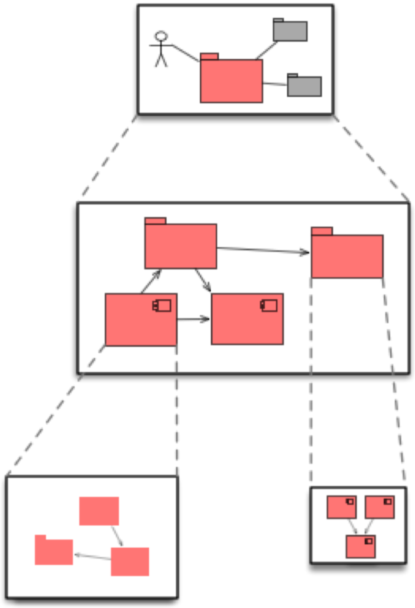
4. Solution strategy
5. Building block view
6. Run time view
7. Deployment view
8. Crosscutting concepts
9. Architectural decisions





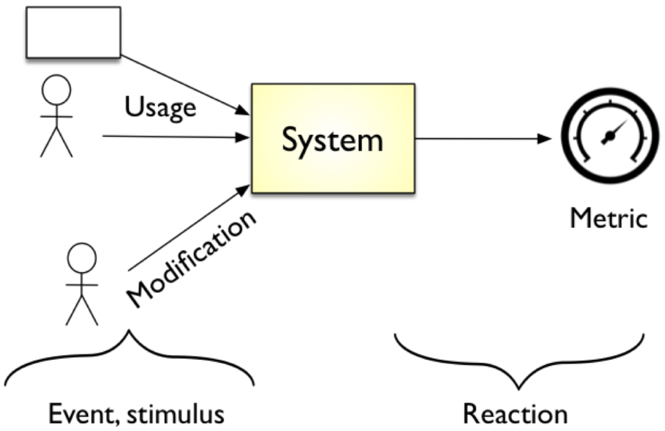




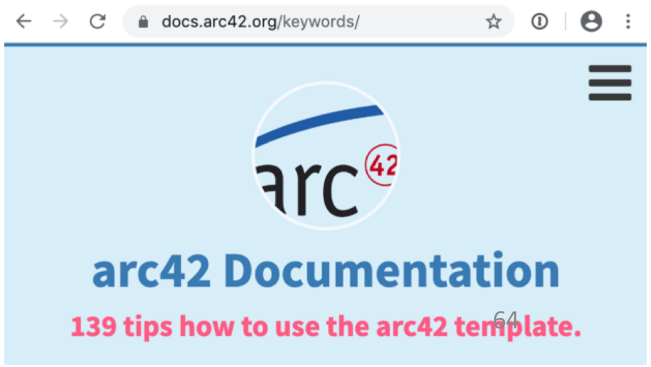


The arc42.org Template for Architecture Communication and Documentation

10. Quality Requirements



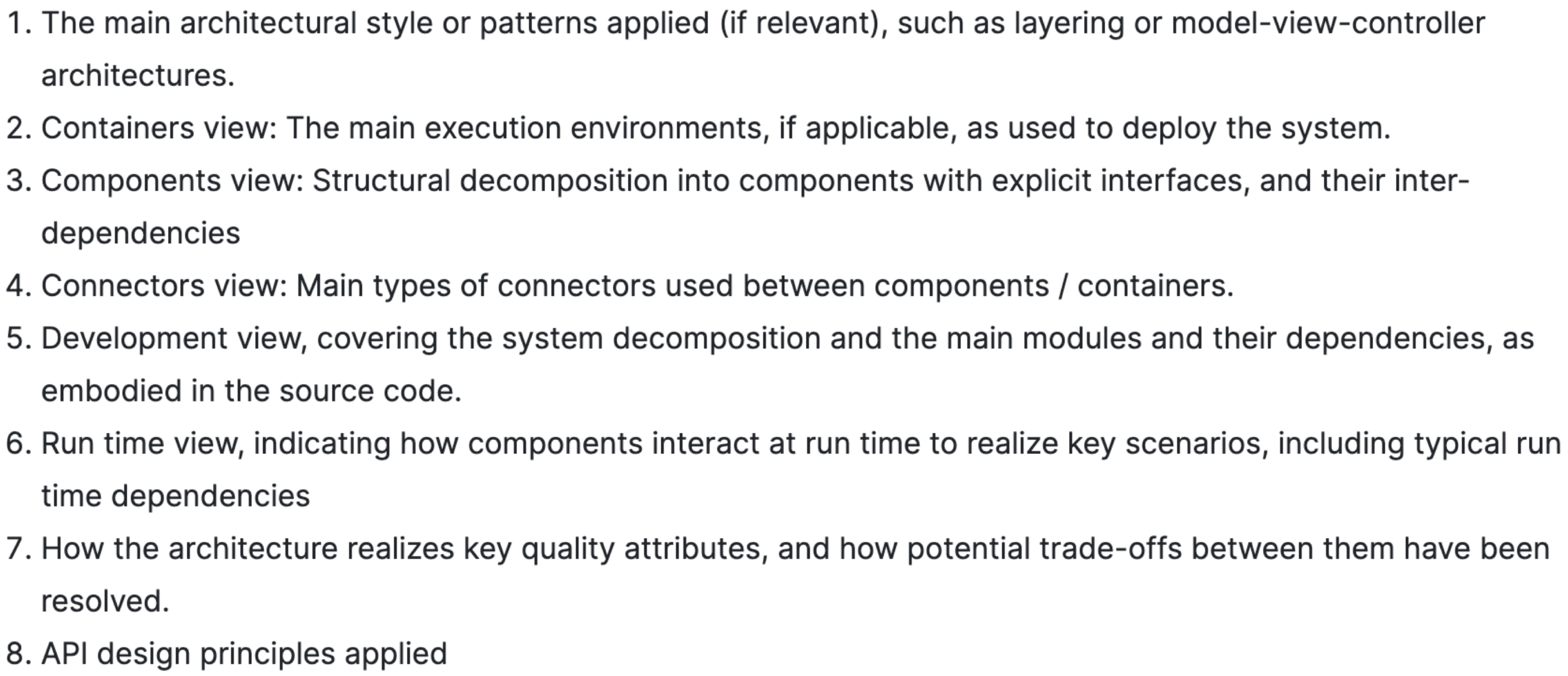
11. Risks and Technical Debt



arc42 Documentation

139 tips how to use the arc42 template.

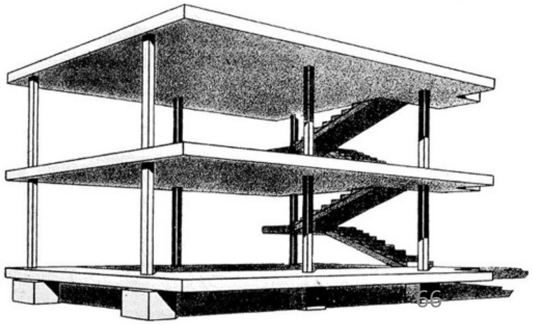
Essay 2: The System’s Architecture



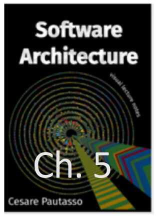
1. The main architectural style or patterns applied (if relevant), such as layering or model-view-controller architectures.
2. Containers view: The main execution environments, if applicable, as used to deploy the system.
3. Components view: Structural decomposition into components with explicit interfaces, 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

Software Architecture: Modularization and Interface Design

Arie van Deursen



Wikipedia, Dom-Ino House, Corbusier



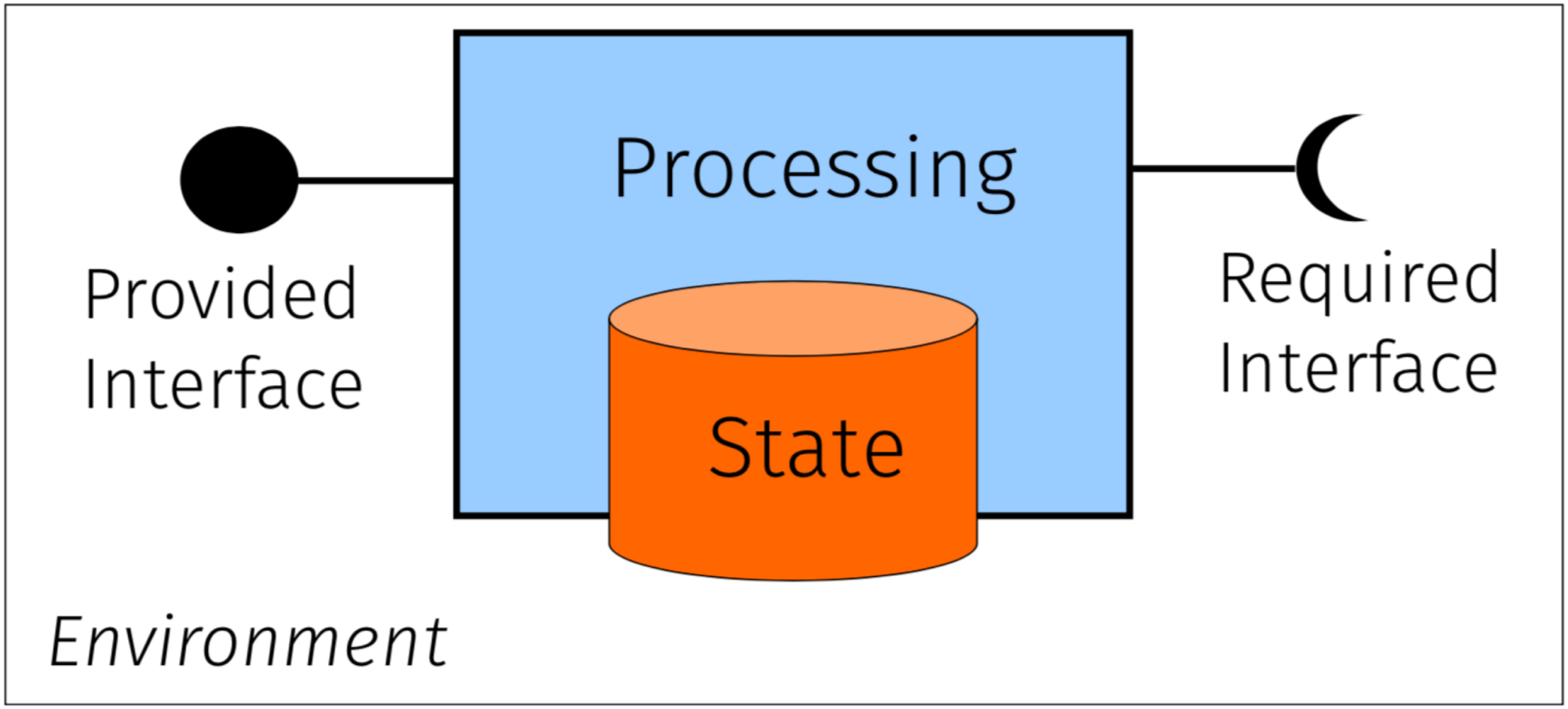
Software Architecture

Ch. 5

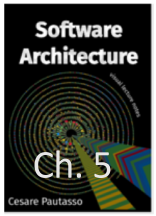
Cesare Pautasso

Software Component

- Locus of computation and state in a system

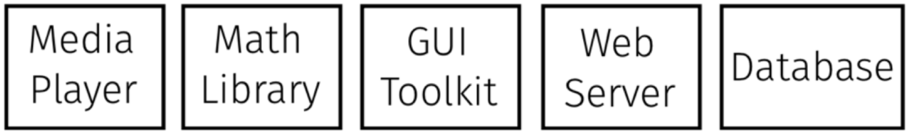


Sorts of Components



Infrastructure

- Address needs of multiple application domains
- Highly reusable
- Customizable
- Support non-functionals


Media Player

Math Library

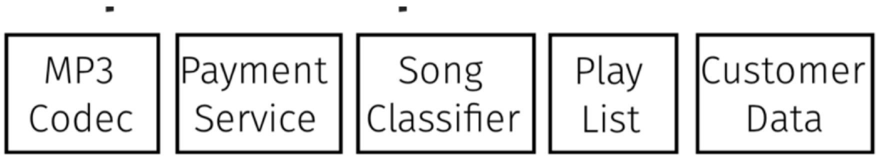
GUI Toolkit

Web Server

Database

Application-specific

- Directly implement main functionality
- Domain knowledge intensive
- Less suitable for reuse

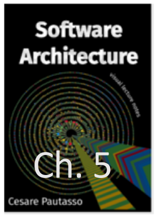

MP3 Codec

Payment Service

Song Classifier

Play List

Customer Data



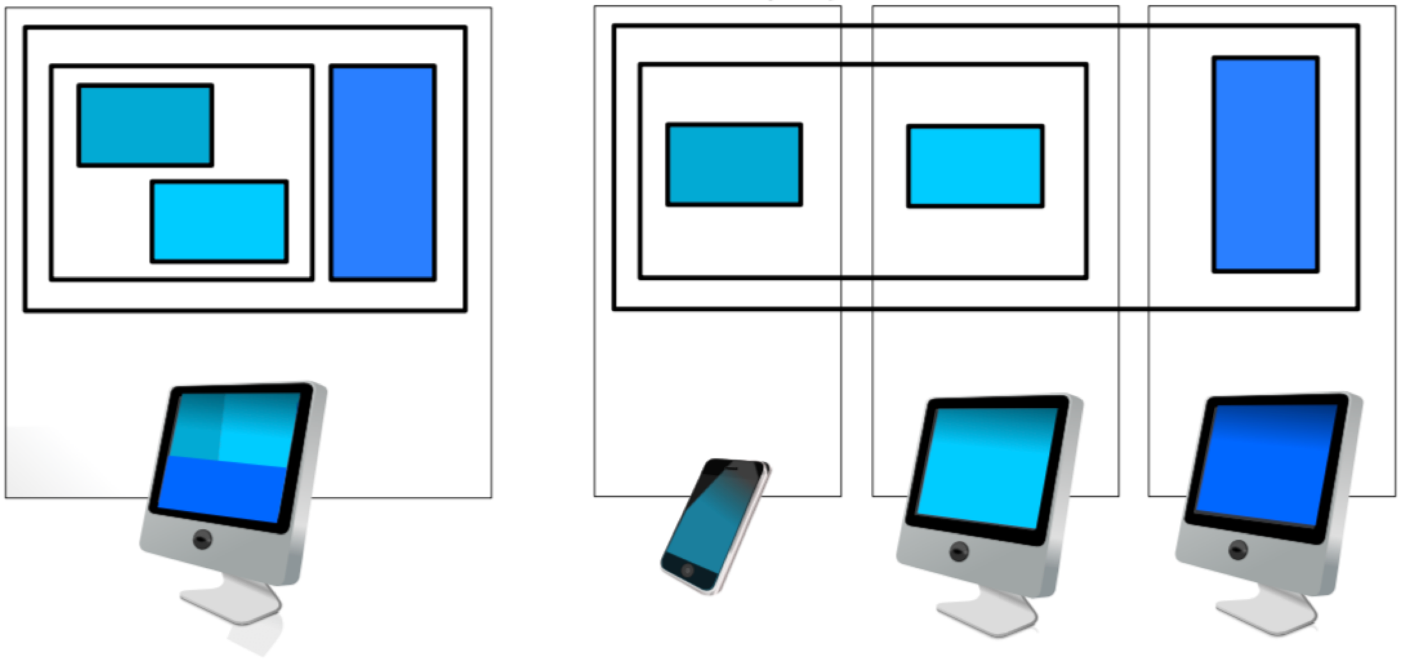
Software Architecture

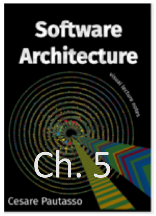
Ch. 5

Cesare Pautasso

Distributed Components

- Components can be deployed on the same physical host
- Components can be distributed over multiple physical hosts



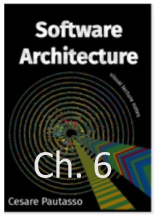


Software Architecture

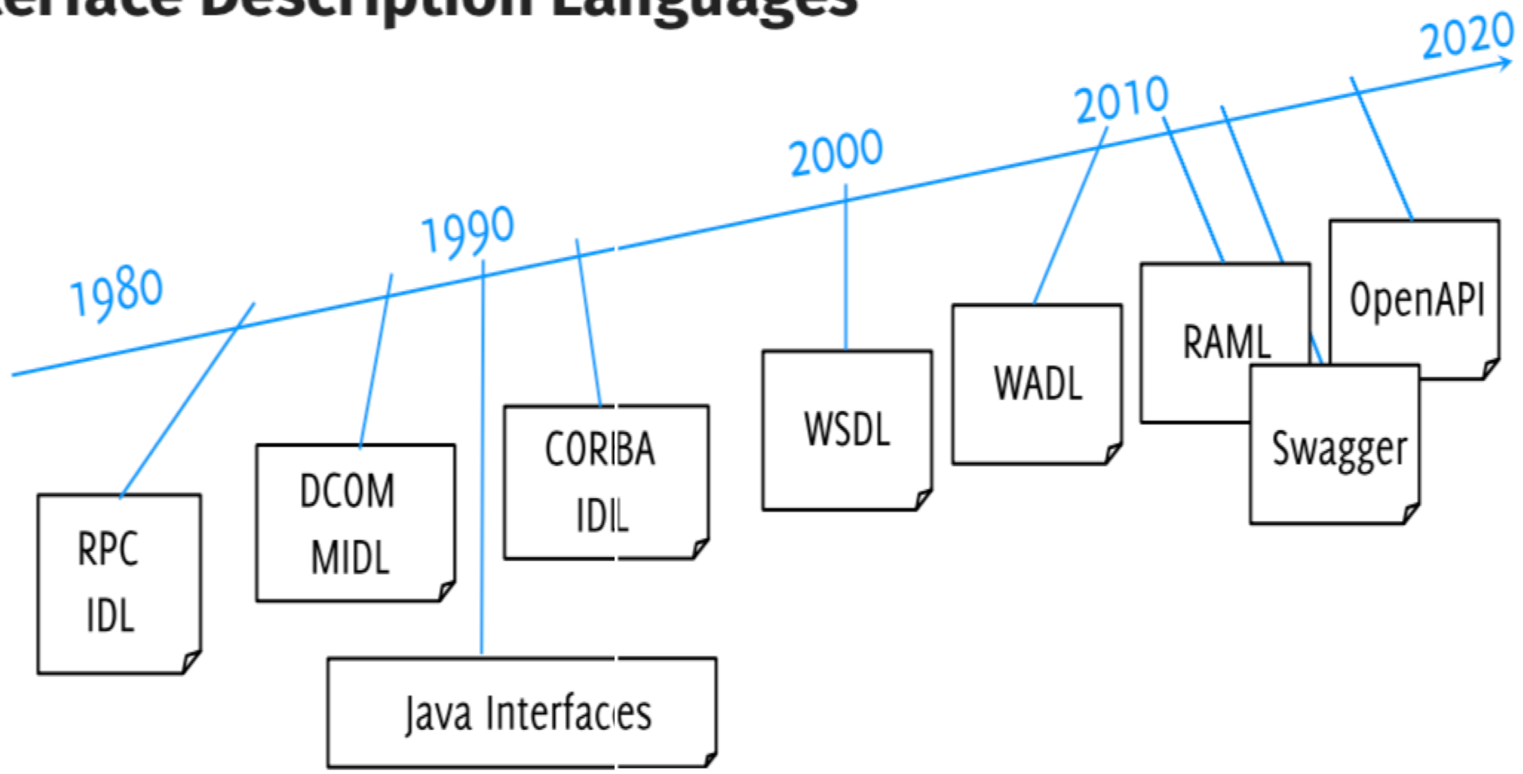
Ch. 5

Cesare Pautasso

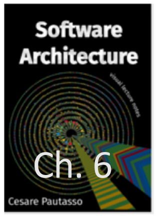
	Components	Objects
Abstraction	Architecture	Code
Encapsulation	State and Functionality	State and Functionality
Granularity	Coarse-grained	Fine-grained
Modularity	Unit of Composition and Deployment	Identifiable Unit of Instantiation
Interface	Well-defined, documented	Optional
Reusability	Explicit dependencies (can be self-contained)	Entangled with other objects (hard to reuse by itself)



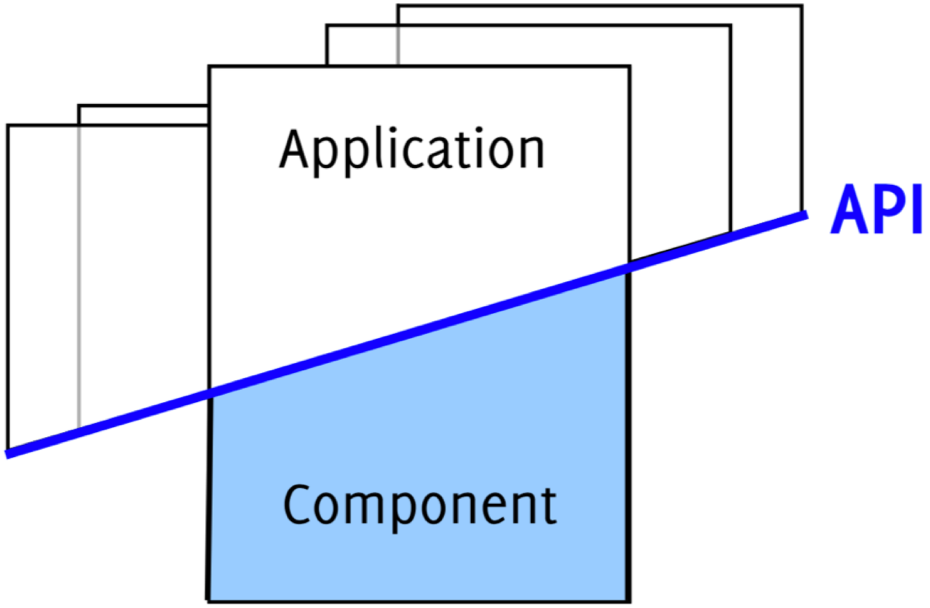
Interface Description Languages



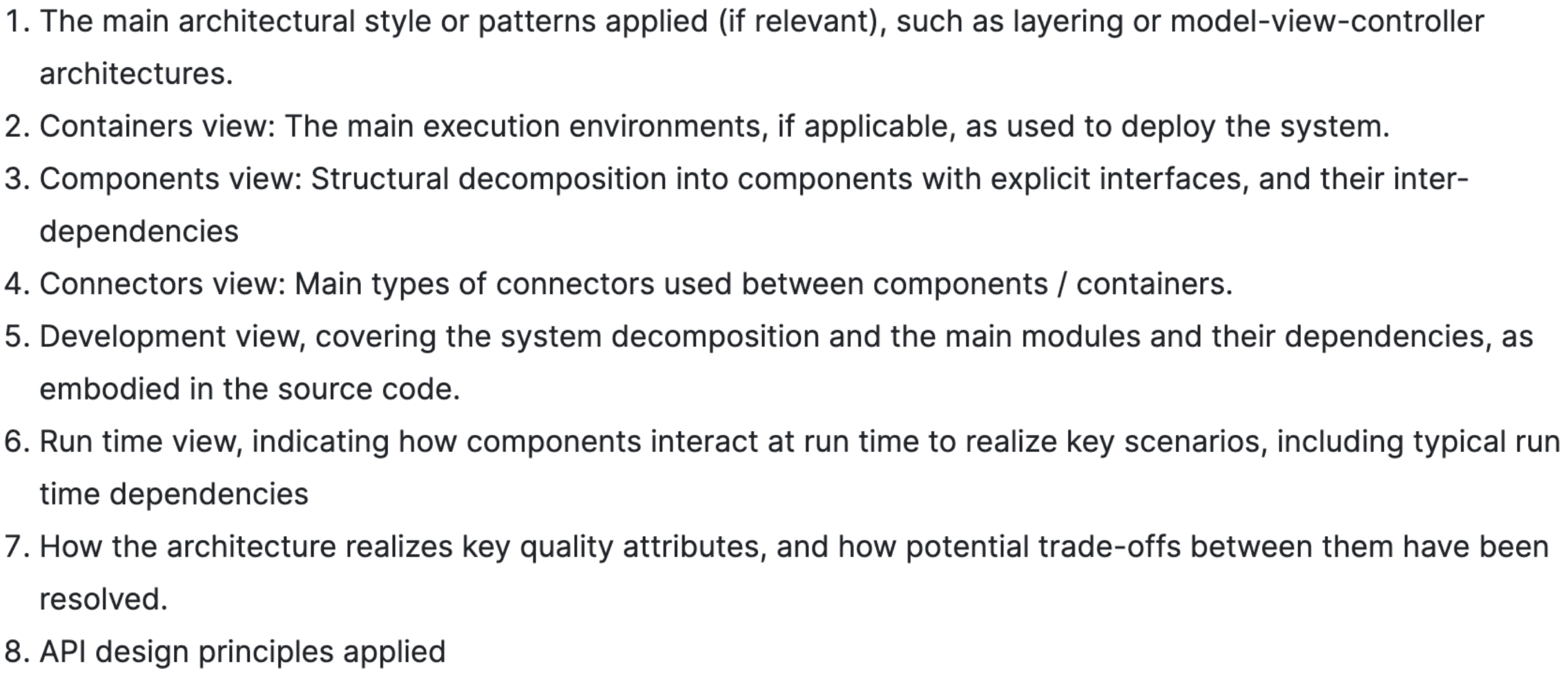
Application Programming Interfaces



- APIs are not found in all architectures:
- APIs can be found in architectures that are designed to be
  - open and stable platforms
  - supporting externally developed components and applications.



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