

IN4315 Lecture 5: Views and Beyond

Arie van Deursen

Date	Start	End	Activity	Teacher	Topic	Slides	Video
Wed Feb 9	13:45	15:30	Lecture 1	Arie van Deursen	Introduction and Course Structure	pdf	video
Fri Feb 11	08:45	10:30	Lecture 2	Arie van Deursen	Envisioning the System (E1, E2)	pdf	video
Wed Feb 16	13:45	15:30	Lecture 3	Diomidis Spinellis	Architecting for Quality (E3)	pdf	video
Fri Feb 18	08:45	10:30	Lecture 4	Diomidis Spinellis	Architecting for Scale (E4)	pdf	video
Wed Feb 23	13:45	15:30	Lecture 5	Arie van Deursen	Views and Beyond (E2 cont.)		
Fri Feb 25	08:45	10:30	Lecture 6	Arie van Deursen	Architecting for Configurability		
Wed Mar 2	13:45	15:30	Lecture 7	Diomidis Spinellis	50 years of Unix Architecture Evolution		
Fri Mar 4	08:45	10:30	Lecture 8	TBD	TBD		
Wed Mar 9	13:45	14:30	Lecture 9	Mattermost (tentative)	The Team / open source / AMA		
	14:45	15:30	Lecture 9	Uber (tentative)	Architecting for Privacy / AMA		
Fri Mar 11	08:45	10:30	Lecture 10	Lukas Vermeer , Kevin Anderson	Architecting for Experimentation		
Wed Mar 16	13:45	15:30	Lecture 11	Maurício , Efe , Thinus , Arthur	Architecture at Adyen		
Fri Mar 18	08:45	10:30	Lecture 12	Pinar Kahraman (ING)	AI Ops and Analytics		
Wed Mar 23	13:45	15:30	Lecture 13	TBD	TBD		
Fri Mar 25	08:45	10:30	Lecture 14	TBD	TBD		
Wed Mar 30	08:45	17:30	Finale	All students	Final presentations		

Coaching

- Coaching: Nine coaches available
- Briefing of coaches today
- Ready for coach meetings from Thu / Fri onwards
- Coaches are available for 2-3 meetings in weeks 3-7
- TA Erik Sennema will coordinate assignment to teams
- Prepare meetings well:
 - agenda, round of introductions, presentation with status update, questions you have, options to explore, contributions, ...
- Primary objective: *help* you (not grade you)



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



Meta	Stakeholders	
	Internal	External
Observability Measurability Repeatability Predictability Auditability Accountability Testability		Functionality Correctness Completeness Compliance Ethics
Design	Feasibility Time to Market Affordability Consistency Simplicity Clarity Stability Modularity Reusability Composability	Aesthetics Deployability
Operation	Manageability	Usability Accessibility Ease of support Serviceability Performance Scalability
Failure	Visibility	Dependability Safety Recoverability Reliability Availability Security Confidentiality Integrity Authentication Authorization Non-Repudiation
Attack	Defensibility	Survivability Privacy
Change	Modifiability Elasticity	Flexibility Configurability Customizability Resilience Adaptability Extensibility Compatibility Portability Ease of Integration Interoperability
Long-term	Maintainability Explainability	Evolvability Durability Disposability Understandability Sustainability

Robust



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

Resilient



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

Antifragile



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

Nassim Taleb: Antifragile

*Some things benefit from shocks;
they thrive and grow when exposed to
volatility, randomness, disorder, and stressors
and love adventure, risk, and uncertainty.*

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

Let us call it antifragile.

Antifragility is beyond resilience or robustness.

*The resilient resists shocks and stays the same;
the antifragile gets better*



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

What's a Good Essay?

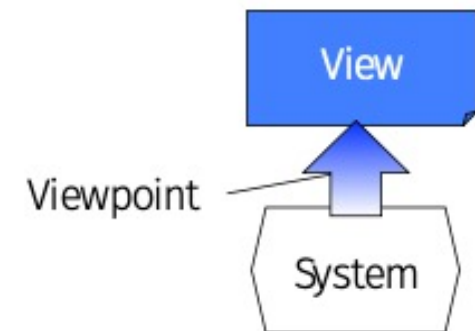
1. The text is well-structured, with a clear goal, a natural breakdown in sections, and a compelling conclusion.
2. Sentences, paragraphs, and sections are coherent. They naturally build upon each other and work towards a clear message.
3. The arguments laid out are technically sound, and of adequate technical depth.
4. The English writing is grammatically correct
5. A standard notation, such as [UML 2](#), is appropriately used for all diagrams
6. The text clearly references any sources it builds upon
7. The essay is unique and recognizable in its voice and its way of approaching the topic
8. The essay is independently readable
9. The story-line is illustrated with meaningful and appealing images and infographics.

Some Essay Advice

- Keep your audience in mind:
 - “computer science students or software engineers, interested in learning about architectural aspects of your open source project.”
- Be courageous – dare to deviate
- Let the system be leading, not the fulfillment of an assignment
- Invest time and let it show – dig as deep as you can

Using “Architectural Views” to Organize Architectural Models

- 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)

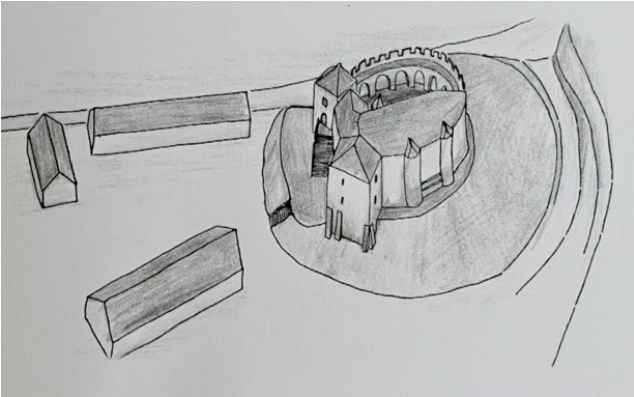


Views on
Kessel
Castle
Keerbergen

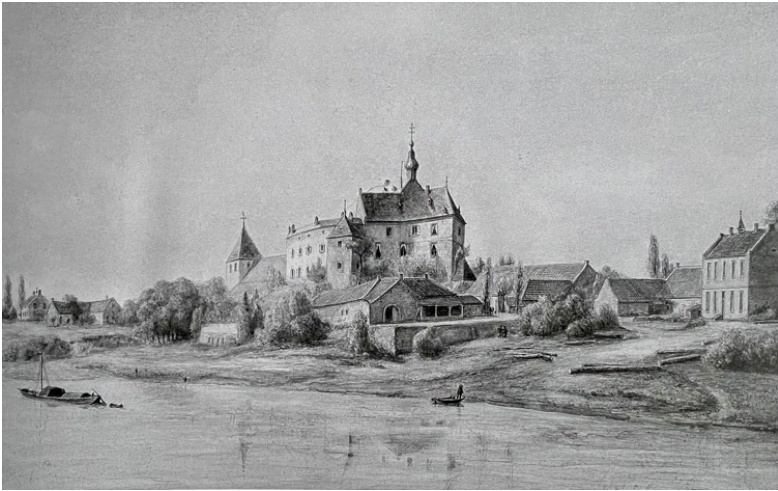


The legacy view

1400
(motte)



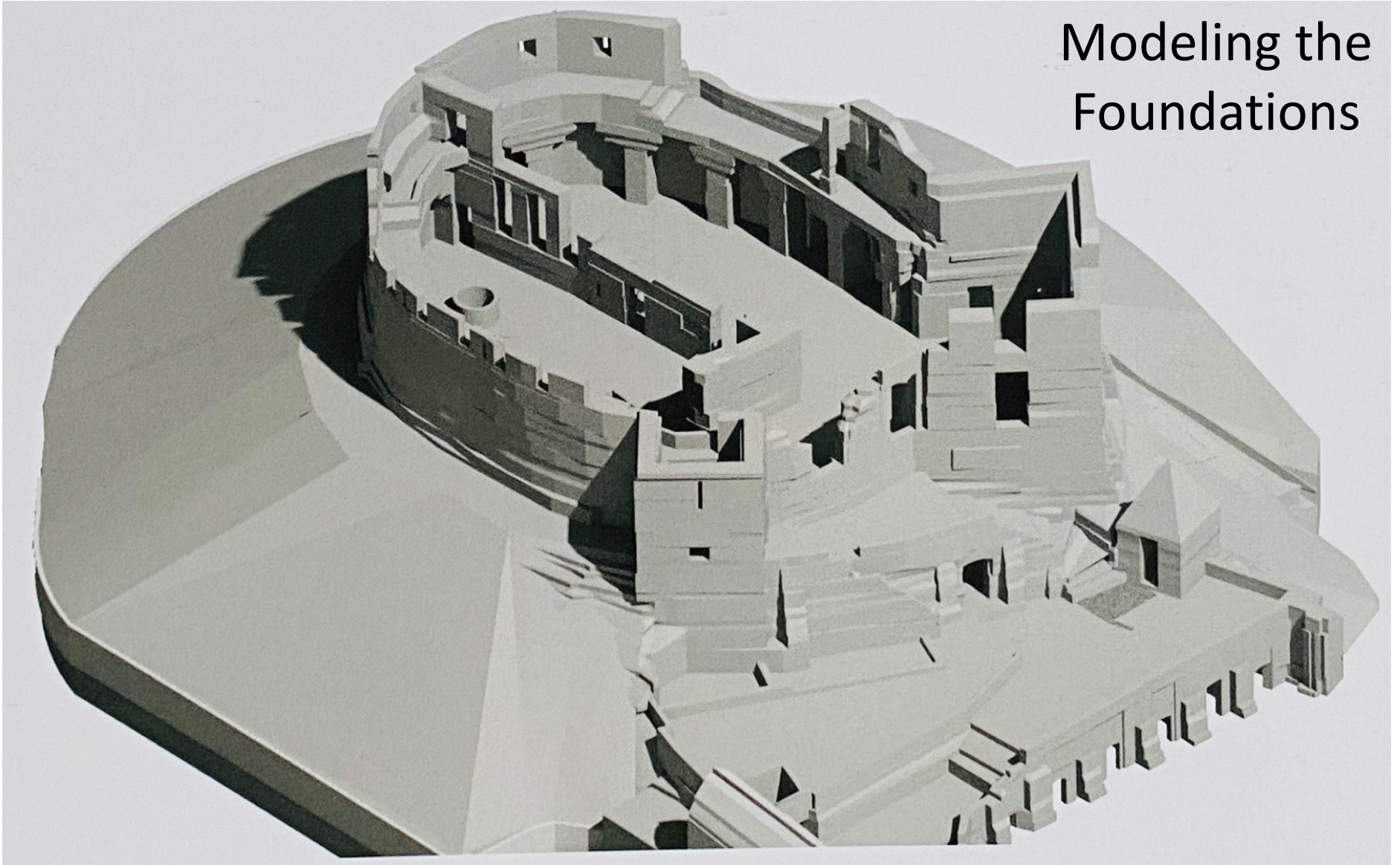
1850

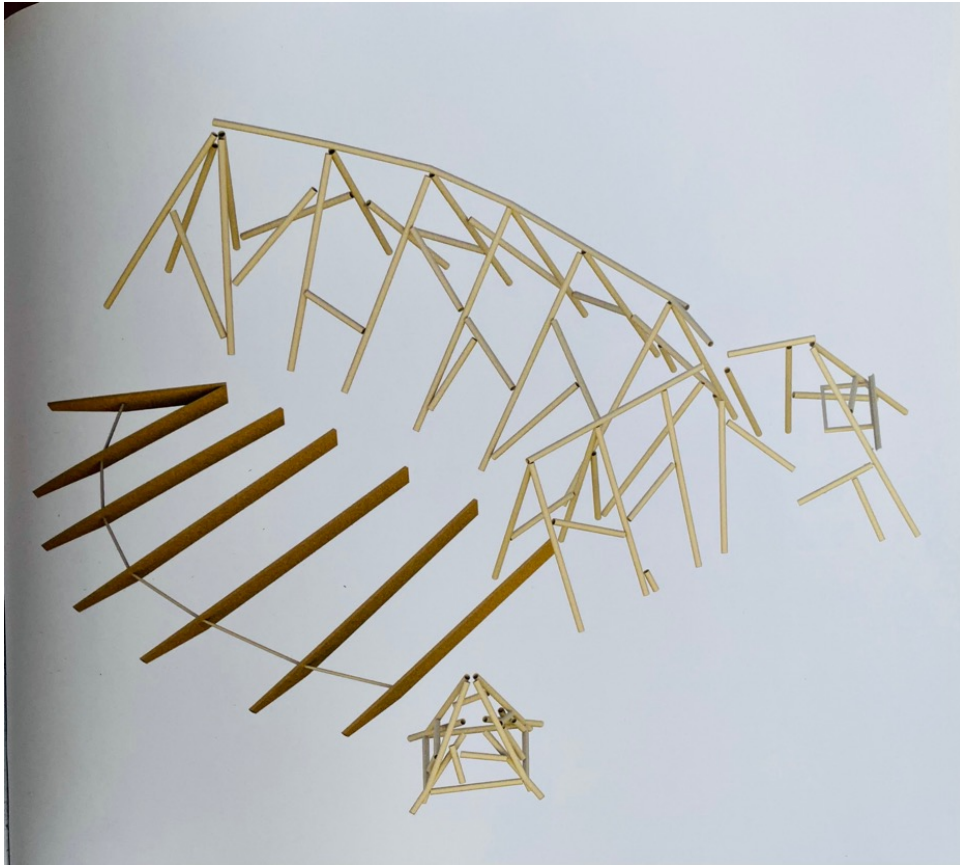


1944



Modeling the Foundations



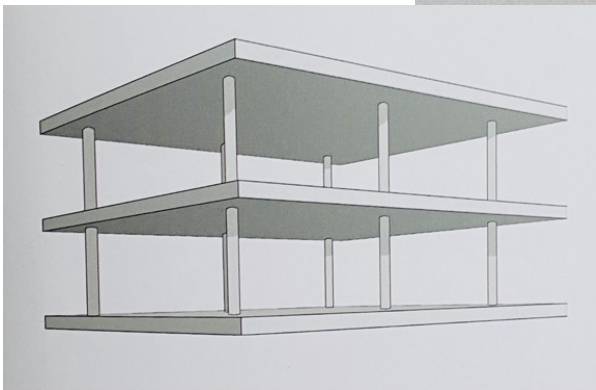
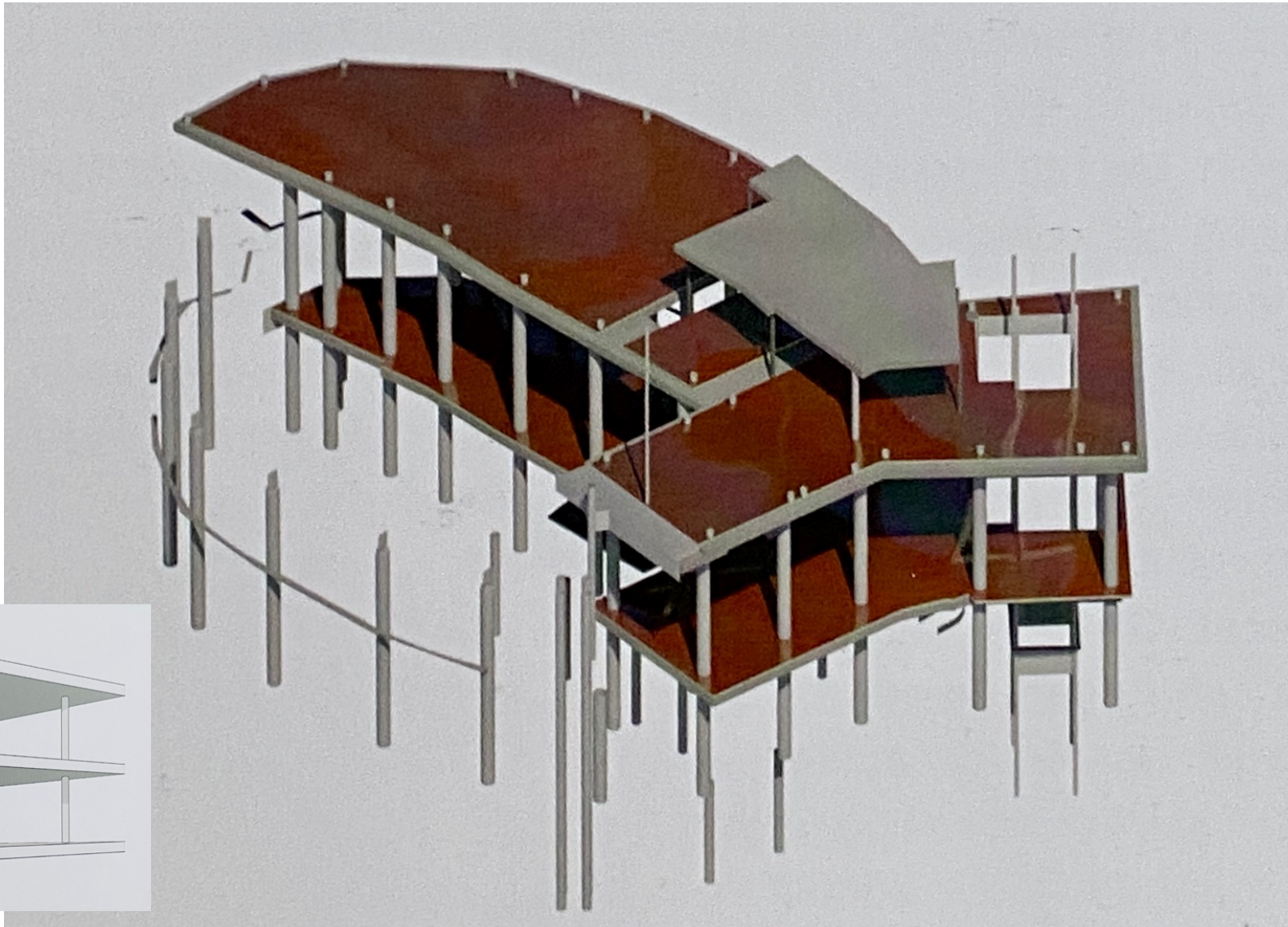


A view on the roof

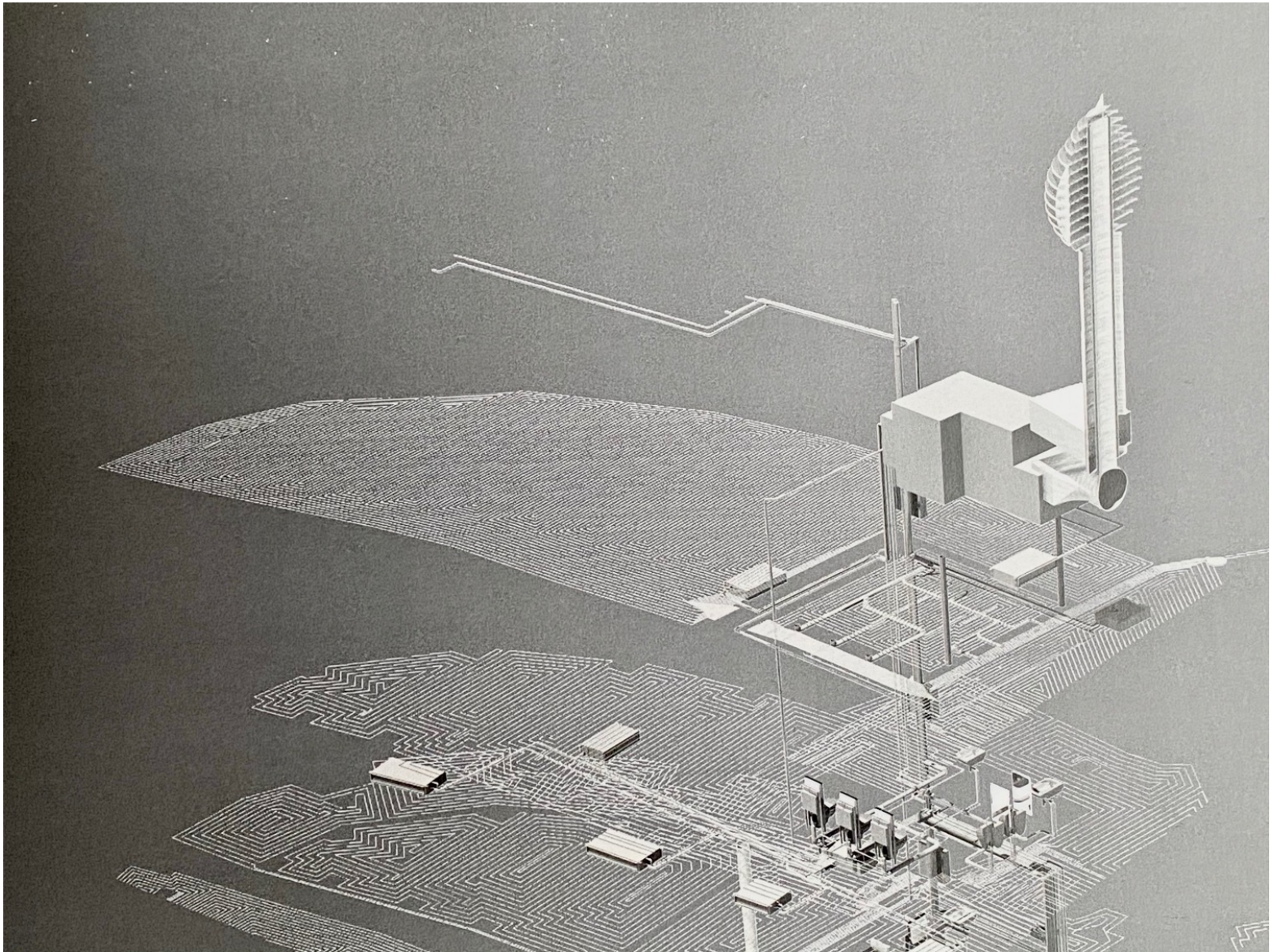


A view
on the
floors

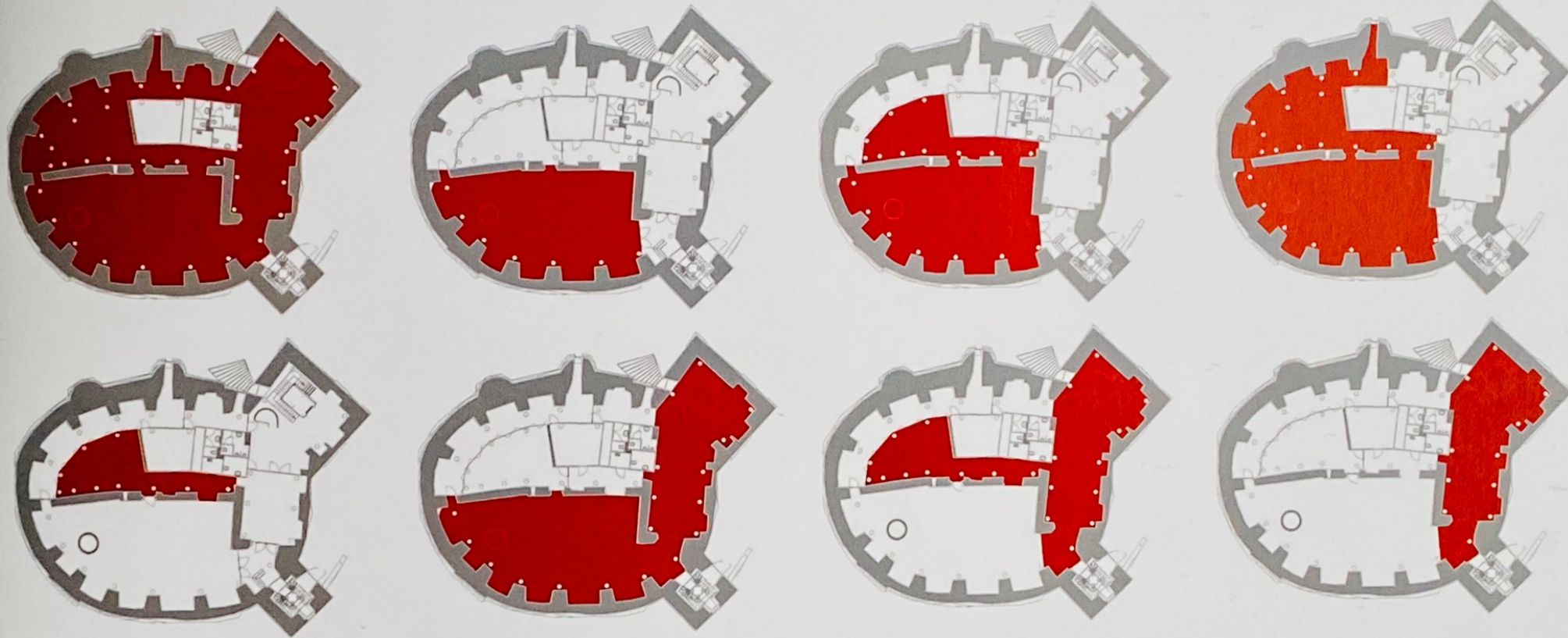
Design pattern
from Le Corbusier



A view
on the
air flow



The Room Configuration View



A view
on the
context



Views on
Kessel
Castle
Keverberg



Reconstruction 2015



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

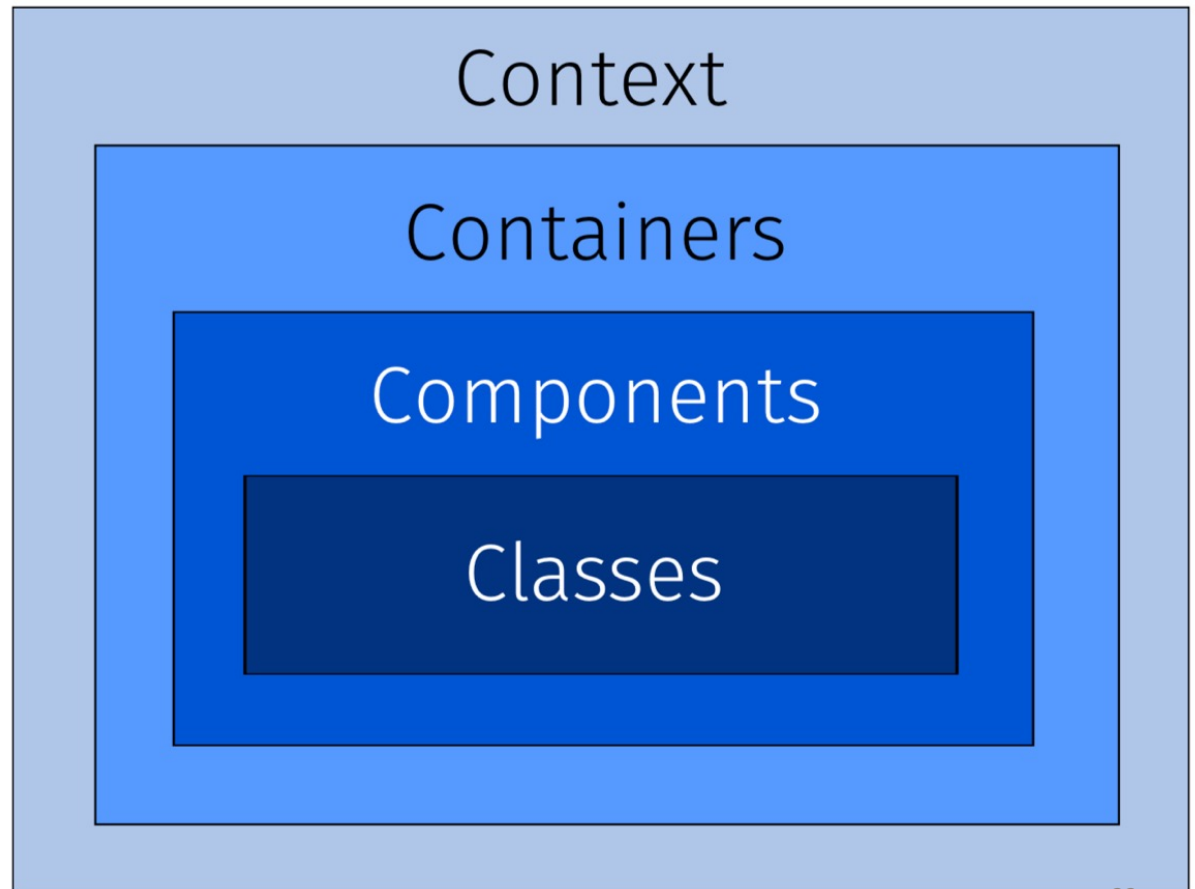
Component or pattern implementation details

**Overview
first**

**Zoom and
filter**

**Details
on demand**

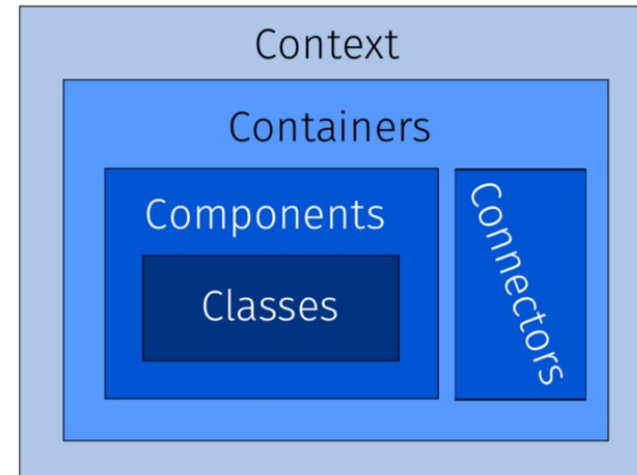
C4



<https://c4model.com/>

Connectors View

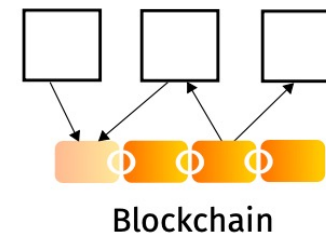
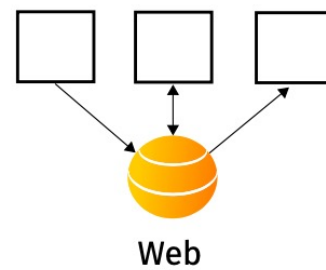
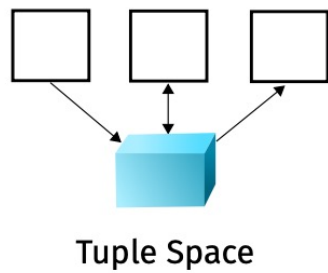
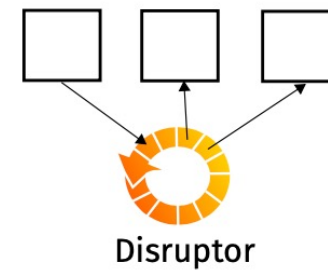
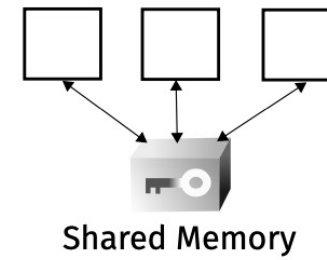
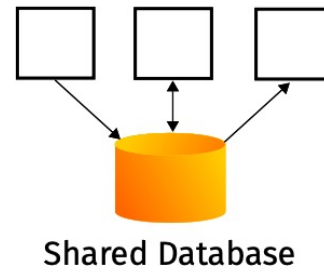
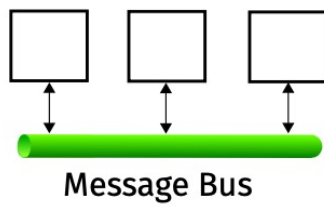
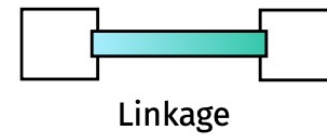
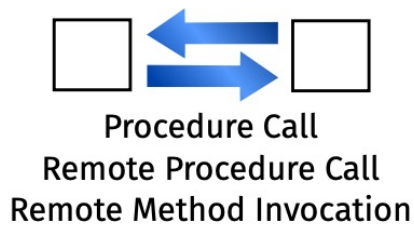
C5



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

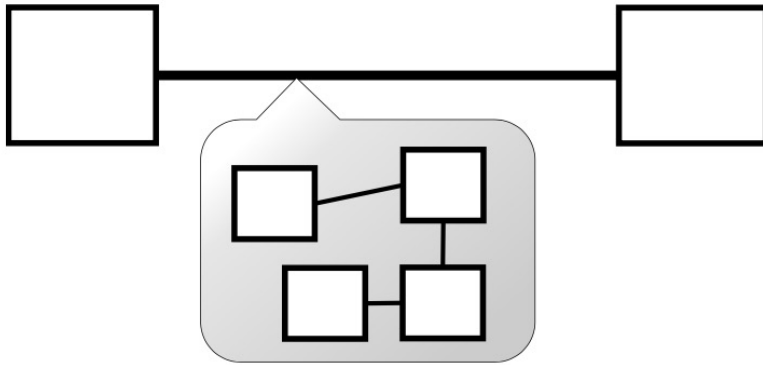
These decisions may depend on the deployment configuration

Software Connector Examples



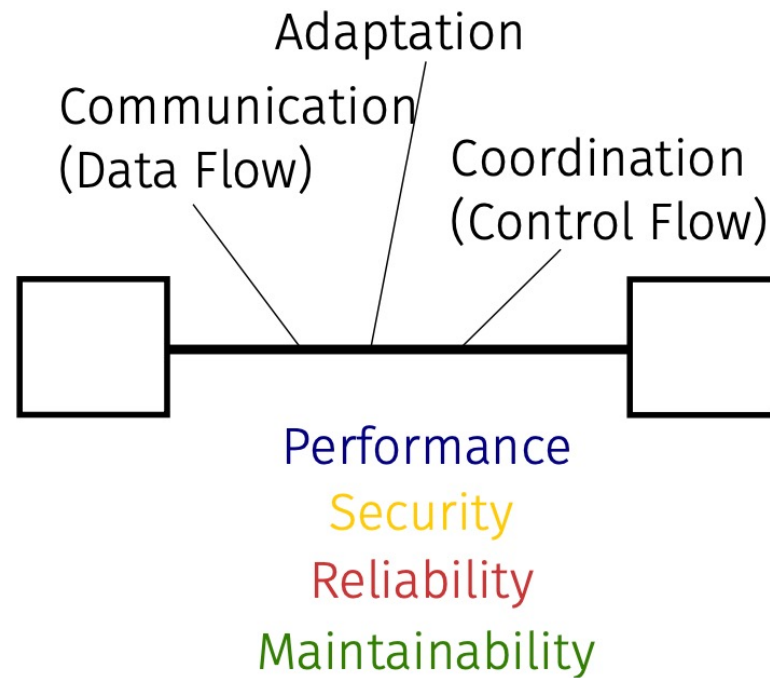
Connectors are Abstractions

- Connectors model interactions between components
- Connectors are built with (very complex) components



- Design Decision: when to hide away components inside a connector?

Connector Roles and Runtime Qualities



Connectors and Transparency

Direct

Components are directly connected and aware of the other component

```
f(x)
```

Indirect

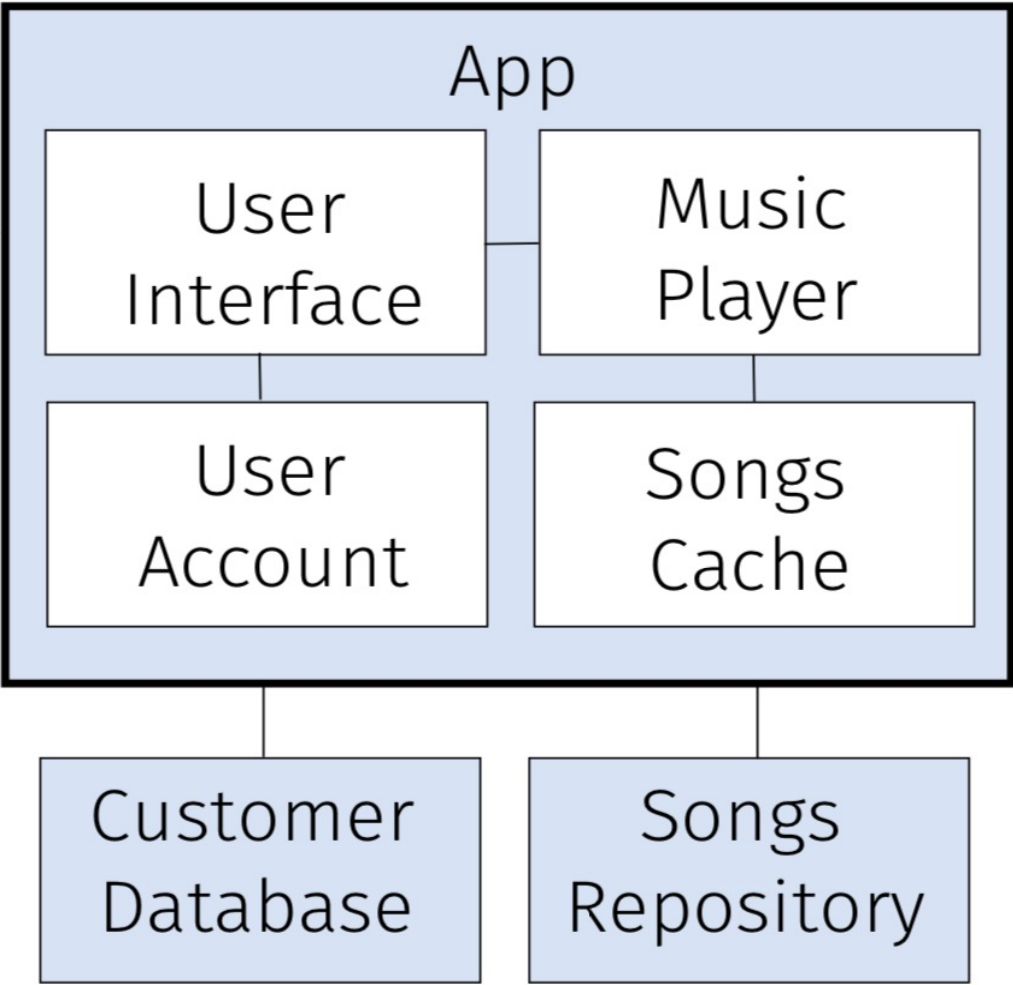
Components are connected to the others via the connector and remain unaware

```
queue.emit(m);
```

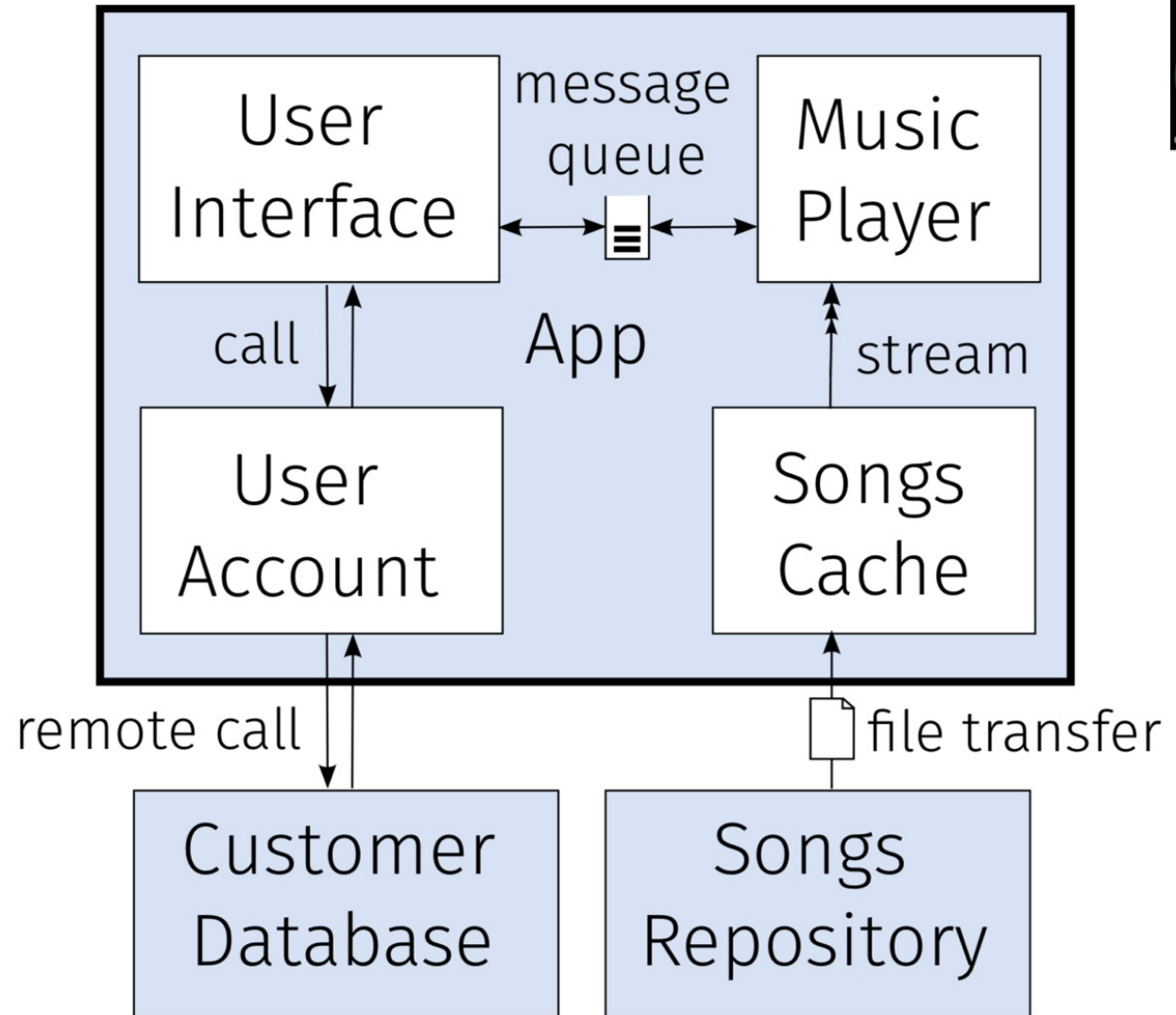
```
queue.on(m=>{});
```



Can you think of a (different) types of connectors for each line between two components?

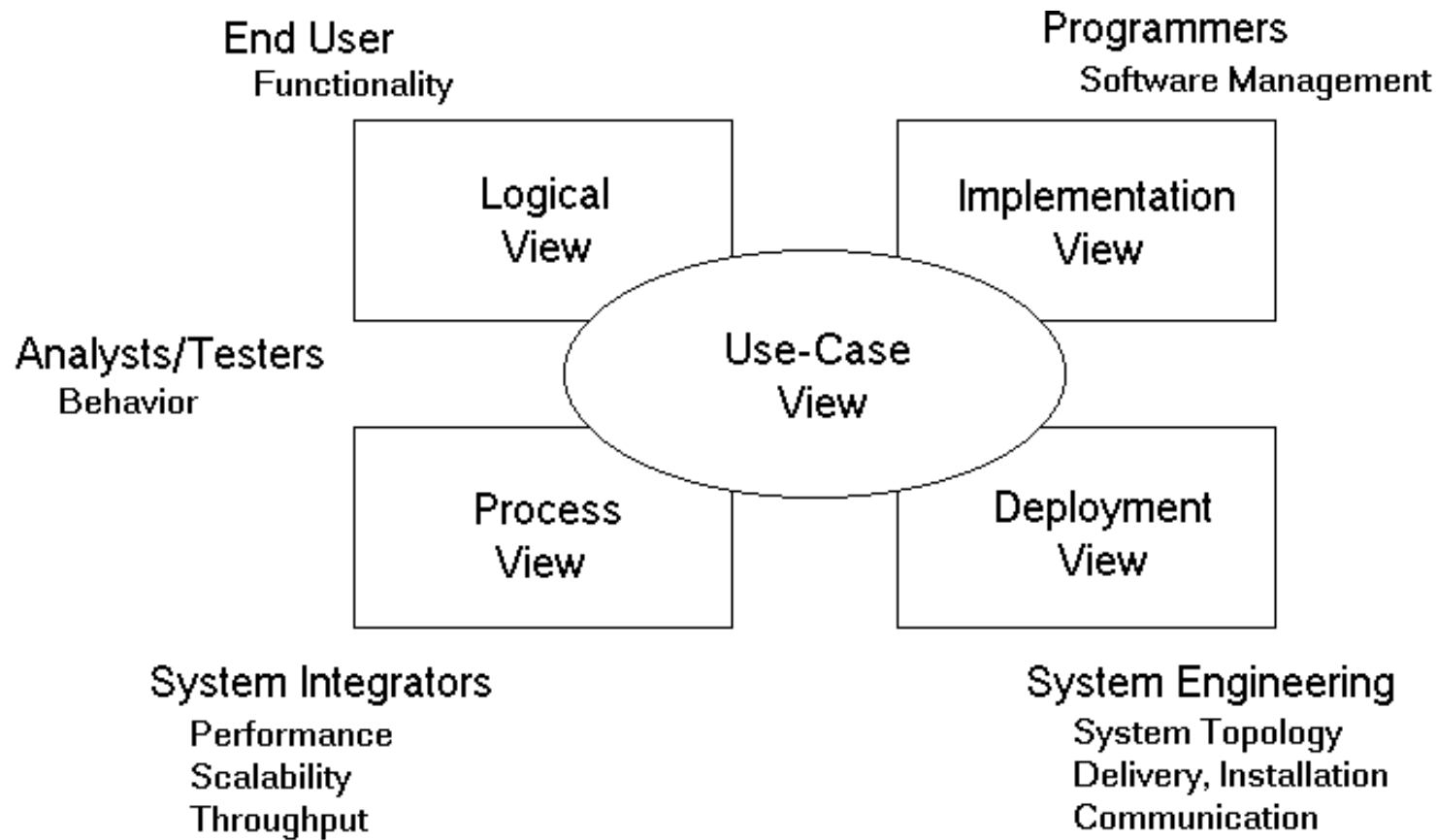


Connectors View Example





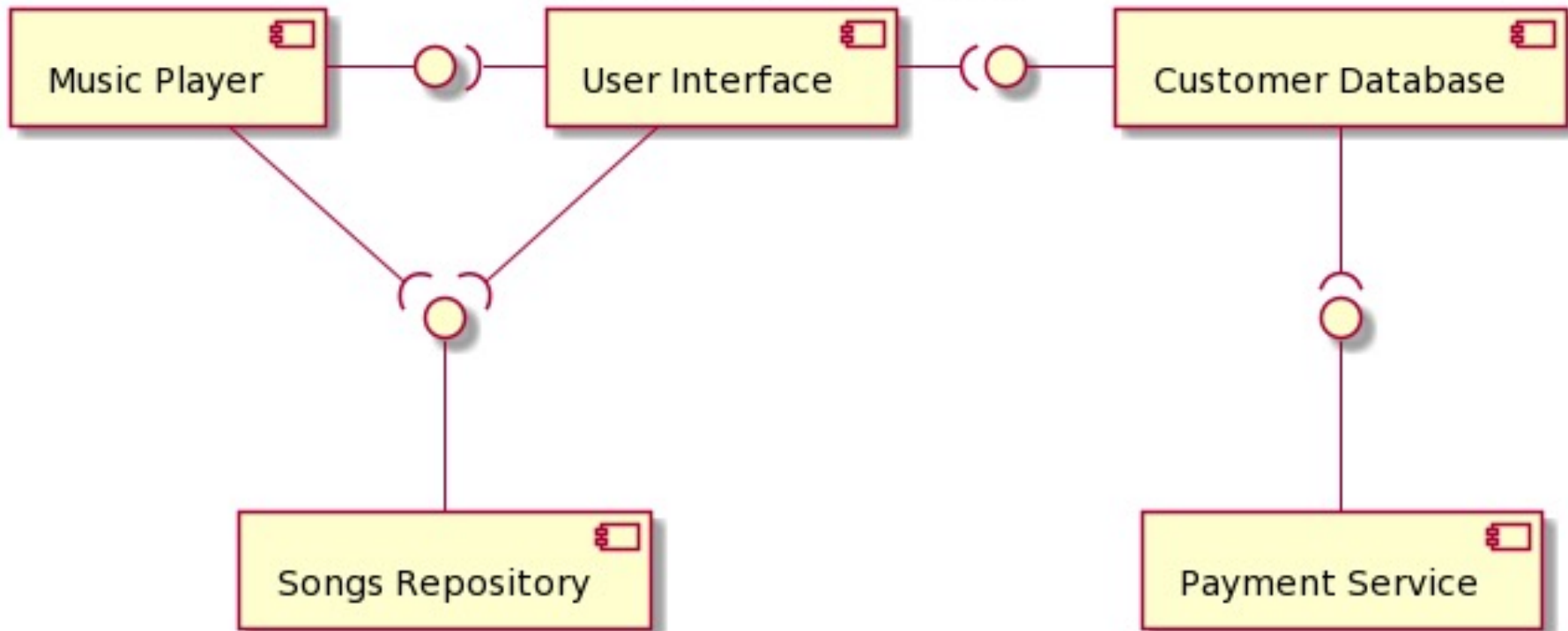
Philippe Kruchten's "4+1 Views"



Kruchten's "Logical View"

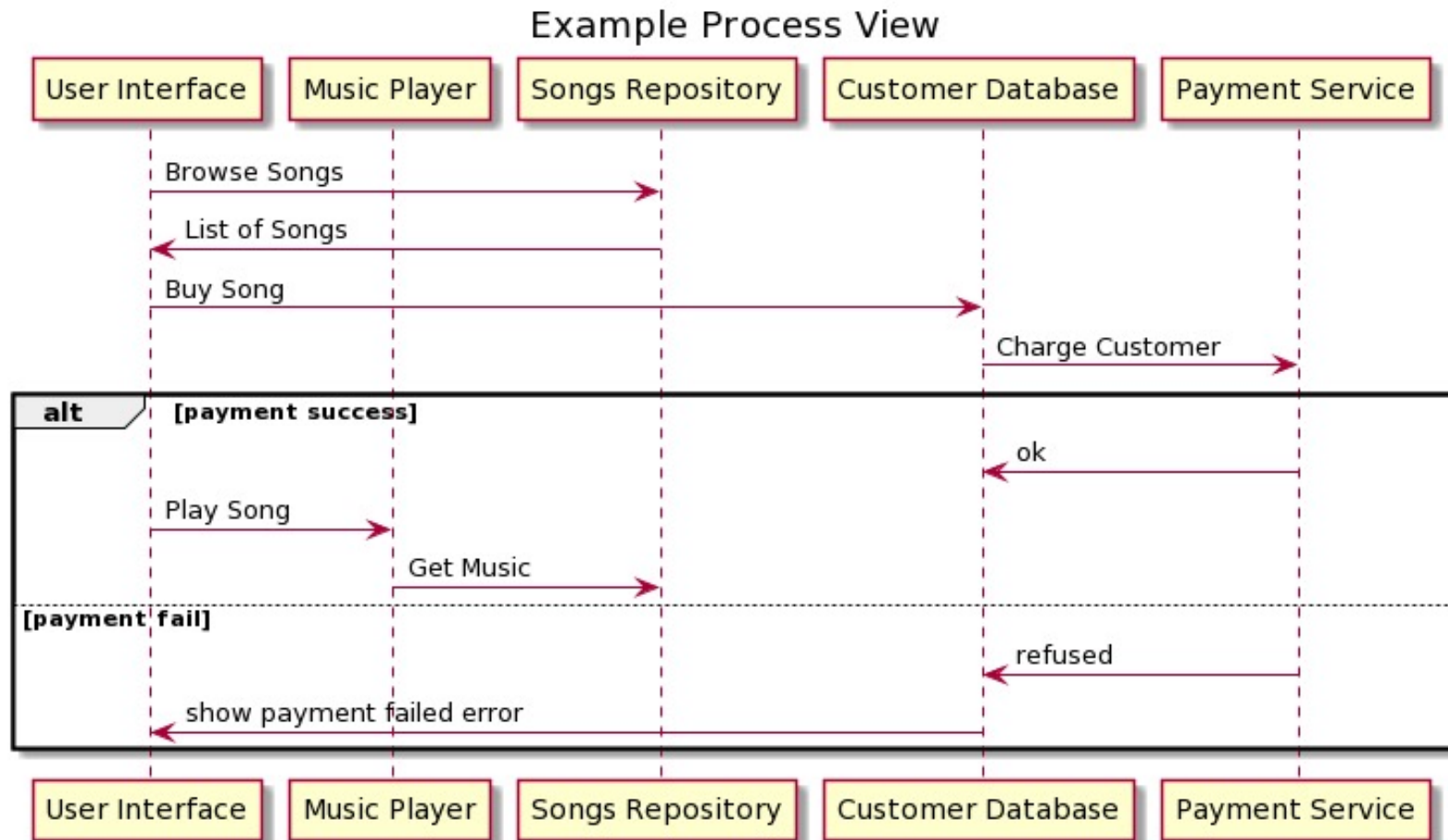
- Similar to C4 component view
- Decompose the system structure into software components and connectors
- Map functionality/requirements/use cases onto the components
- Concern: Functionality
- Target Audience: Developers and Users

Example Logical View



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

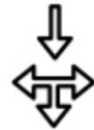


Kruchten's "Development View"

- 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

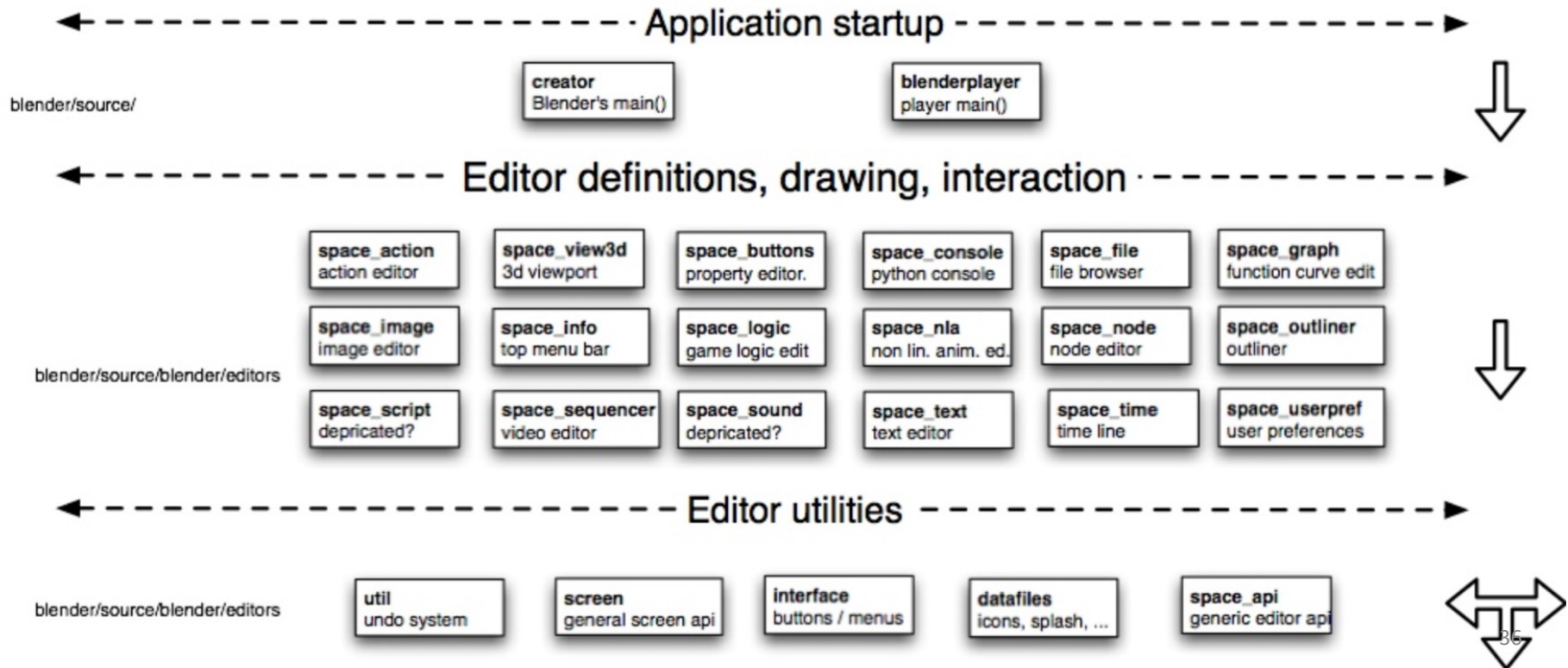
First line of thinking for
"us, developers"

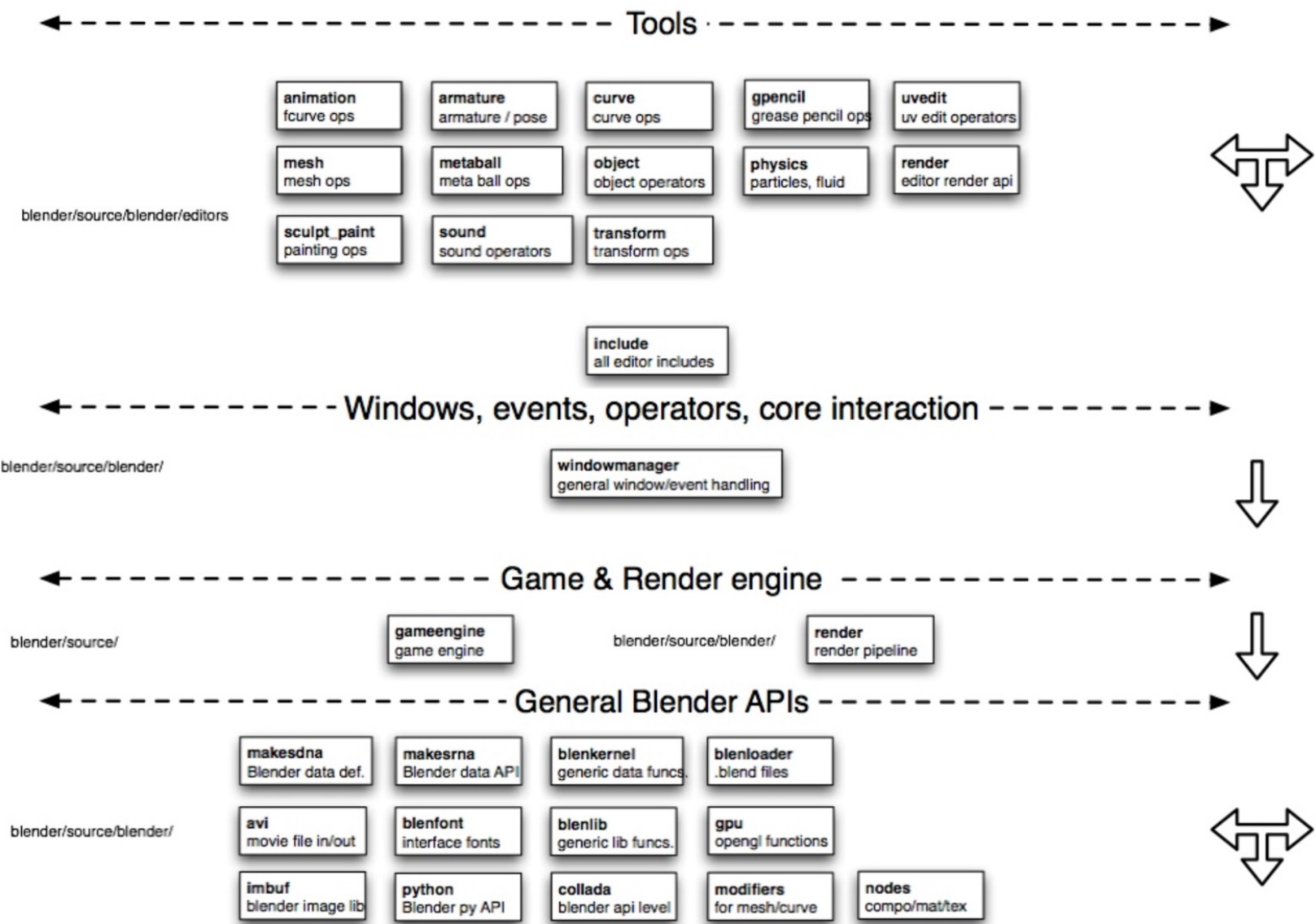
Blender code layout

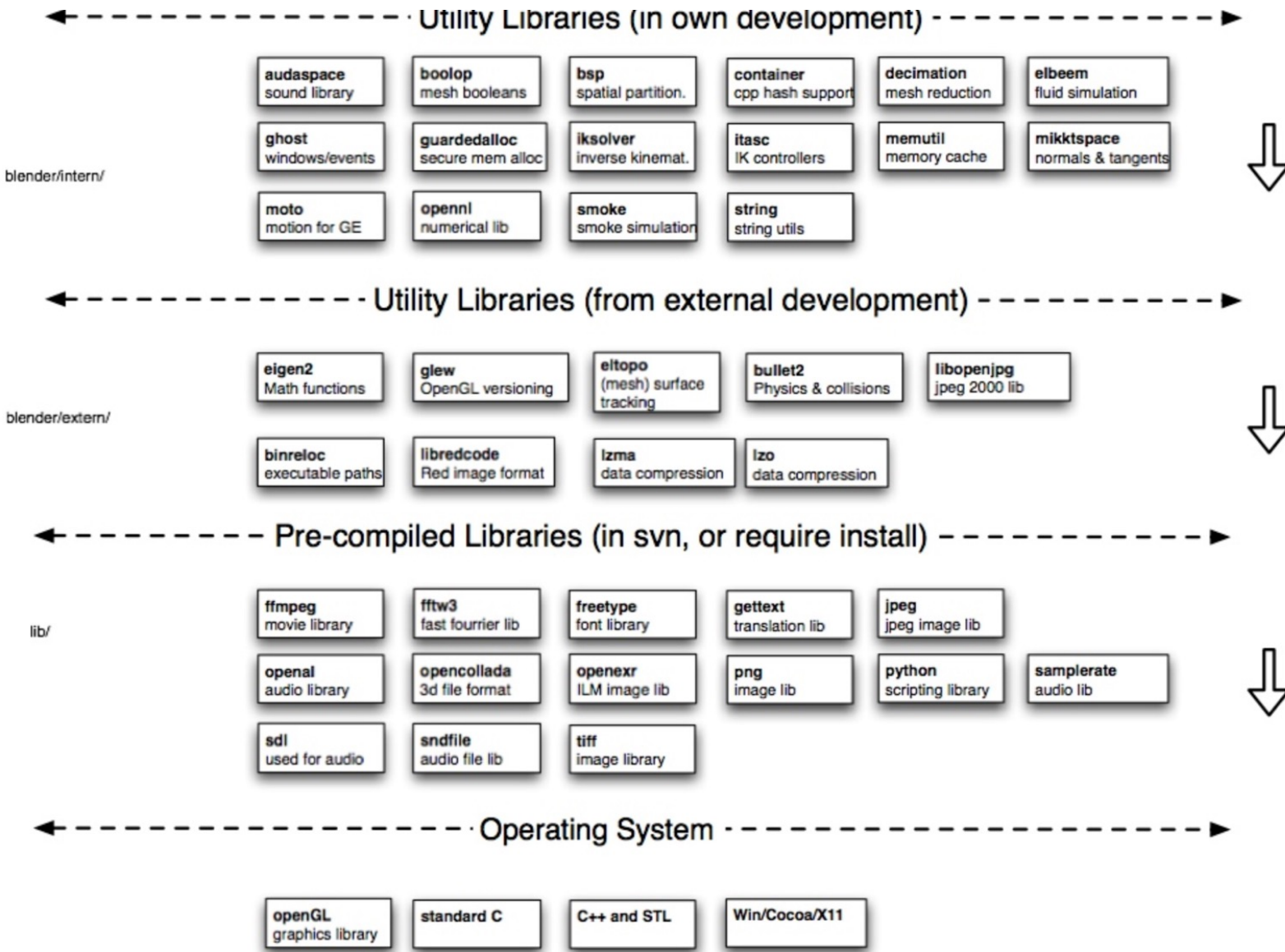


Modules only call lower level code

Modules call each other, and lower level code





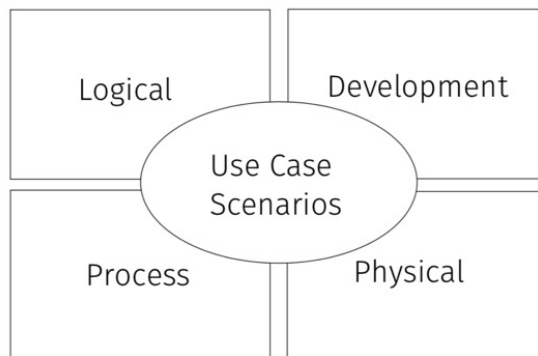


Kruchten's "Physical View"

- 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

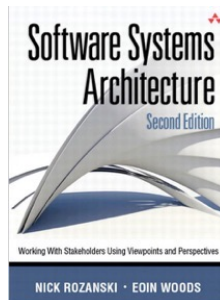
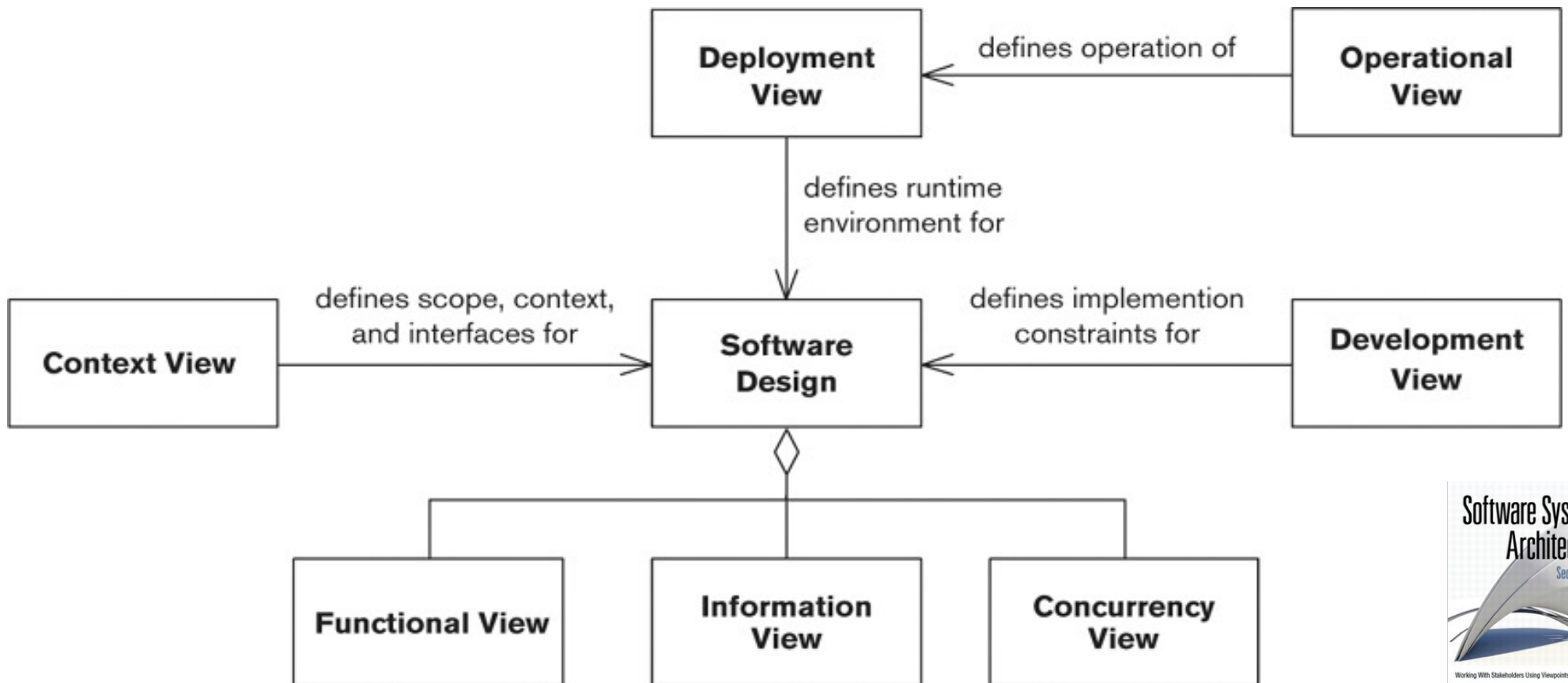
- Views should not contradict each other
- Use cases can be “executed” in each view

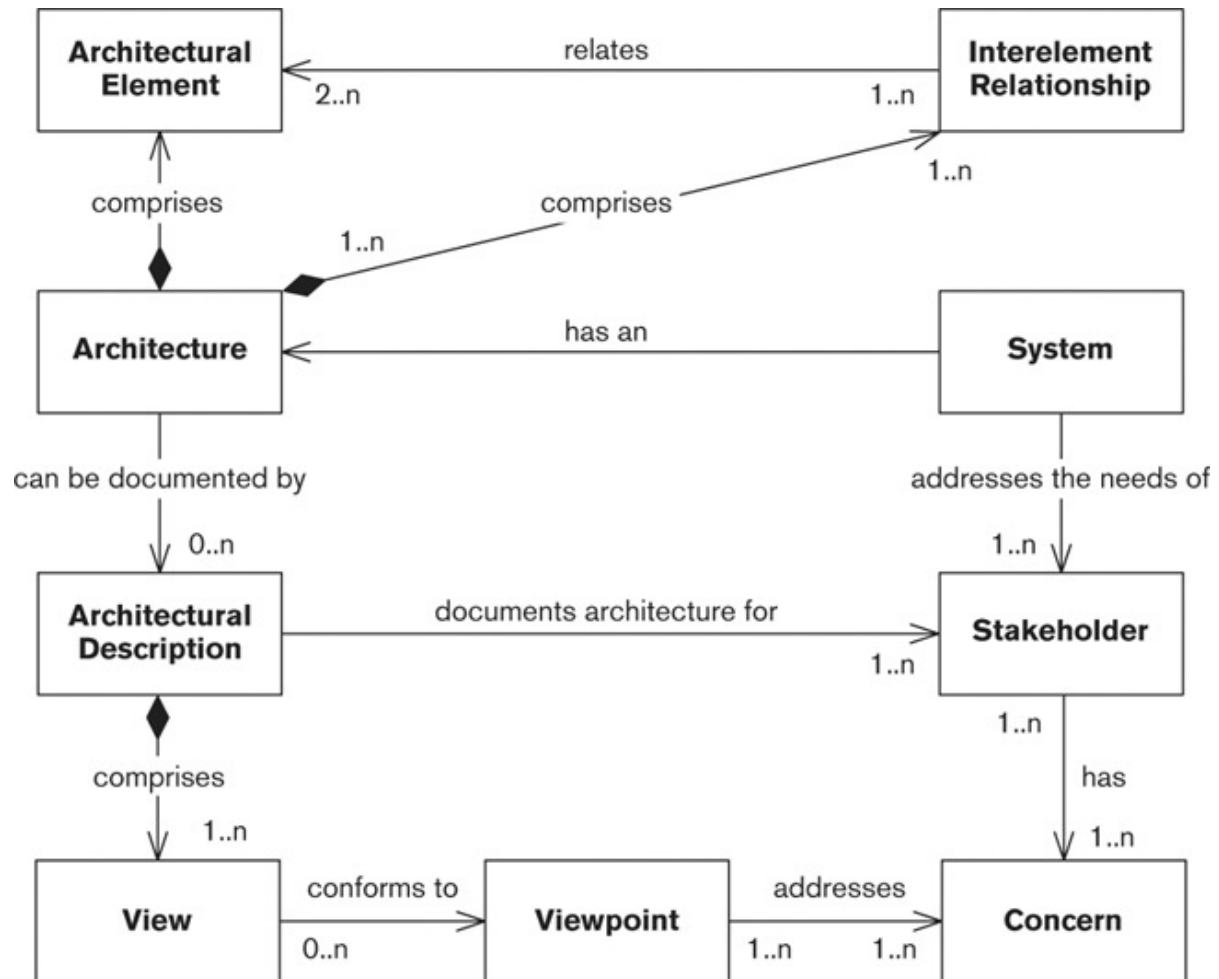


Example Music Player Scenarios

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

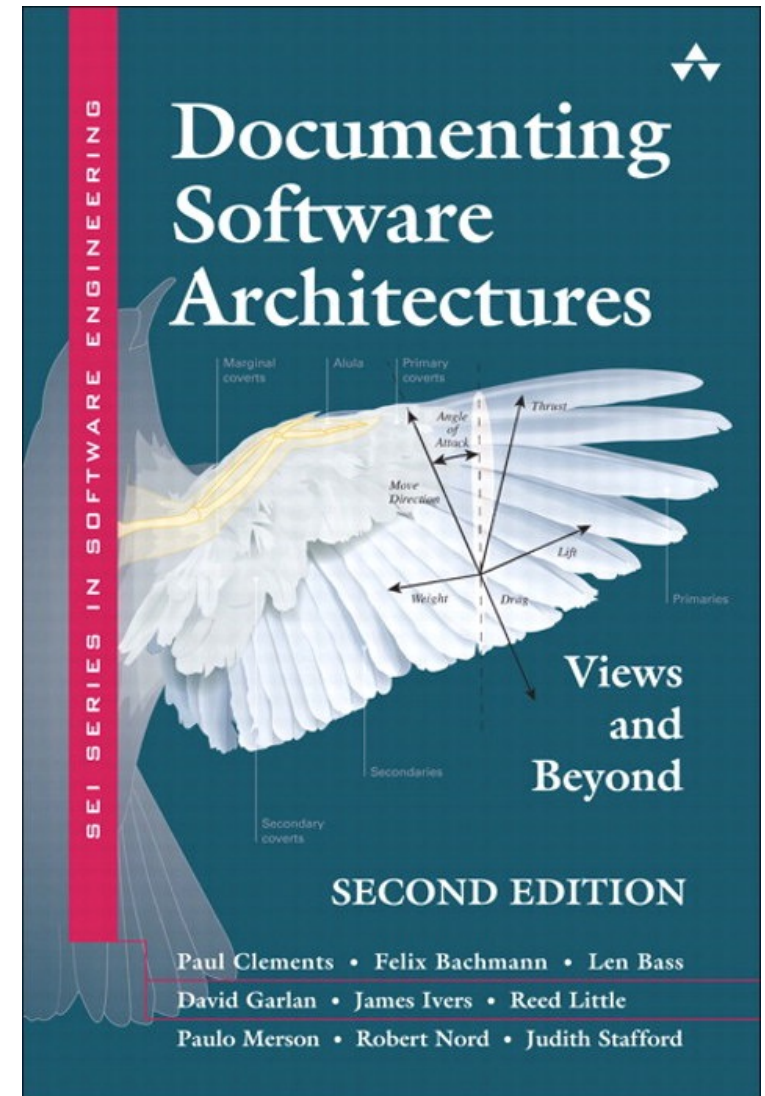
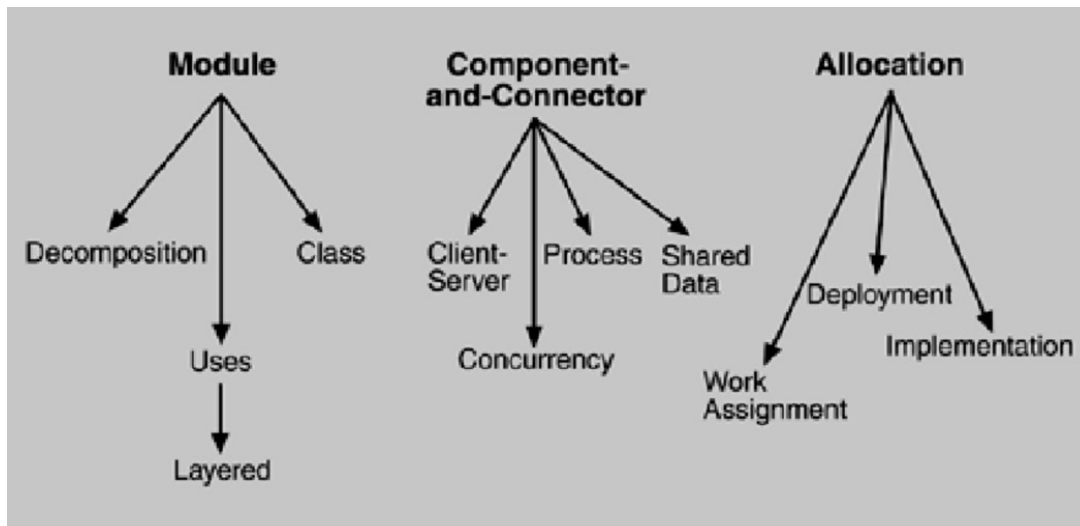




“SEI DSA” Taxonomy

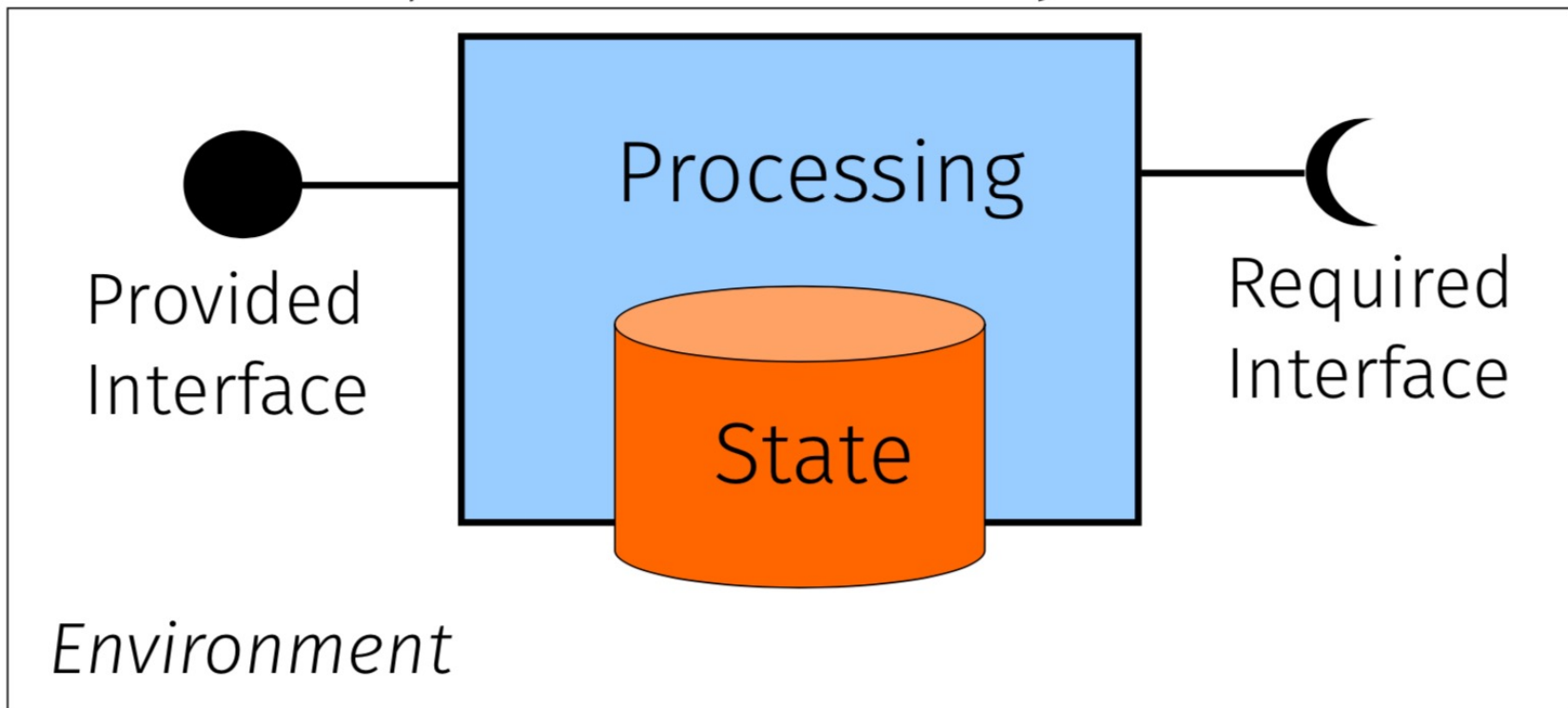
“View types”:

- Module
- Component & Connector
- Allocation

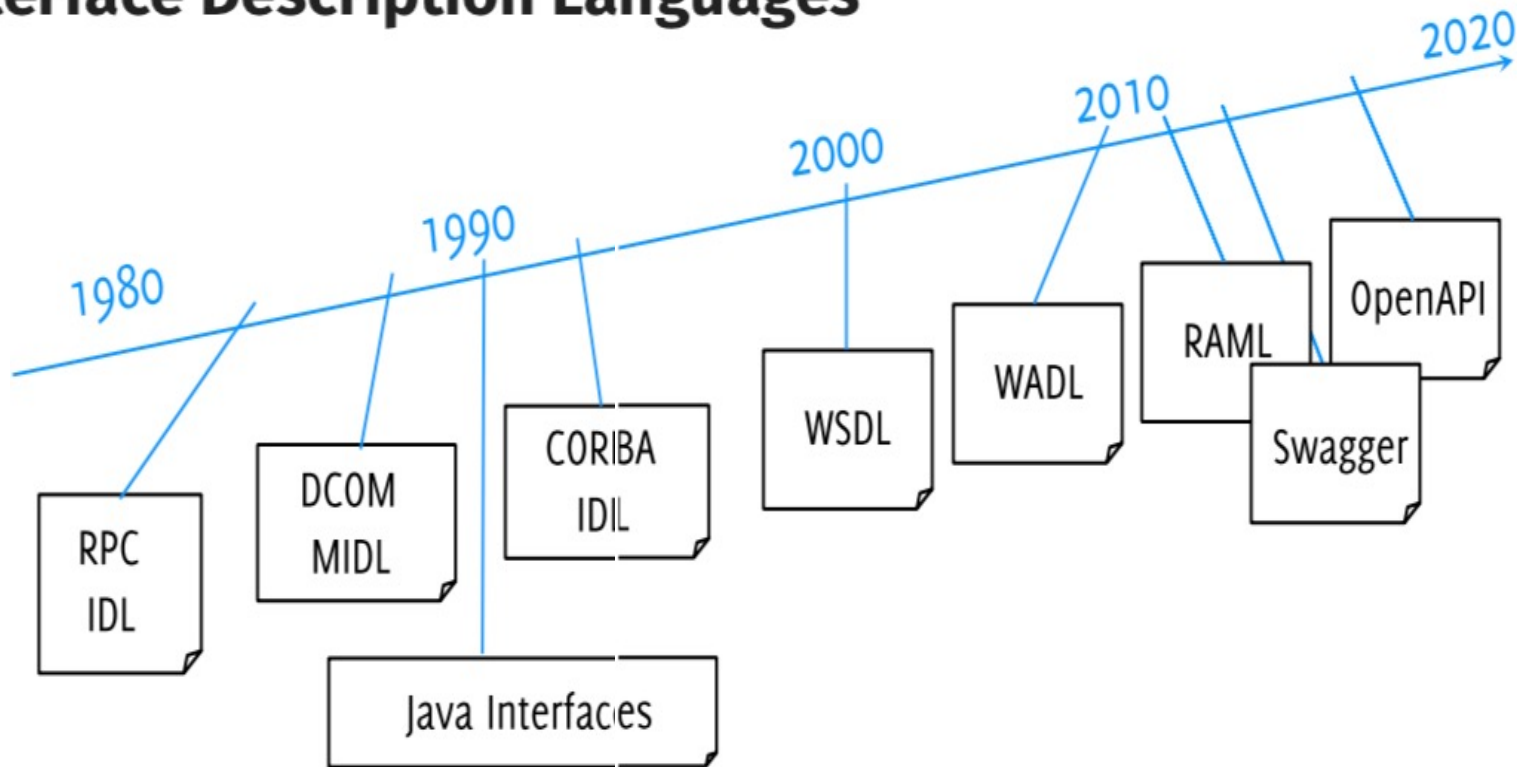


Software Component

- Locus of computation and state in a system

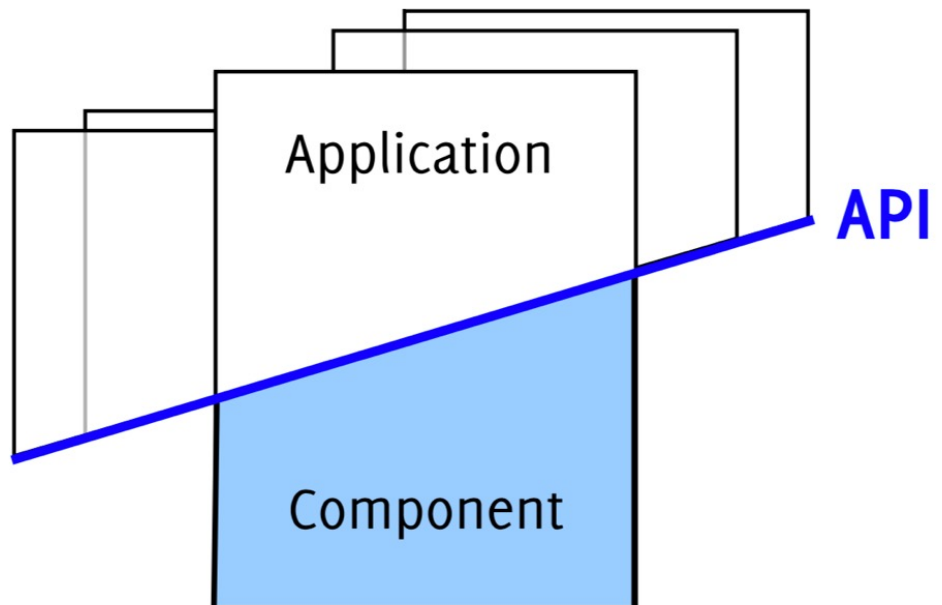


Interface Description Languages



Application Programming Interfaces

- APIs can be found in architectures that are designed to be
 - open and stable platforms
 - supporting externally developed components and applications.

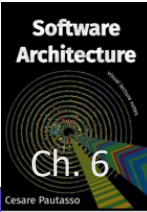


API Design Principles: Your Answers?

- Easy to understand
 - Usability
 - Simplicity
 - Small interfaces
- Quality of Service:
 - Scalability, Reliable, Available
- Compliance with standards
 - RESTful
- Licensing
- Naming consistency (end points, parameters, methods)
- Robust against untrusted clients
 - Security
 - Authentication
- Defensive API
- Meaningful error messages
- Compatibility

Design Advice

- Keep it simple
 - Do One Thing and do it well
 - Do not surprise clients
- Keep it as small as possible but not smaller
 - When in doubt leave it out
 - You can always add more later
- Maximize information hiding
 - API First
 - Avoid leakage: implementation should not impact interface



The slide is a presentation slide with a white background and a black border. On the left side, there are vertical bars in yellow, red, and purple. The title 'How to Design a Good API and Why it Matters' is centered in a large, bold, black font. Below the title, the author's name 'Joshua Bloch' is written in a smaller font, followed by his title 'Principal Software Engineer' and the Google logo. At the bottom left of the slide, there is a small number '1' and the text 'How to Design a Good API and Why it Matters'. At the bottom right, there is a URL: 'http://www.cs.bc.edu/~muller/teaching/cs102/s06/lib/pdf/api-design'.

<http://www.cs.bc.edu/~muller/teaching/cs102/s06/lib/pdf/api-design>

Design Advice

- Names Matter
 - Avoid cryptic acronyms
 - Use names consistently
- Internally Consistent
 - Naming Conventions
 - Argument Ordering
 - Return values
 - Error Handling
- Externally Consistent
 - Imitate similar APIs
 - Follow the conventions of the underlying platform

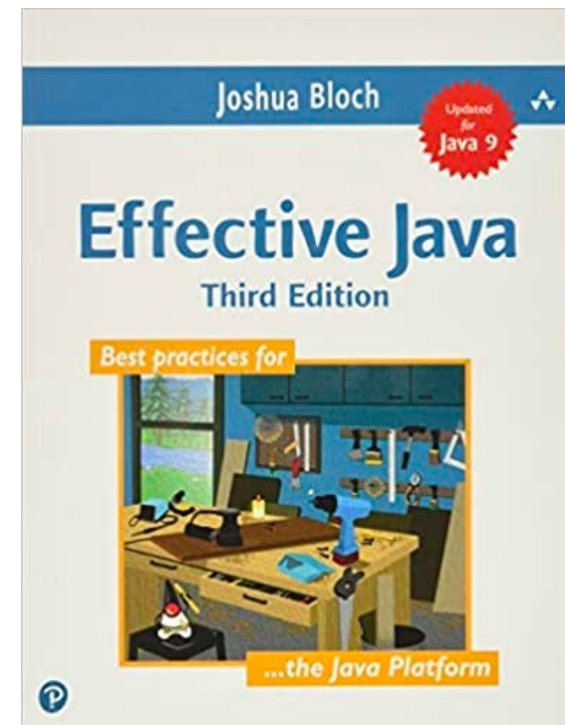
Joshua Bloch



<https://www.youtube.com/watch?v=aAb7hS0tvGw>

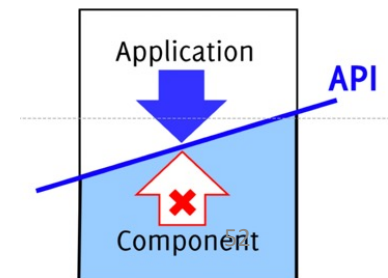
Design Advice

- Document Everything
 - Classes, Methods, Parameters
 - Include Correct Usage Examples
 - Quality of Documentation critical for success
- Make it easy to learn and easy to use
 - without having to read too much documentation
 - by copying examples
- Make it hard to misuse



API Design Principles

- Explicit interfaces principle
- Principle of least surprise
- Small interfaces principle
- Uniform access principle
- Few interfaces principle
- Clear interfaces principle
- Maximize information hiding
- 90% immediate use; 9% with effort; .9% misuse
- Balance usability and reusability
- Balance performance and reusability
- Design from client's perspective



API Reflection

- Consider an application you know well
- Which public APIs does it expose?
- Does the API realize a clear, compelling function?
- Which of the principles discussed does it adhere to explicitly?
 - Which ones does it violate?
- Is the design rationale behind the API documented?

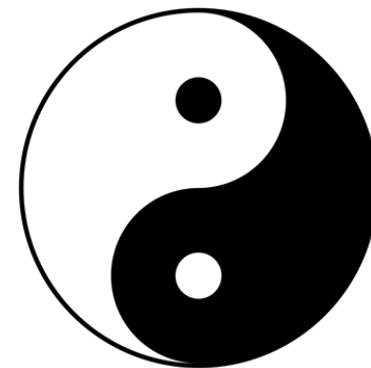
Essay 2: The System's Architecture

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

Dialectic Learning in Architecture

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

Thesis:	Theory
Anti-Thesis:	Practice
Synthesis:	Understanding



This Year's 38 Teams and Systems

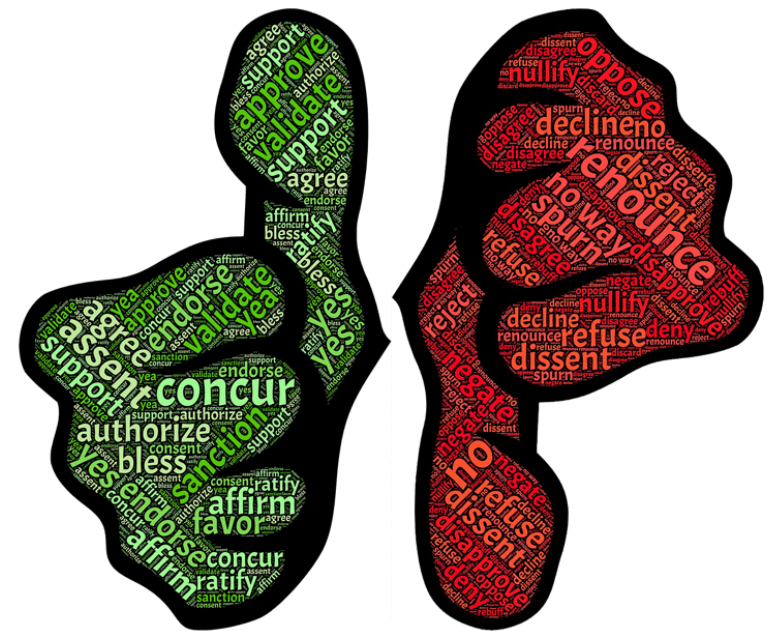
1	elasticsearch
2	pandoc
3	scrapy
4	robotframework
5	assertj
6	pmd
7	netdata
8	ghidra
9	beets
10	storybook
11	egeria
12	dolphin
13	hugo

14	react-native
15	godot
16	checkstyle
17	quodlibet
18	selenium
19	expressjs
20	react
21	wireshark
22	backstage
23	processing
24	sonic-pi
25	mattermost-server
26	log4j2

27	cheat-engine
28	audacity
29	serenity-os
30	snakemake
33	project64
42	prettier
51	mattermost-mobile
62	metamask
65	wikipedia-android
69	element
88	moby
96	podman

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

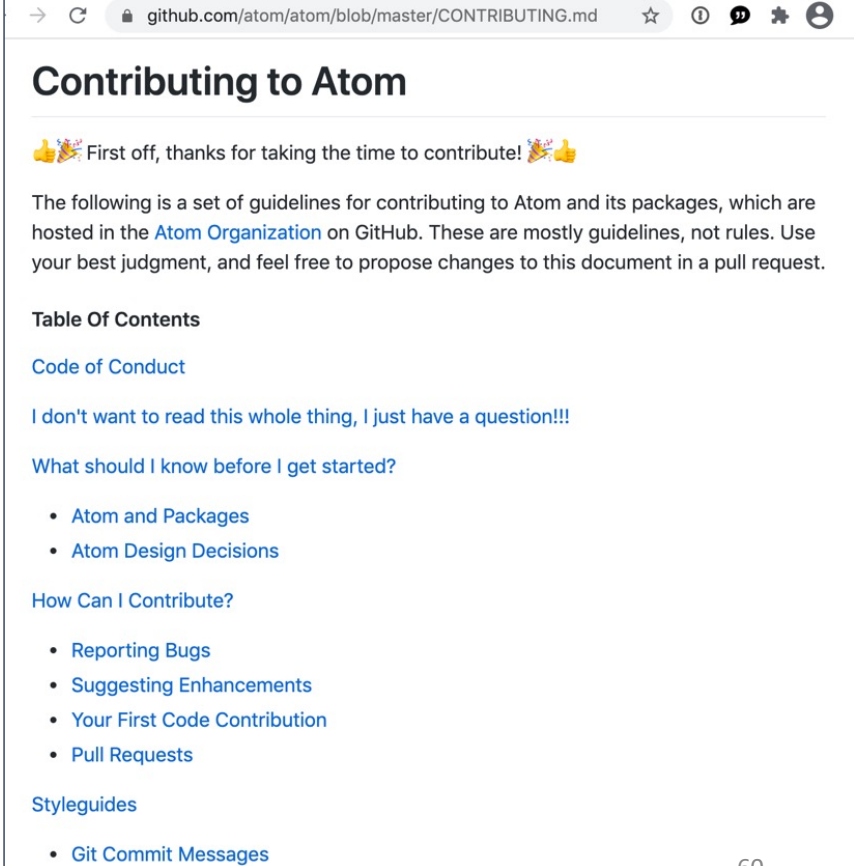
- Documentation
- Internationalization
- Report an issue
- Add some tests (e.g. reproducing a bug)
- Fix a reported bug (with test case)
- Add requested feature (with test case)
- Propose feature (in issue) and build it
- Remove unused or redundant code
- ...

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, ...)



The screenshot shows a web browser window displaying the GitHub page for the Atom organization's CONTRIBUTING.md file. The browser's address bar shows the URL: `github.com/atom/atom/blob/master/CONTRIBUTING.md`. The page title is "Contributing to Atom". The content begins with a friendly message: "First off, thanks for taking the time to contribute!". It then explains that the following are guidelines for contributing to Atom and its packages, hosted in the Atom Organization on GitHub, and that these are mostly guidelines, not rules. The page includes a "Table Of Contents" section with the following links: "Code of Conduct", "I don't want to read this whole thing, I just have a question!!!", "What should I know before I get started?" (which includes sub-links for "Atom and Packages" and "Atom Design Decisions"), "How Can I Contribute?" (which includes sub-links for "Reporting Bugs", "Suggesting Enhancements", "Your First Code Contribution", and "Pull Requests"), and "Styleguides" (which includes "Git Commit Messages").

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

- [Code of Conduct](#)
- [I don't want to read this whole thing, I just have a question!!!](#)
- [What should I know before I get started?](#)
 - [Atom and Packages](#)
 - [Atom Design Decisions](#)
- [How Can I Contribute?](#)
 - [Reporting Bugs](#)
 - [Suggesting Enhancements](#)
 - [Your First Code Contribution](#)
 - [Pull Requests](#)
- [Styleguides](#)
 - [Git Commit Messages](#)

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



The screenshot shows a web browser window with the URL `apache.org/licenses/contributor-agreements.html`. The page features the Apache Software Foundation logo, which includes a feather icon and the text "THE APACHE SOFTWARE FOUNDATION 20TH ANNIVERSARY". Below the logo is the slogan "COMMUNITY-LED DEVELOPMENT 'THE APACHE WAY'". A circular "SUPPORT APACHE" logo is also visible. The main heading is "ASF CONTRIBUTOR AGREEMENTS". The text explains that the ASF uses various agreements to accept contributions and grants, and that these agreements help achieve the goal of providing reliable and long-lived software products through collaborative open source development. The page also lists "CONTRIBUTOR LICENSE AGREEMENTS" with sub-points for ICLA (Individual Contributor License Agreement) and CCLA (Corporate Contributor License Agreement).

← → ↻ 🔒 apache.org/licenses/contributor-agreements.html ☆ ⓘ 🗨️ ⚙️ 👤 ⋮

 THE APACHE®
SOFTWARE FOUNDATION
20TH ANNIVERSARY

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

61

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!

- 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?



Image credit: wikipedia

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.

MERGED

OPEN PR [↗](#)

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.

MERGED

OPEN PR [↗](#)

Docs: Add missing directories in readme.md

OpenRCT2/OpenRCT2

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

MERGED

OPEN PR [↗](#)

Fix #10993: Guest Count Intent Not Listened To

OpenRCT2/OpenRCT2

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

MERGED

OPEN PR [↗](#)

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.

MERGED

OPEN PR [↗](#)

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.

MERGED

OPEN PR [↗](#)

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

MERGED

OPEN PR [↗](#)

[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.

OPEN

OPEN PR [↗](#)

desosa.nl/projects/gitlab/

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.

MERGED OPEN PR

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.

MERGED OPEN PR

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'.

OPEN OPEN PR

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.

MERGED OPEN PR

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.

MERGED OPEN PR

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/>
- Diomidis Spinellis. Why computing students should contribute to open source software projects. CACM 64(7):36-38, July 2021.
<https://dl.acm.org/doi/10.1145/3437254>