IT University of Copenhagen

1 Software Architecture Reconstruction: Introduction

Mircea Lungu

github.com/mircealungu/reconstruction

1.1 Imagine . . .

- Onboarding on a new system
- Buying a software company
- Having to do
    - an architectural evaluation
    - a risk assessment for security

Riddle: What would be nice to have in all these circumstances but we almost never have?

Helper: What is the software artifact that you are not guaranteed to have, not even when paying 50B for a company?
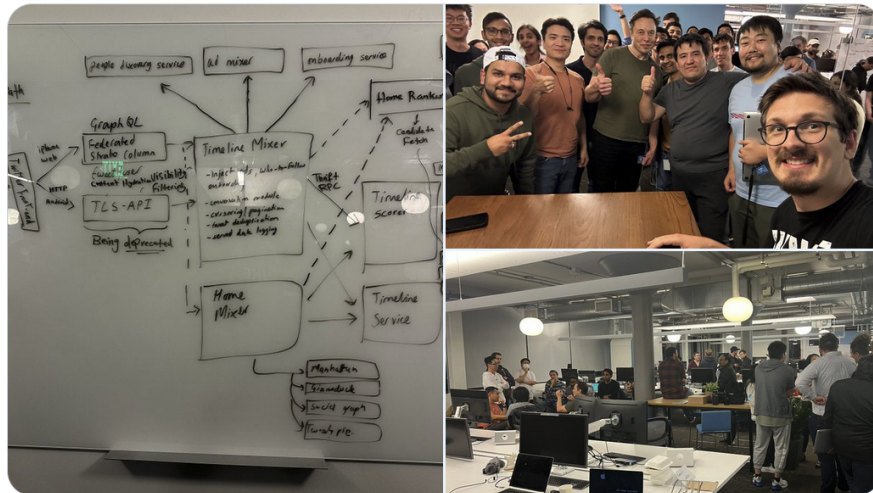
1.2 An up-to-date architectural diagram

Elon Musk ✓
@elonmusk

Just leaving Twitter HQ code review

link to original tweet

1.3 Discussion

Have you seen architectural documentation for every system?

- No, *Why is it missing?*
- Yes?
    - Is it up to date?
    - *No? Why not?*

1.4 Why does architectural documentation not always exist and is not up to date?

Incentives - Sometimes that's not a priority at all - you're a startup that needs to show that the idea is viable. And then you never find the time.

- Maybe you're designing your own product and nobody to ask you to do it

- Often there is no perceived value for the customer (or more likely, no clear immediate value)

Difficulty

- Traceability (link) between architecture and code is not easy to establish

- It requires a better and more general understanding of the system than just coding -> not everybody can do it

- Hard to maintain - especially when they are in .ppt or .png

- Because developers make decisions and changes
    - that are not aligned with the original vision => **architectural drift**
    - that go against prescriptive architecture => **architectural erosion**

1.4.1 Architecture Erosion Example
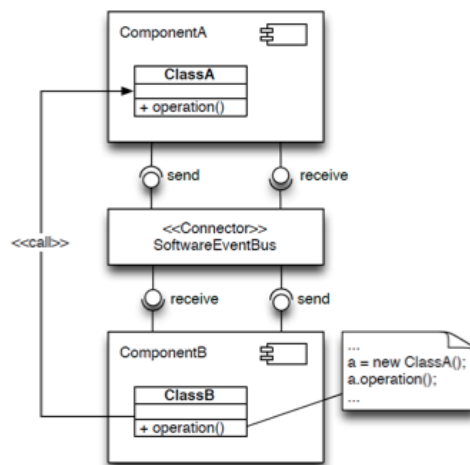
What could be the cause of erosion here?

Why would it be a problem?

1.5 How to Keep Architectural Documentation up to Date?

1 / **Enforcing architectural constraints** - Type systems – too low-level - Special tools for architecture constraints definition - DSL - domain specific language (e.g. Dictō) - tools that take inspiration from unit testing (e.g. ArchUnit) - How to integrate these tools? - CI/CD - Pre-commit hooks - IDE - Declarative architecture: docker compose, swarm stack specifications, infrastructure-as-code specifications, etc.

–

2 / **Generating architectural diagrams from code** - As opposed to *drawing*

them in Powerpoint - No great tools for this - often too much low-level noise - The focus of: **reconstructing the Architecture**

2   Architecture Reconstruction (AR)

a.k.a: *architecture recovery* (the two are used interchangeably)

(def.) **A reverse engineering approach that aims at reconstructing viable architectural views of a software application** [1]

- reverse engineering?

[1] Ducasse & Pollet, Software Architecture Reconstruction: a Process-Oriented Taxonomy

2.1   Reverse Engineering

(def.) the **process** of analyzing a subject system to **identify** the system's **components** and their **interrelationships** and create representations of the system [. . .] at a **higher level of abstraction**. (Demeyer et al., Object Oriented Reengineering Patterns, Chapter 1.2)

Focus on - components - relationships - higher level of abstraction

Relation with architecture recovery? They are overlapping activities and use overlapping methods.

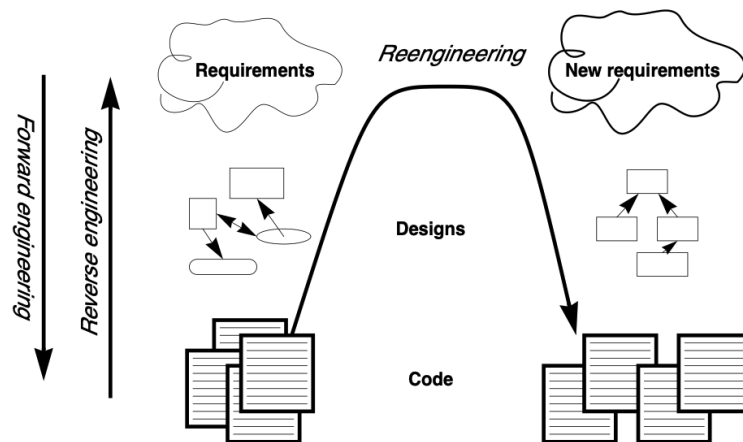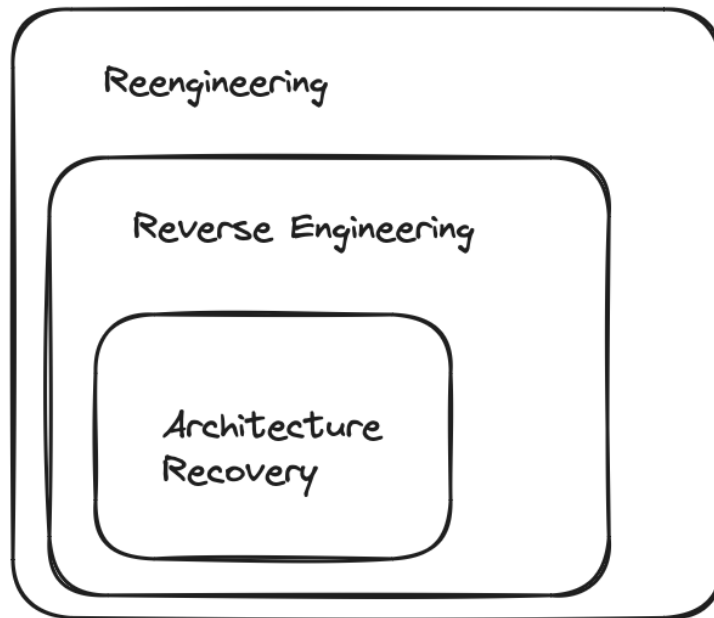## 2.2   Reverse Engineering vs. Reengineering?

Figure 1.1: Forward, reverse and reengineering

4

"Reengineering is the **examination and alteration** of a subject system to reconstitute it in a new form" (Demeyer et al., Object Oriented Reengineering Patterns, Chapter 1.2)

? Relation with AR? AR could be a possible first step in reengineering

3   How To Do Architecture Reconstruction?

Symphony: View-Driven Software Architecture Reconstruction

- Classical, principled way
- View-driven approach
- Distinguishes between three kinds of *views*
    1. **Source**
        - view represents directly artifacts of a system
        - not necessarily architectural (e.g. see later example)
    2. **Target**
        - describes architecture-as-implemented
        - any of the (first three from the) 3+1 views
    3. **Hypothetical**
        - architecture-as-designed
        - existing documentation
        - presentations

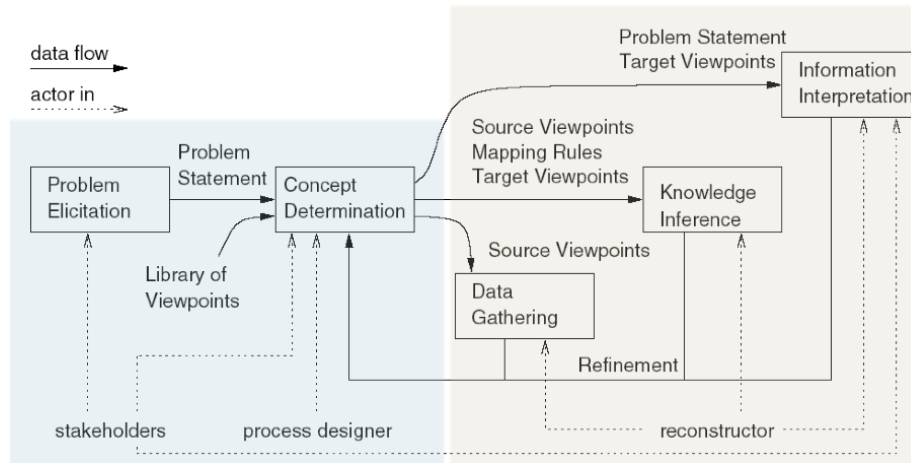## 3.1 Symphony Stages: Design (blue) & Execution (yellow)
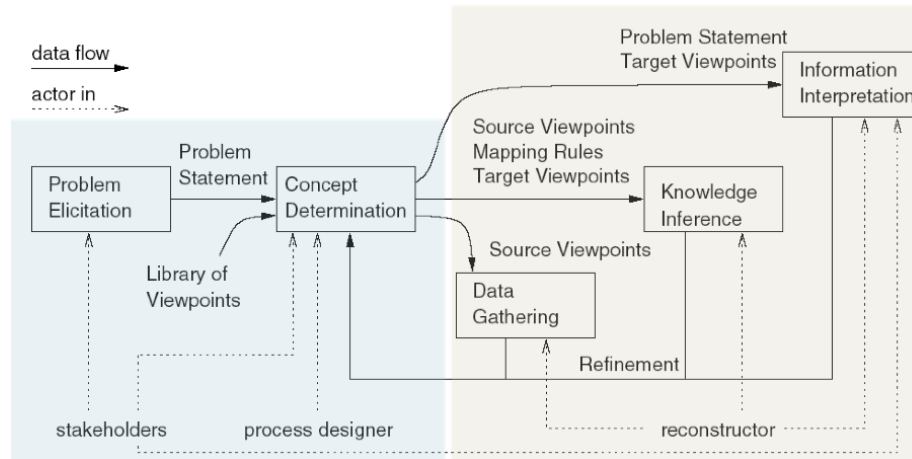
3.1.1 Desgin: Problem elicitation

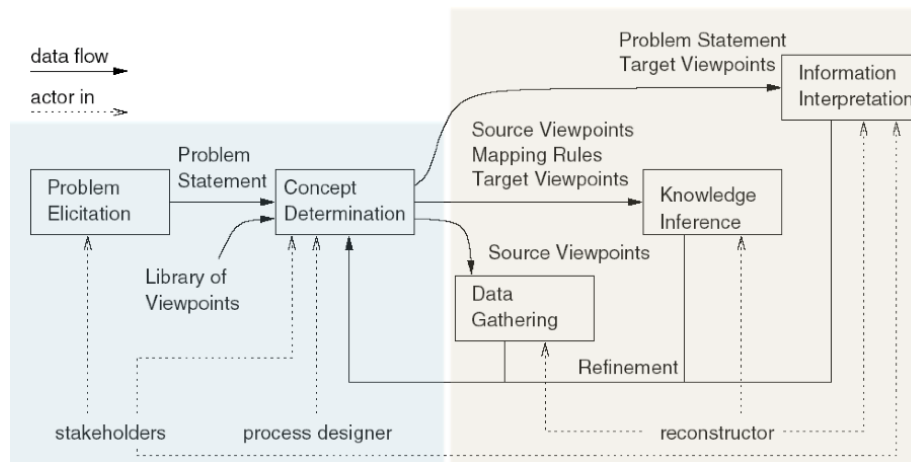- "Business case" for reconstruction
- What is the problem?
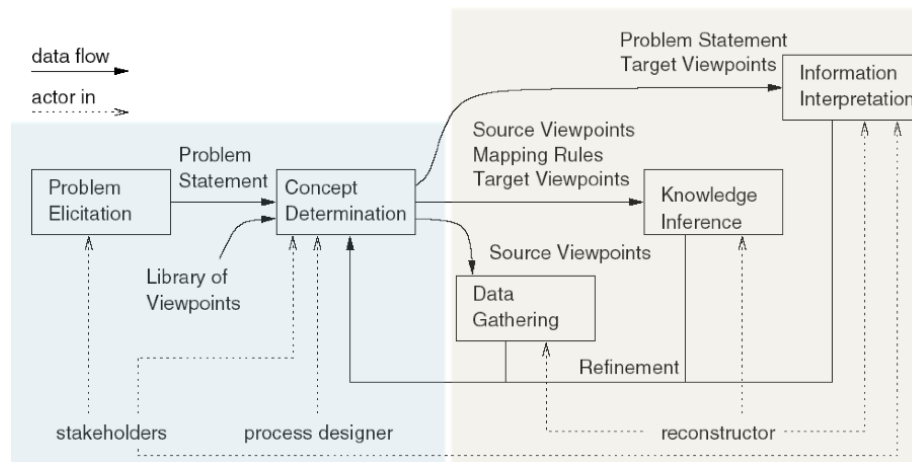
3.1.2 Design: Concept determination

- What architectural information is needed to solve the problem?
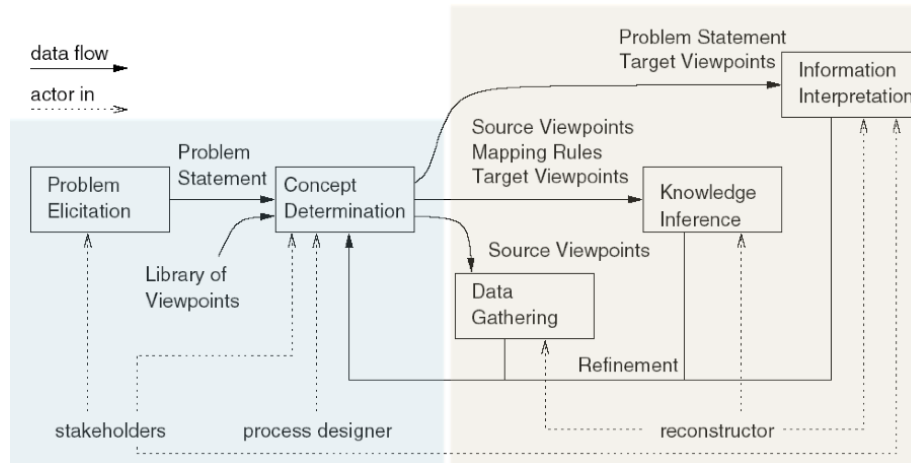- **Which viewpoints are relevant?**

3.1.3 Execution: Data gathering

- Collecting and extracting low-level **source views**
- Can involve a multitude of sources even besides source code (e.g., git repo, runtime information)
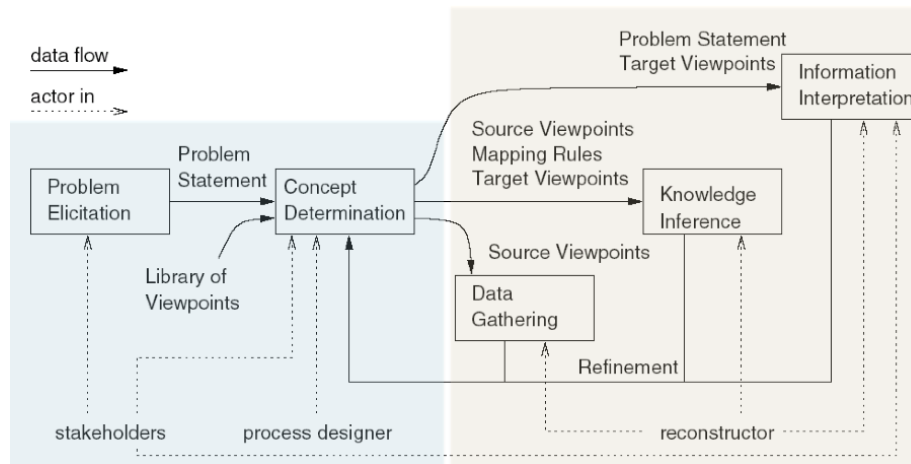

3.1.4 Execution: Knowledge inference

- Going from **source** to target **views**
- Abstracting low-level information

3.1.5   Execution: Information interpretation

- Visual representation
- Analysis, creating new documentation

3.2   Data Gathering: Interactive Case Study

Example: Google Collab with Basic Data Gathering

Or, *why source viewpoints are not necessarily architectural?*

4 Individual Assignment

4.1 Goal

- **Recover the architecture of an existing system**

- Document the outcome in an **individual report**

  - the target reader is a developer, who needs to take over that system and maintain it
  - brief (not more than 3 – 5 pages)
  - do not explain to us what Symphony does in the report; you assume it's done
  - focus on your results

4.2 Case-Study Systems

1. The Zeeguu Project
   - zeeguu.org (invite code: zeeguu-preview)
   - Code:
     - Python Backend: Zeeguu-API
     - React Frontend: Zeeguu-Web
   - A paper about the system

or,

2. Another system that you know
   - if it has comparable complexity (>200 files)
   - you confirm with me about the appropriateness of the system

4.3 Viewpoints

1. Module Viewpoint (**default**)
   - we will write example code snippets in collab to support this
   - makes the most sense for the Zeeguu system
2. Other Viewpoints
   - you could look at the execution or deployment information
   - might make more sense for another system - the Zeeguu one is too simple (could be done together with the module)

4.4 Tools

- Are important for recovery

- **If you can program**, then this is your chance to be coding **analysis tools** over the upcoming lectures - you can still code as a team! you only have to write the analysis on your own - you don't have to use collab

- **If you can't program**, then you'll have to find third party tools (the time the programming ones spend on programming, you'll be spending on finding third party tools)

5   For Next Week

5.1   Reading

- Symphony: View-Driven Software Architecture Reconstruction

5.2   Individual Project

- Start looking for a case study that you would like to analyze

5.3   Practice

- Google Collab with Basic Data Gathering
  - Understand the code
  - Apply it on your own case study if you already have one
  - Can you complete the implementation of the import extractor with the missing part?
    * (consider using an AST parser to extract more precise dependencies)
    * what if an import is not at the beginning of the line?

5.4   Questions & Feedback

- Use the anonymous form
- Or on Discord if it's of general interest

5.5   Further Reference

- Demeyer et al., Object Oriented Reengineering Patterns (Chapter 1.2)