WP21

Speedrun

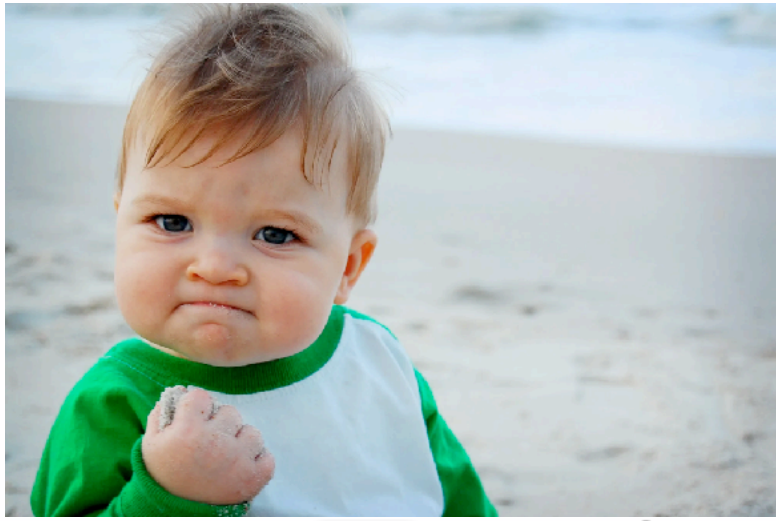
Design of Software Architectures

Dr. Vadim Zaytsev aka @grammarware, September..October 2022

UNIVERSITY OF TWENTE.

DoSA goals:

- understand what software architecture is
- understand when software architecture is needed
- understand why software architecture is needed
- have a conceptual framework for software architects
- be able to explain the above to others
- (know the rest)



the rest in https://osiris.utwente.nl/student/OnderwijsCatalogusZoekCursus.do

Are goals reached?

- Motivate why SA is important for developing quality software.
- Apply concern analysis.
- Set up an architecture description and the tasks to achieve it.
- Apply the process for creating software architecture.
- Apply architectural patterns in the design of an architecture.
- Analyse SAs using scenario-based analysis techniques.
- Apply these techniques & methods for any software design problem.

https://osiris.utwente.nl/student/OnderwijsCatalogusZoekCursus.do

Software Architecture

What is it about?

- various components and relations among them
- architecture patterns
- data flow management
- use cases
- requirements
- business logic
- non-functional reqs

Why do you need it?

- to ease the development
- maintainability, ensure it
- to connect CEOs to devs
- communication is easier
- ensure robustness
- cost evaluation
- risk evaluation
- to solve a real life problem correctly

(filled in on 7 September 2022)

INTERNATIONAL STANDARD **ISO/IEC/ IEEE 42010**

First edition
2011-12-01

Systems and software engineering — Architecture description

Ingénierie des systèmes et des logiciels — Description de l'architecture

Reference number
ISO/IEC/IEEE 42010:2011(E)

INTERNATIONAL STANDARD **ISO/IEC 25010**

First edition
2011-03-01

Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models

Ingénierie des systèmes et du logiciel — Exigences de qualité et évaluation des systèmes et du logiciel (SQuaRE) — Modèles de qualité du système et du logiciel

Reference number
ISO/IEC 25010:2011(E)

Volere

Requirements Specification Template

Edition 20—2020

by James Robertson & Suzanne Robertson

principals of the Atlantic Systems Guild

The Volere Requirements Specification Template is intended for use as a basis for discovering and communicating your requirements. The template provides sections for each of the requirements types appropriate to today's software systems. You may download the template from the Volere site and adapt it to your requirements process and requirements tool. The template is process independent and can be used by Agile, Traditional, and Outsourced projects. The template can be used with any combination of automated tools you are using see http://www.volere.co.uk/tools.htm for a summary of available tools.

An Introduction to Software Architecture

David Garlan and Mary Shaw
January 1994

CMU-CS-94-166

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Also published as "An Introduction to Software Architecture," *Advances in Software Engineering and Knowledge Engineering, Volume I*, edited by V.Ambriola and G.Tortora, World Scientific Publishing Company, New Jersey, 1993.

Also appears as CMU Software Engineering Institute Technical Report CMU/SEI-94-TR-21, ESC-TR-94-21.

This work was funded in part by the Department of Defense Advanced Research Project Agency under grant MDA972-92-J-1002, by National Science Foundation Grants CCR-9109469 and CCR-9112880, and by a grant from Siemens Corporate Research. It was also funded in part by the Carnegie Mellon University School of Computer Science and Software Engineering Institute (which is sponsored by the U.S. Department of Defense). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government, the Department of Defense, the National Science Foundation, Siemens Corporation, or Carnegie Mellon University.

Keywords: Software architecture, software design, software engineering

Defining Execution Viewpoints for a Large and Complex Software-Intensive System

Trosky B. Callo Arias[1], Pierre America[2], and Paris Avgeriou[1]
[1]*Department of Mathematics and Computing Science - University of Groningen*
[2]*Philips Research and Embedded Systems Institute*
The Netherlands
trosky@cs.rug.nl, pierre.america@philips.com, paris@cs.rug.nl

Abstract

An execution view is an important asset for developing large and complex systems. An execution view helps practitioners to describe, analyze, and communicate what a software system does at runtime and how it does it. In this paper, we present an approach to define execution viewpoints for an existing large and complex software-intensive system. This definition approach enables the customization and extension of a set of predefined viewpoints to address the requirements of a specific development organization. The application of this approach has helped us to identify a set of execution viewpoints that we are currently using to construct execution views of an MRI system, a large software-intensive system in the healthcare domain.

1. Introduction

The usage of multiple views is a common practice to construct and document the architecture of large software-intensive systems [4, 8]. The ISO/IEC 42010 standard provides a widely accepted conceptual definition of architectural views, viewpoints and models [8]:

- *An architectural view* is a representation of a set of system elements and relations associated with them, conforming to a specific viewpoint.
- *An architectural viewpoint* addresses particular concerns of the system stakeholders and consists of the conventions for the construction, interpretation, and use of an architectural view.
- A view may consist of one or more *architectural models*. Each such architectural model is developed using the conventions and methods established by its associated viewpoint. An architectural model may participate in more than one view.

In this paper, we focus on the stakeholder concerns related to system evolvability and the corresponding views that can address them. As part of our research on the evolvability of large software-intensive systems [16], we observed that suitable architectural views are important assets to facilitate system evolution [11, 12]. Such views help practitioners to understand the existing system, to plan and evaluate intended changes, and to communicate them to others.

In particular, we are interested in *execution views*, which consist of a set of models that describe and document *what a software system does at runtime and how it does it*. The term runtime refers to the actual time that the software system is functioning (during testing or in the field). Obviously, it is very important to understand this runtime behavior of the software, but in practice documenting it often does not receive enough attention. Thus, our particular focus is to support practitioners in how to construct execution views for large and complex software-intensive systems. Such systems often have a heterogeneous implementation and consist of multiple processes, each with multiple threads, deployed across several computers.

In our initial work, we constructed an execution view of an existing large software system [2], which addressed specific stakeholder concerns. However, a development organization of such a large and complex system has several stakeholders with numerous concerns. Therefore, the organization needs to be able to define a number of execution viewpoints addressing the needs and matching the characteristics of its particular system. To achieve this, an organization may either reuse the predefined viewpoints available in the literature (e.g. [3, 5, 11, 14]) or define new ones.

Paper published in IEEE Software 12 (6)
November 1995, pp. 42-50

Architectural Blueprints—The "4+1" View Model of Software Architecture

Philippe Kruchten
Rational Software Corp.

Abstract

This article presents a model for describing the architecture of software-intensive systems, based on the use of multiple, concurrent views. This use of multiple views allows to address separately the concerns of the various 'stakeholders' of the architecture: end-user, developers, systems engineers, project managers, etc., and to handle separately the functional and non functional requirements. Each of the five views is described, together with a notation to capture it. The views are designed using an architecture-centered, scenario-driven, iterative development process.
Keywords: software architecture, view, object-oriented design, software development process

Introduction

We all have seen many books and articles where one diagram attempts to capture the gist of the architecture of a system. But looking carefully at the set of boxes and arrows shown on these diagrams, it becomes clear that their authors have struggled hard to represent more on one blueprint than it can actually express. Are the boxes representing running programs? Or chunks of source code? Or physical computers? Or merely logical groupings of functionality? Are the arrows representing compilation dependencies? Or control flows? Or data flows? Usually it is a bit of everything. Does an architecture need a single architectural style? Sometimes the architecture of the software suffers scars from a system design that went too far into prematurely partitioning the software, or from an over-emphasis on one aspect of software development: data engineering, or run-time efficiency, or development strategy and team organization. Often also the architecture does not address the concerns of all its "customers" (or "stakeholders" as they are called at USC). This problem has been noted by several authors: Garlan & Shaw[1], Abowd & Allen at CMU, Clements at the SEI. As a remedy, we propose to organize the description of a software architecture using several concurrent *views*, each one addressing one specific set of concerns.

An Architectural Model

Software architecture deals with the design and implementation of the high-level structure of the software. It is the result of assembling a certain number of architectural elements in some well-chosen forms to satisfy the major functionality and performance requirements of the system, as well as some other, non-functional requirements such as reliability, scalability, portability, and availability. Perry and Wolfe put it very nicely in this formula2, modified by Boehm:

Software architecture = {Elements, Forms, Rationale/Constraints}

Software architecture deals with abstraction, with decomposition and composition, with style and esthetics.

Technical Report
CMU/SEI-95-TR-021
ESC-TR-95-021
December 1995

Quality Attributes

Mario Barbacci
Thomas H. Longstaff
Mark H. Klein
Charles B. Weinstock

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

From [GarlanS94]

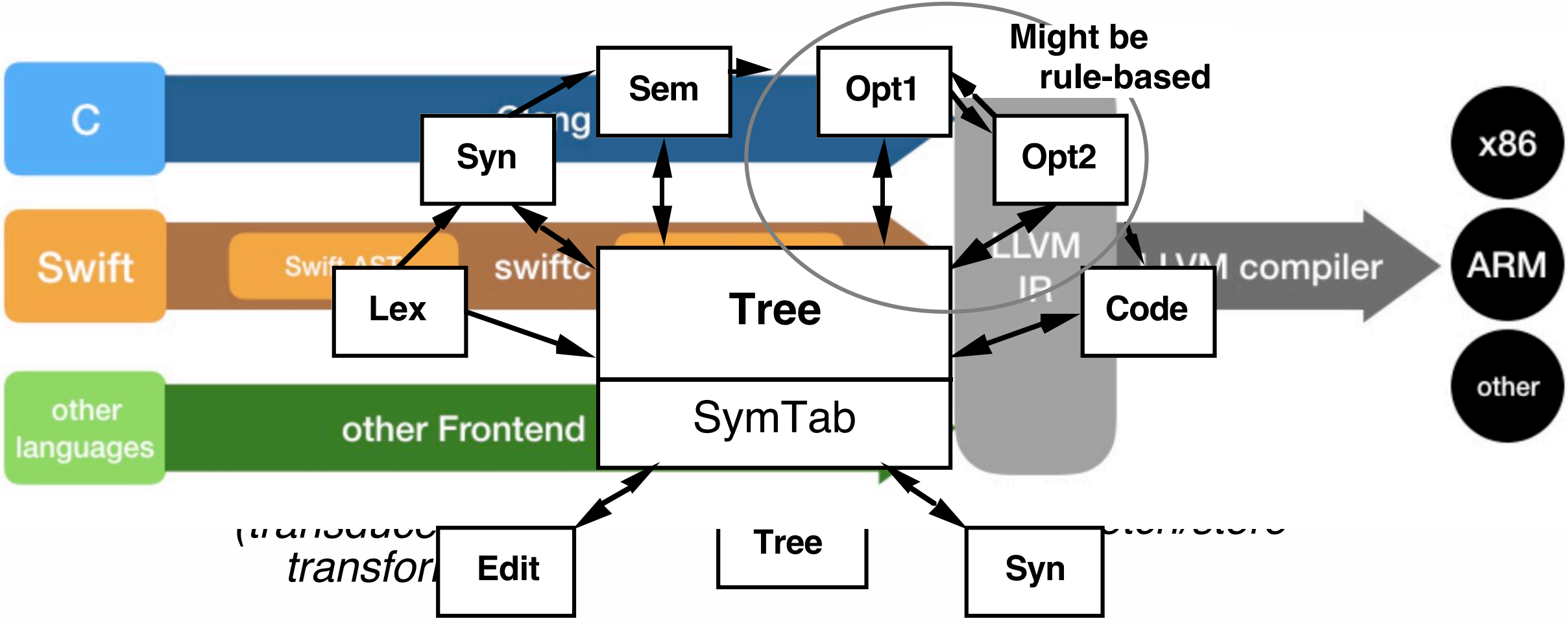


Figure 17: Modern Canonical Compiler

Figure 18: Canonical Compiler, Revisited

https://www.omnisci.com/technical-glossary/llvm

From [AriasAA09]

Table 1. Predefined viewpoints for execution views

Viewpoint	What it describes (concern)	System elements
Concurrency [14]	- Task structure and mapping of functional elements to tasks - Inter-process communication and state management - Synchronization and integrity - Startup, shutdown, task failure, and reentrancy	Processes, process groups, threads, inter-process communication
Behavior description [3]	- Types of communication - Constraints on ordering - Clock-triggered stimulation	Use cases, structural elements, processes, states, applications, and objects.
Deployment [14]	- Hardware required (specification and quantity) - Third-party software requirements and technology compatibility - Network requirements and capacity and physical constrains	Processing and client nodes, network links, hardware components, and processes.
Deployment style [3]	- Allocation, migration, and copy relations between software elements and computing hardware. - Properties of computing hardware, e.g., bandwidth, and resource consumption.	Software elements (processes) and computing hardware (processor, memory, disk, etc.)
Execution architecture [5]	- Execution configuration and its mapping to hardware devices - Dynamic behavior of configuration - Communication protocol - Description of runtime entities and their instances	Processes, tasks, threads, clients, servers, buffers, message queues, and classes

From [Kruchten95]

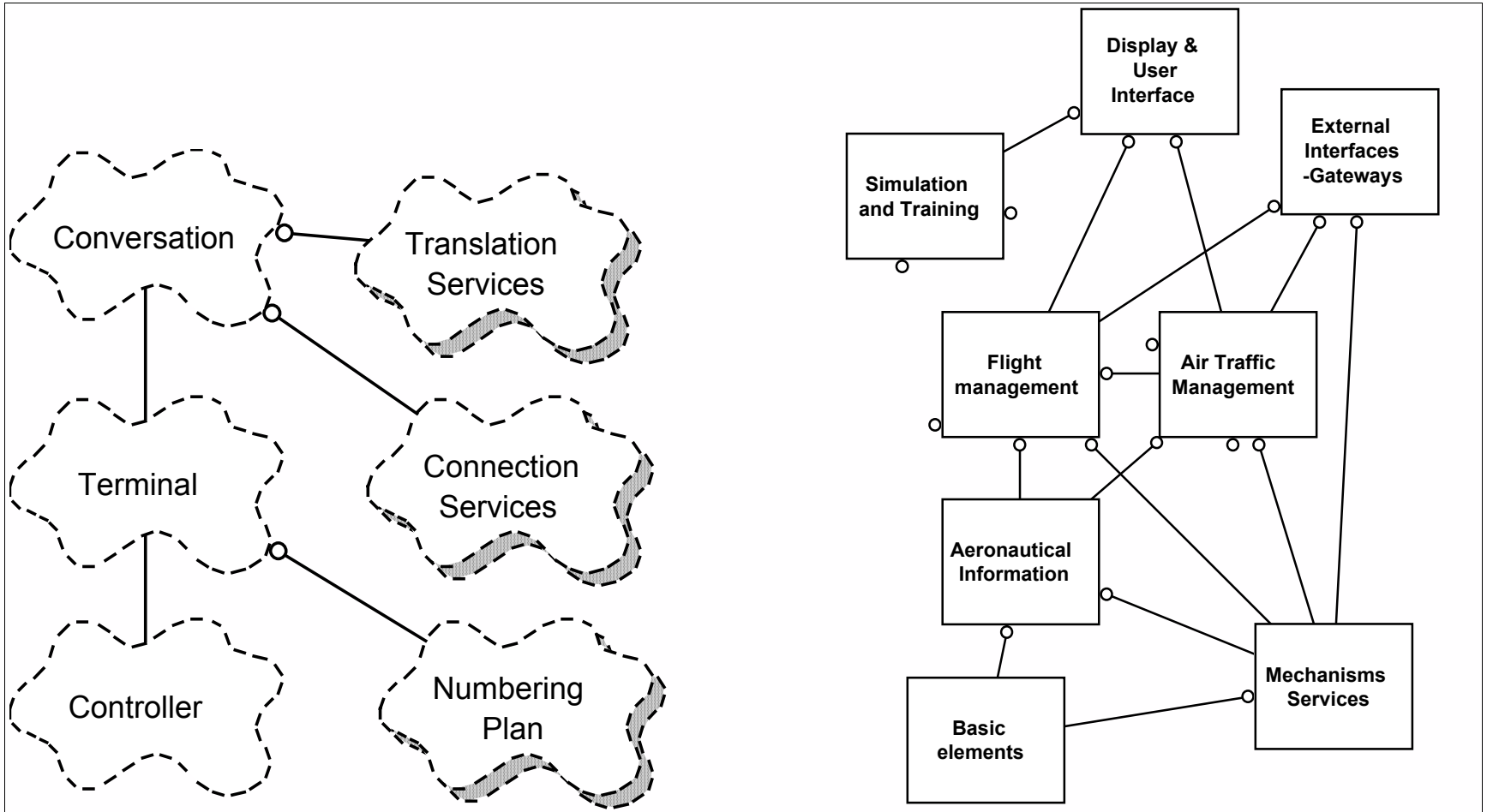
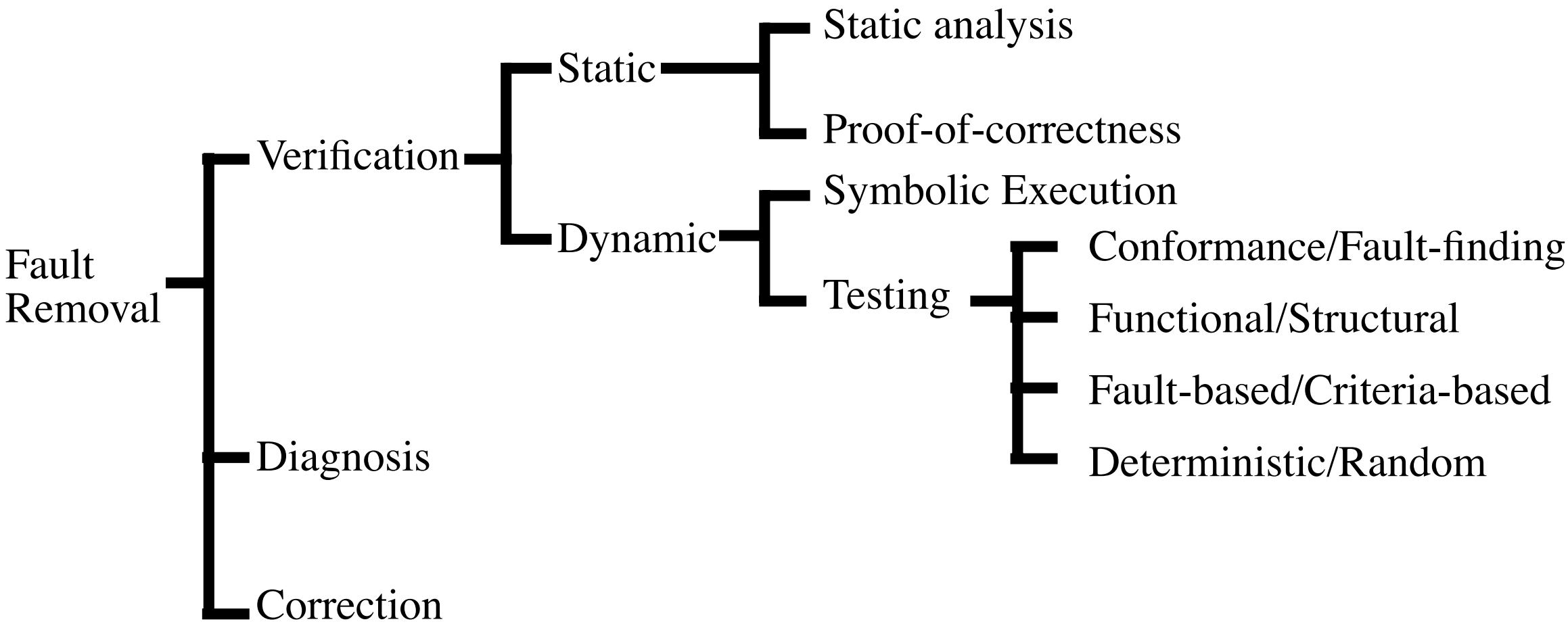


Figure 3— a. Logical blueprint for the Télic PABX . b. Blueprint for an Air Traffic Control System

From [BarbacciLKW95]



What is Important in Software in Autonomous Cars?

- reliability (car not crashing, not killing other people, reaching the destination)
- navigation
- what the hardware can do
- the breaking system
- control over acceleration
- weather systems
- latency & performance
- recoverability after crashing
- security / hackability
- sitting in the car and talking, ease of use
- error protection
- avoiding liability
- redundancy
- should be maintainable
- (in)compatibility across vendors
- indefinite support

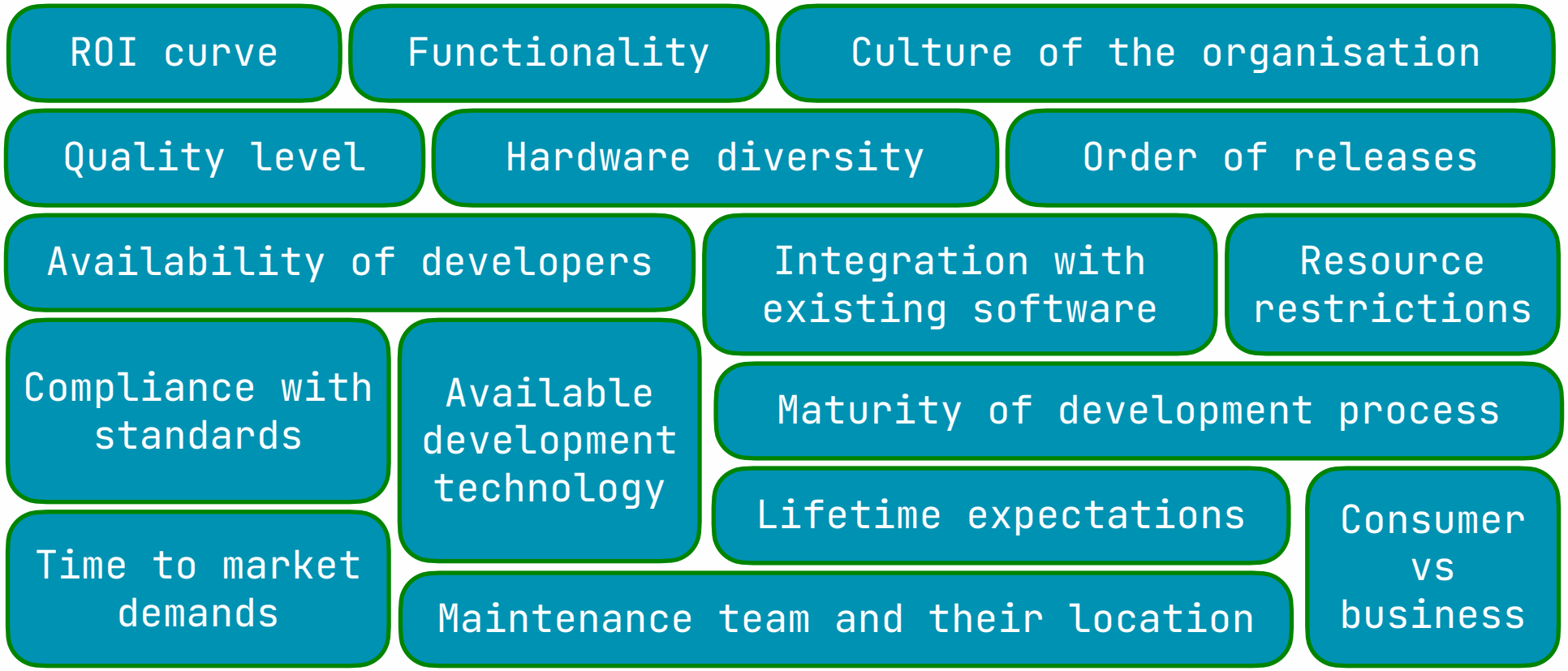
(filled in on 7 September 2022)

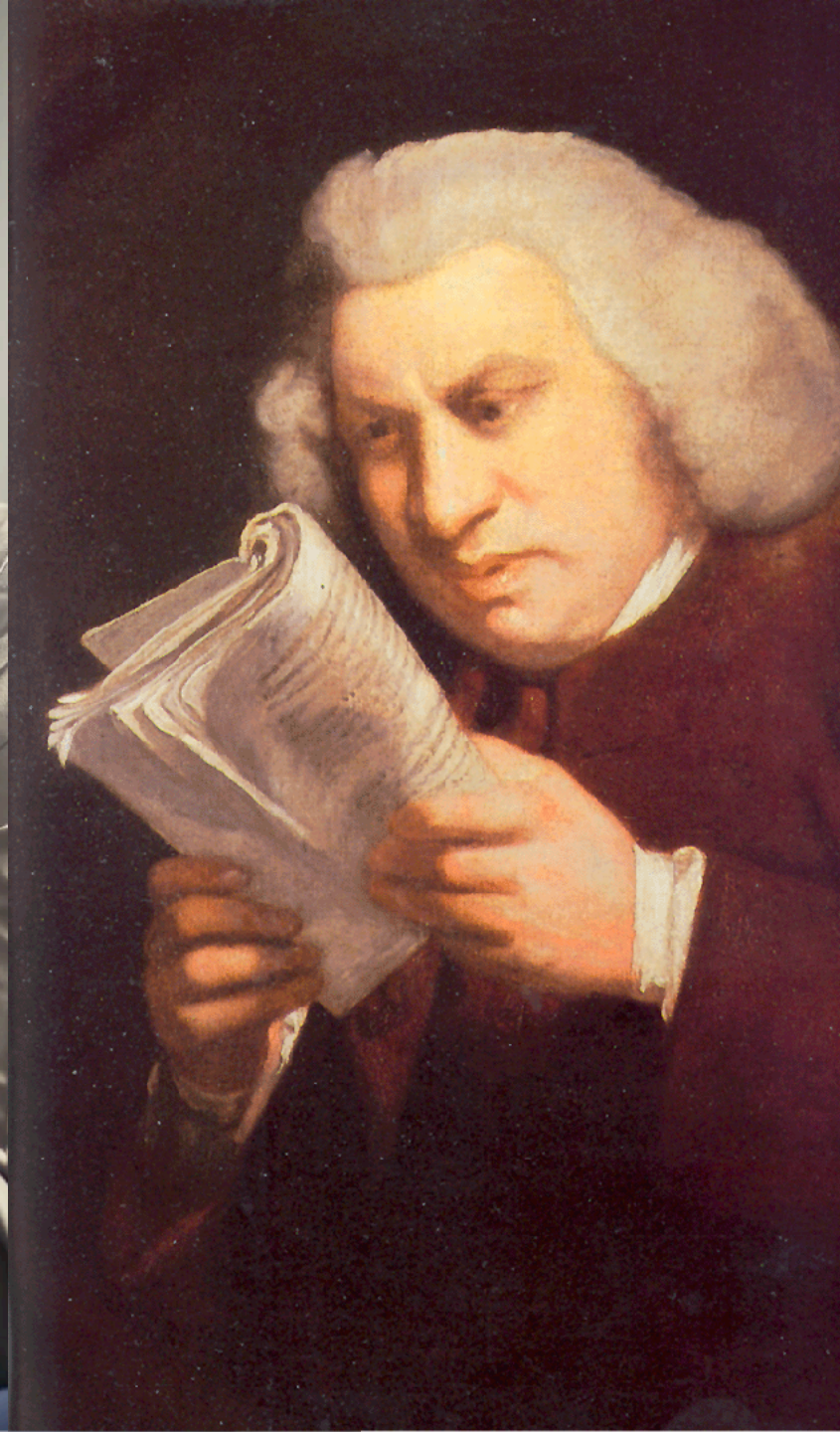
What is important in software?

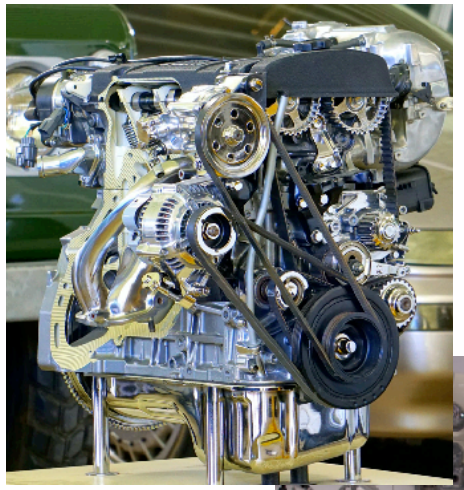
- planning
- simplicity
- something that a stakeholder wants
- it should do what it is meant to do
- performance
- adaptability
- security
- maintainability
- usability
- feasibility
- interface
- documentation
- testing
- integration
- coding standards

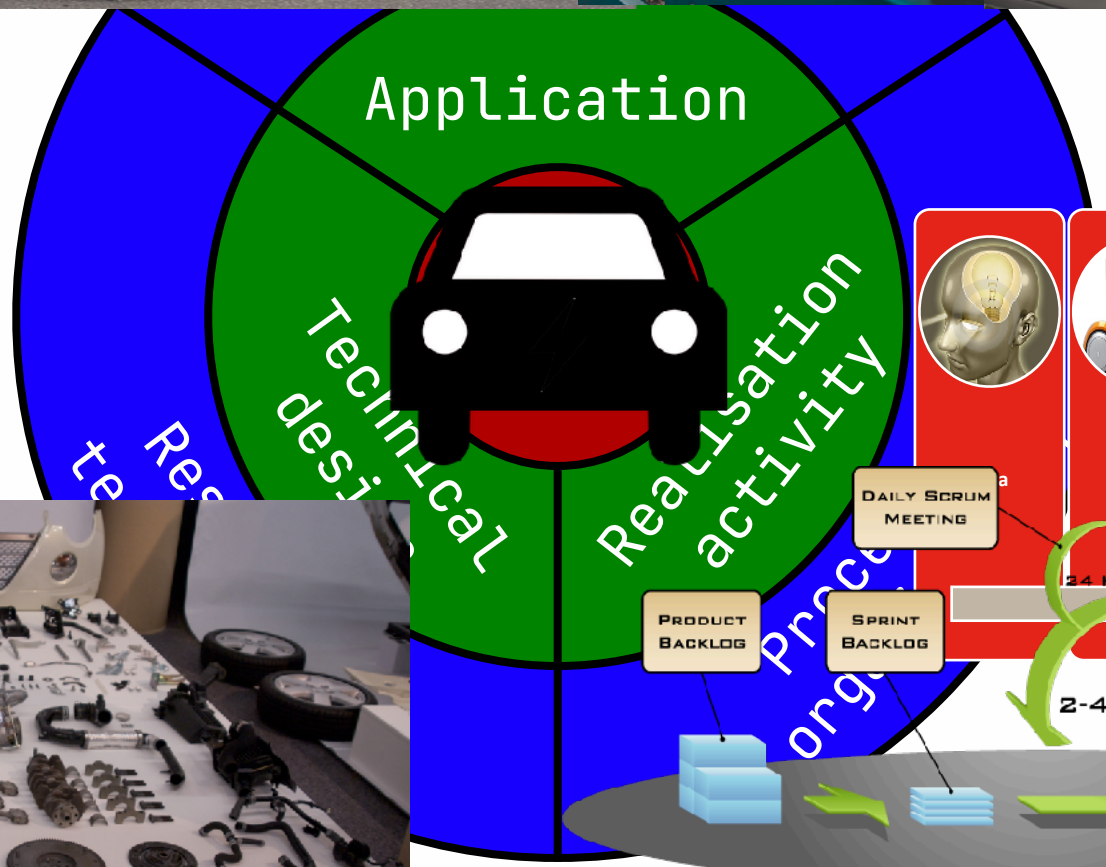
(filled in on 7 September 2022)

What is important in software?





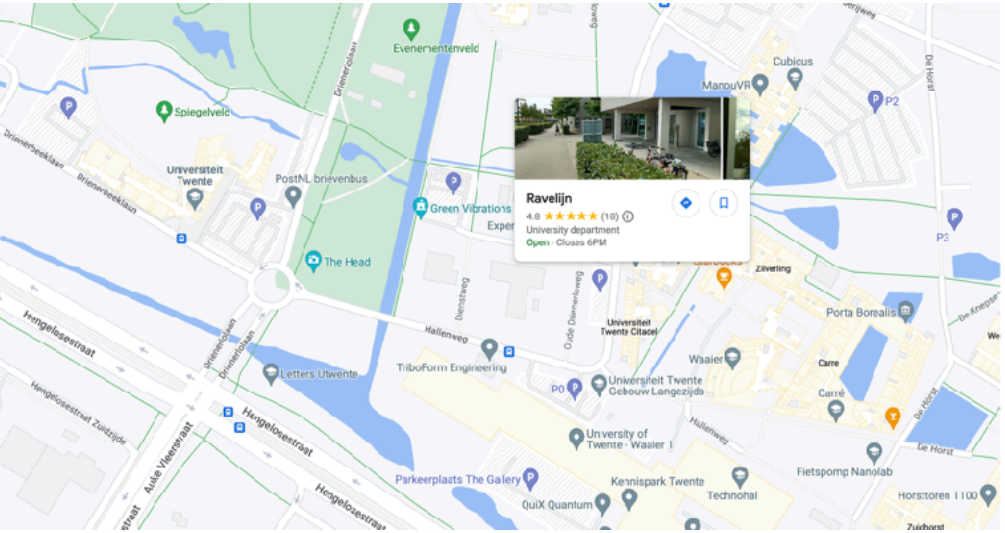




UNIVERSITY OF TWENTE.

Direction in architecture

- The software architecture of a system is
  - a collection of statements
  - that gives directions
- A **direction** can be given:

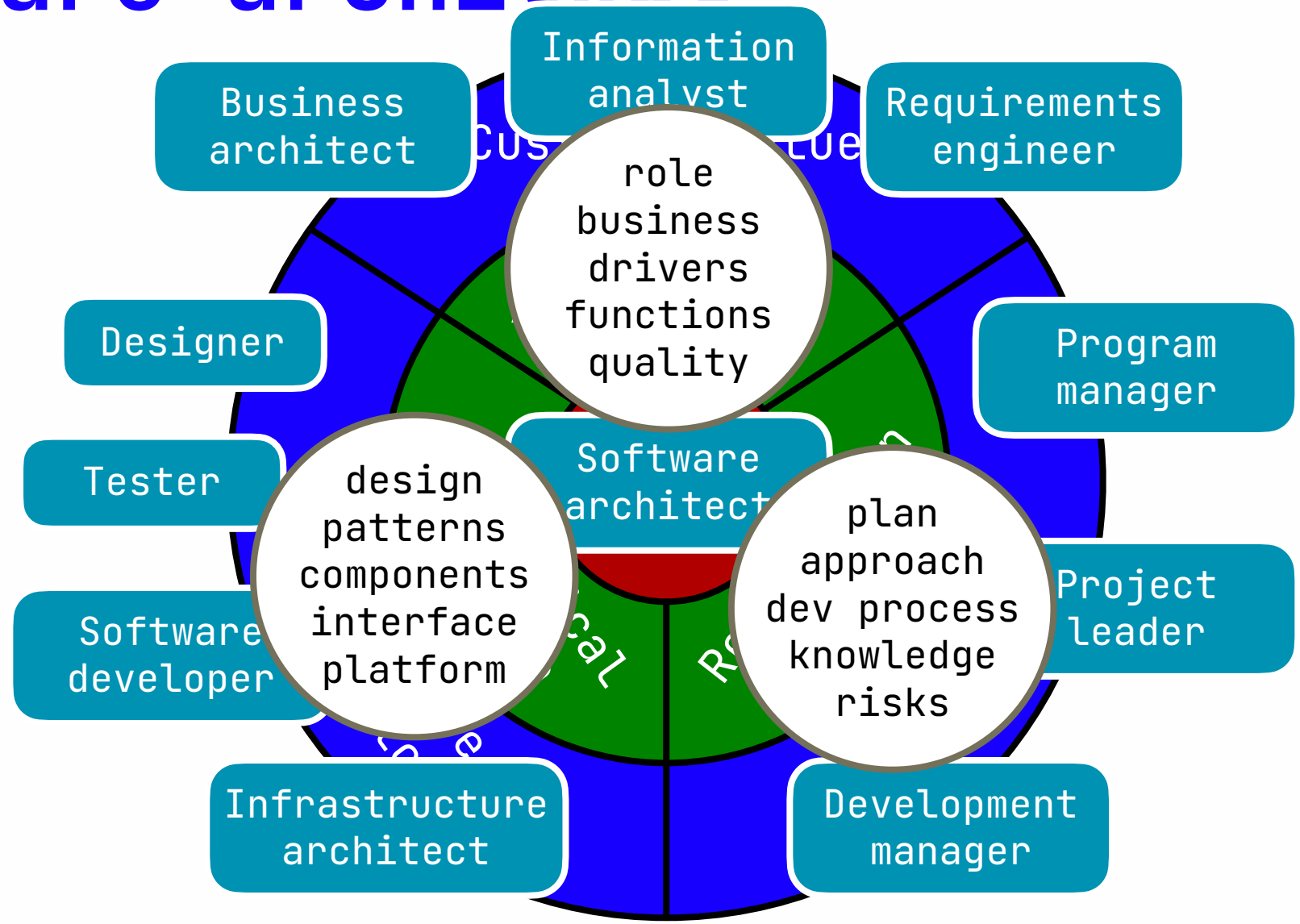


UNIVERSITY OF TWENTE.

Good architecture is…

- consistent
- unambiguous
- descriptive
- encompasses the entire design
- structured
- suitable for its purpose
- implementable
- clear
- embedded in its environment
- understandable
- future proof
- solving real life problems
- editable
- maintainable
- correct
- communicated
- etc

Software architect



What do you do with stakeholders?

- prioritise
- ask about their interests
- examine their concerns
- address their concerns
- satisfy
- inform
- ask for feedback
- search for information
- keep informed
- check project’s realisability
- keep expectations in check
- ignore
- resist
- terminate collaboration

(filled in on 16 September 2022)

Stakeholder analysis

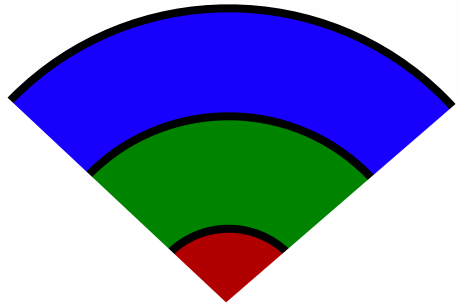
Communication matrix

Who	Stakeholder (group/individual)
Why	What do you want to achieve (inform/think/decide)
About what	Topic of the communication (proposal/decision/…) Which view(point)?
When	At which moment (once/periodically/milestones)
With what	Format (presentation/workshop/meeting/email/call)
By whom	Who is doing it (manager/project leader/architect)

How to find usable concerns?

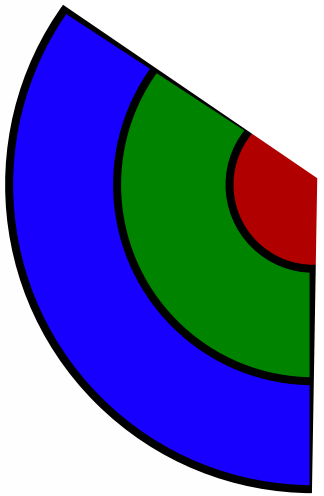
- Formulate concerns from stakeholder perspective (no generic terms!)
  - goals to achieve
  - tasks/activities to perform
  - information needs to satisfy
- Must be usable for trade-offs with other concerns:
  - be clear on importance for the stakeholder
  - prioritise per stakeholder, then overall
  - know which concerns are non-negotiable
- Concerns are the reason behind a requirement
- Cast doubt on concerns defined in system terms:
  - find out what the real concern is

Typical application aspects



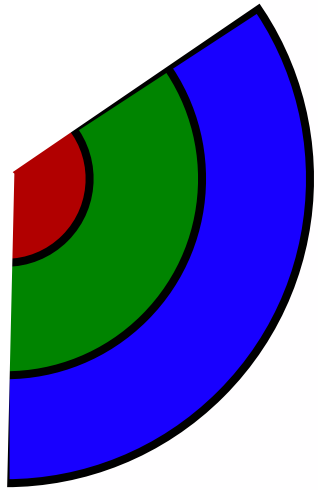
- Legal compliance, actuality, traceability
- Social responsibility, impartiality
- Credibility, transparency, money traceability
- Safety, customer support, energy consumption
- Duration, eco footprint, yield, continuity
- Energy, availability, reliability, maintainability

Typical design aspects



- Decomposition
  - components, patterns
  - data/control flow, interfaces
- Software quality
  - deployment, life cycle, upgrades, lock in
  - resource usage, energy efficiency, time behaviour
  - error handling, data integrity, recovery

Typical realisation aspects



- Waterfall ⟺ Iterative
- Manual ⟺ Automated
- Buy ⟺ Reuse
- Adapt ⟺ Remake
- Feature first ⟺ Market first
- Develop ⟺ Outsource
- Assign ⟺ Hire
- Test for release ⟺ Test for integration

View(point) & model

- Model
  - spec for a part of the world
- View
  - work product
  - represents a system
- Viewpoint
  - explains how to make a view

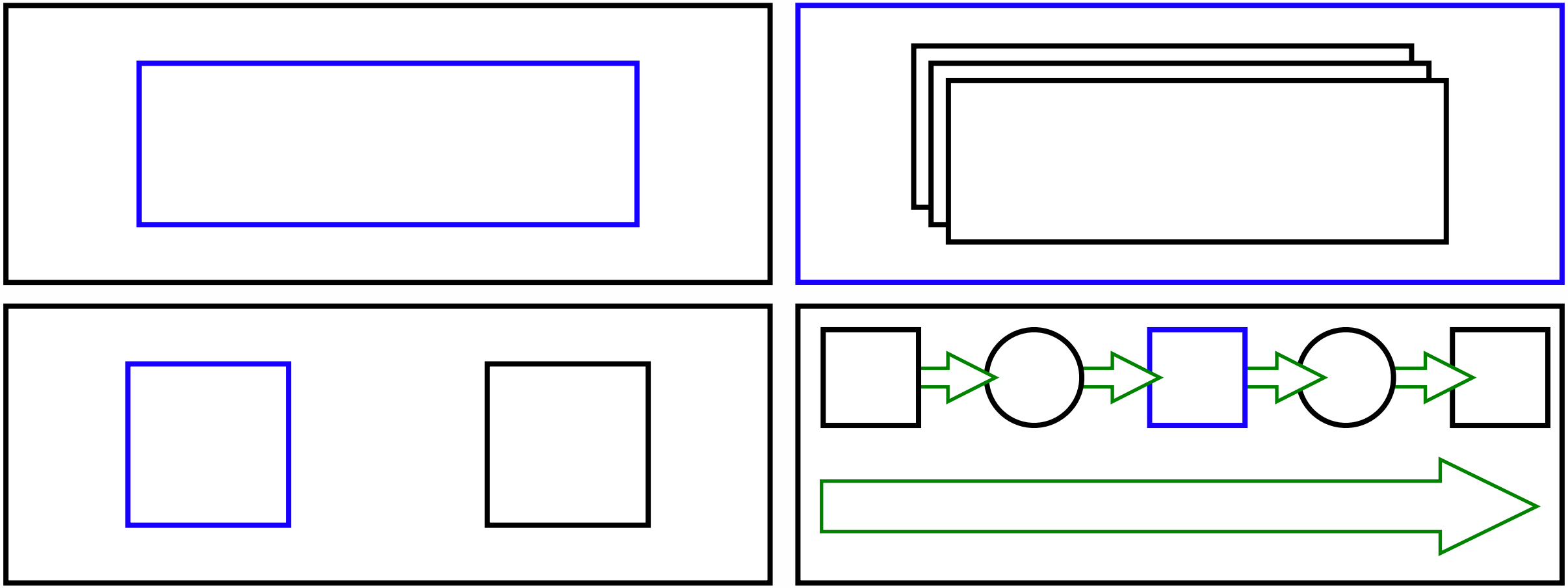

THE EXPERT
7 RED LINES
0:00 / 7:34 (7:25)
HD

Keep track of specifications

Keep track of all the specifications that you use and introduce

- Identification (name, version, source)
- Relation to the architecture:
  - For what architecture element is it used?
  - How does it use the architecture?
- Dependencies on other specifications
  - (graph) specifications are nodes, relations are arrows
  - (table) a row for each specification

How can systems relate?



Trends

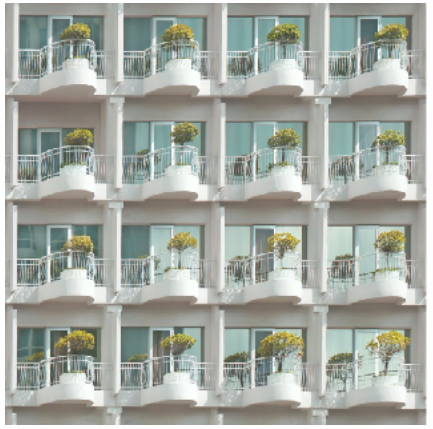
- Real impact!

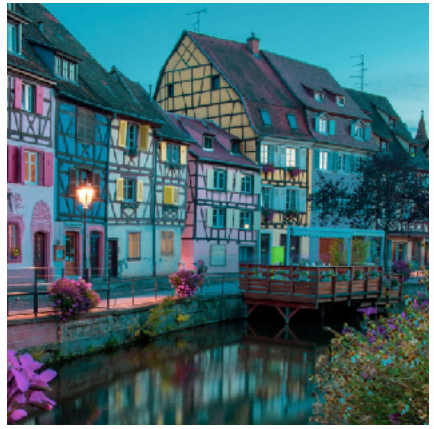
- Find
- Identify
- Choose
- Follow
- Ignore

Consistent ADD

- Don’t forget:
  - rationale
  - target audience
  - filling a template ≠ architecting
  - decision kinds
    - system
    - env
    - boundary

AD should contain…





Realisation

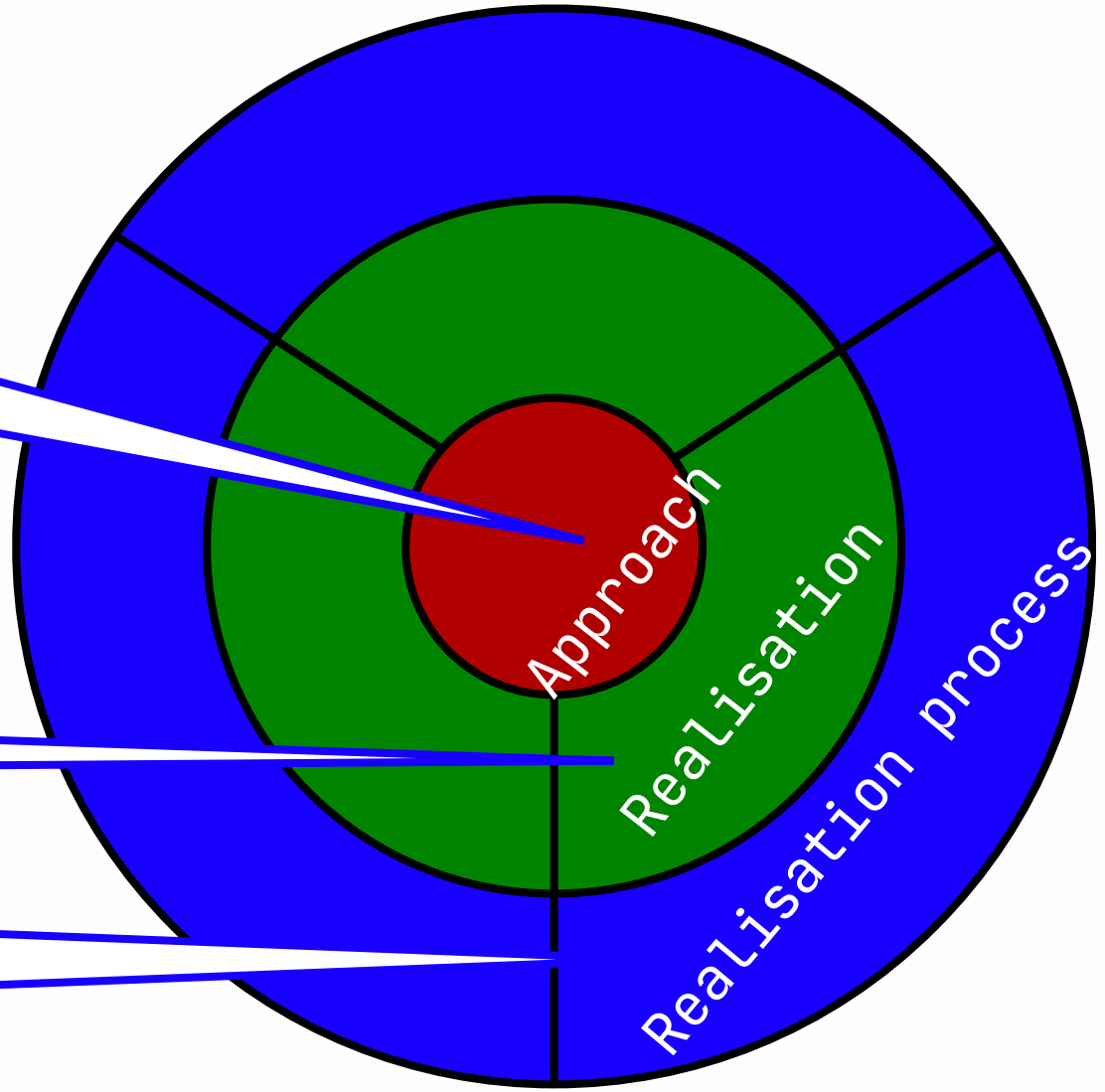
- Think of
  - activities
    - testing, prototyping, negotiating
  - artefacts
    - code, test models, backlog
  - constraints
    - systems, requirements, laws
  - people
    - skills, knowledge, beliefs

Realisation domain

By which activities & what behaviour?
- main tasks
- phases
- milestones

Who does what when?
- planning, roadmap,
- responsibilities,
- work packages

Which organisation/people?
- team structure,
- knowledge/skills,
- education/training



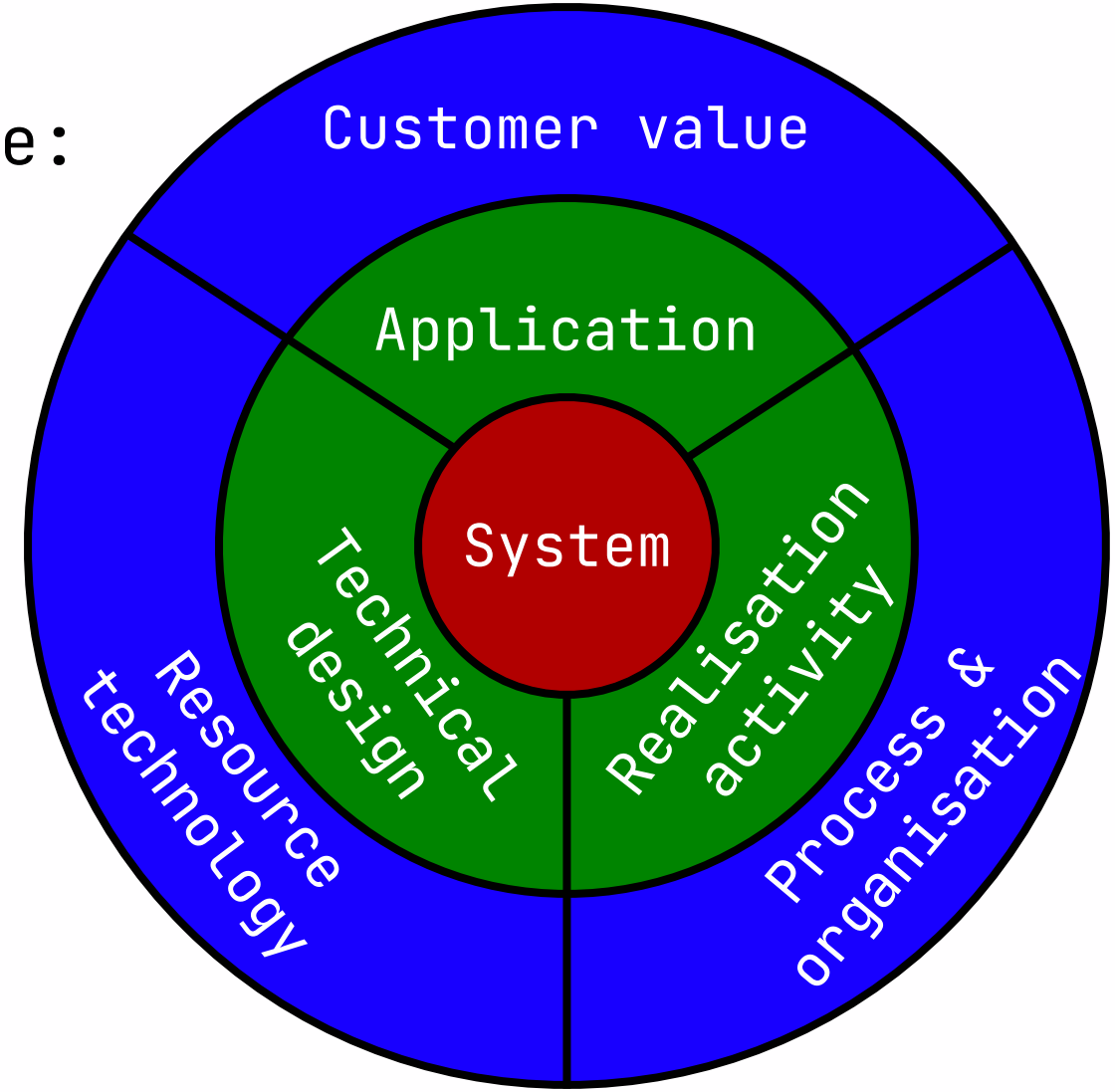
by Robert Deckers, based on ISO 42010 & DYA

Guidelines for giving direction

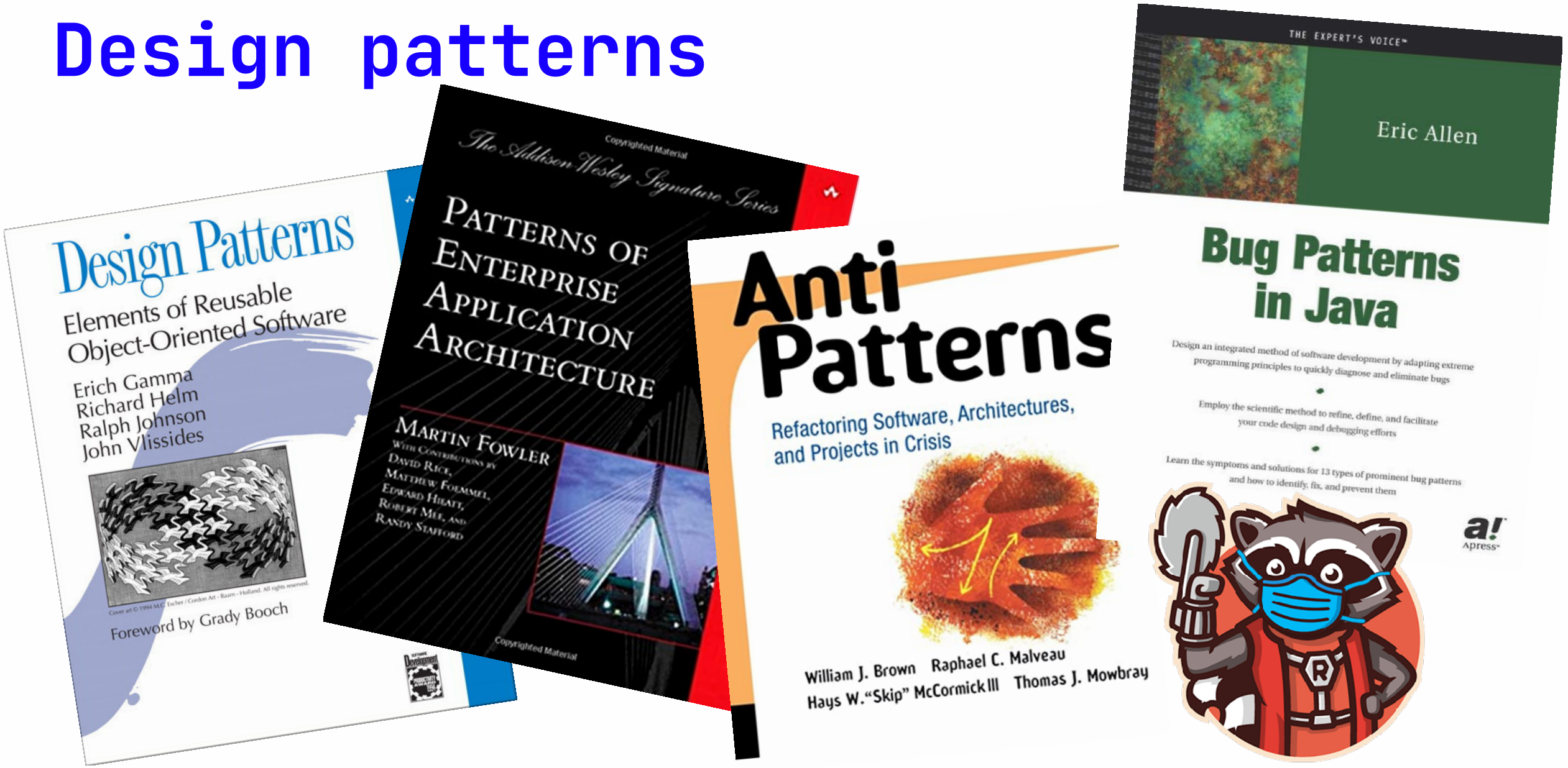
negation is nonsense	generic truths
inconsistent in itself	effortlessly fulfilled
not understandable	impossible to enforce

Dominant decomposition

- Elements shaping the architecture:
  - Functions
  - Components
  - Services
  - Guidelines
  - Expertise
  - Frameworks
  - Events
  - Messages
  - States
  - . . .



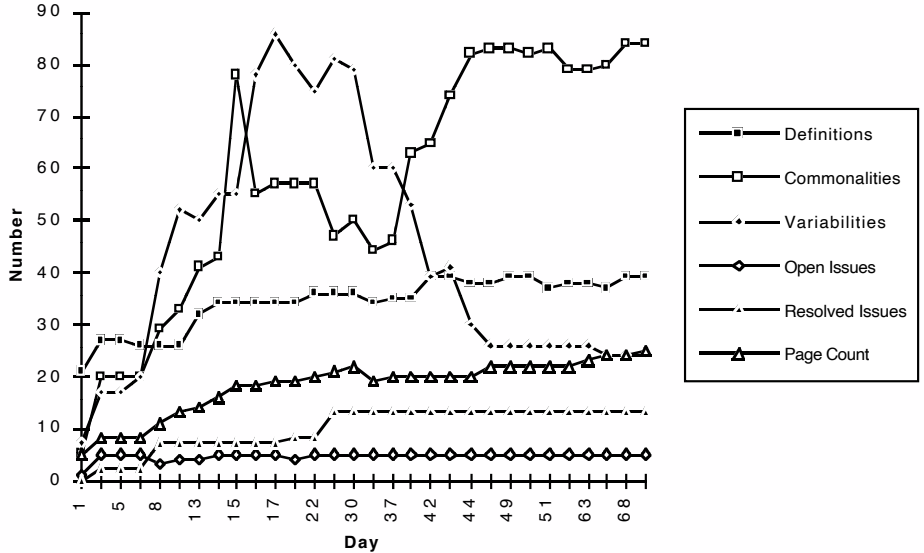
Design patterns



Design Patterns (1994), *Patterns of EAA* (2002), *AntiPatterns* (1998), *Bug Patterns* (2002)

Family architecture

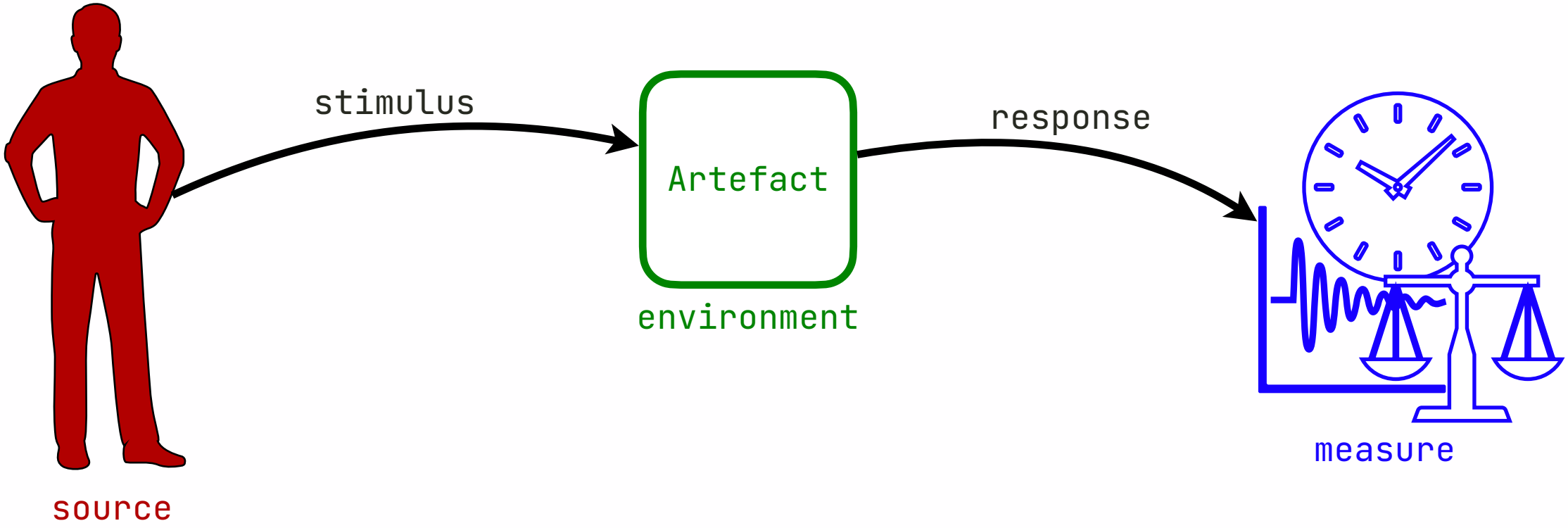
- Lower costs if done right
- Predictable
- Commonalities
- Variabilities
- Directions to achieve



Quality attributes

- **Red flags**
  - speed
  - ease
  - performance
  - bug-free
  - usability
  - . . .
- **Green flags**
  - configure
  - recover
  - analyse
  - replace
  - maintain
  - . . .

QA scenarios



Weichang Du, *Understanding Quality Attributes*, 2009-2010

Role of the software architecture

to provide

solution direction

for

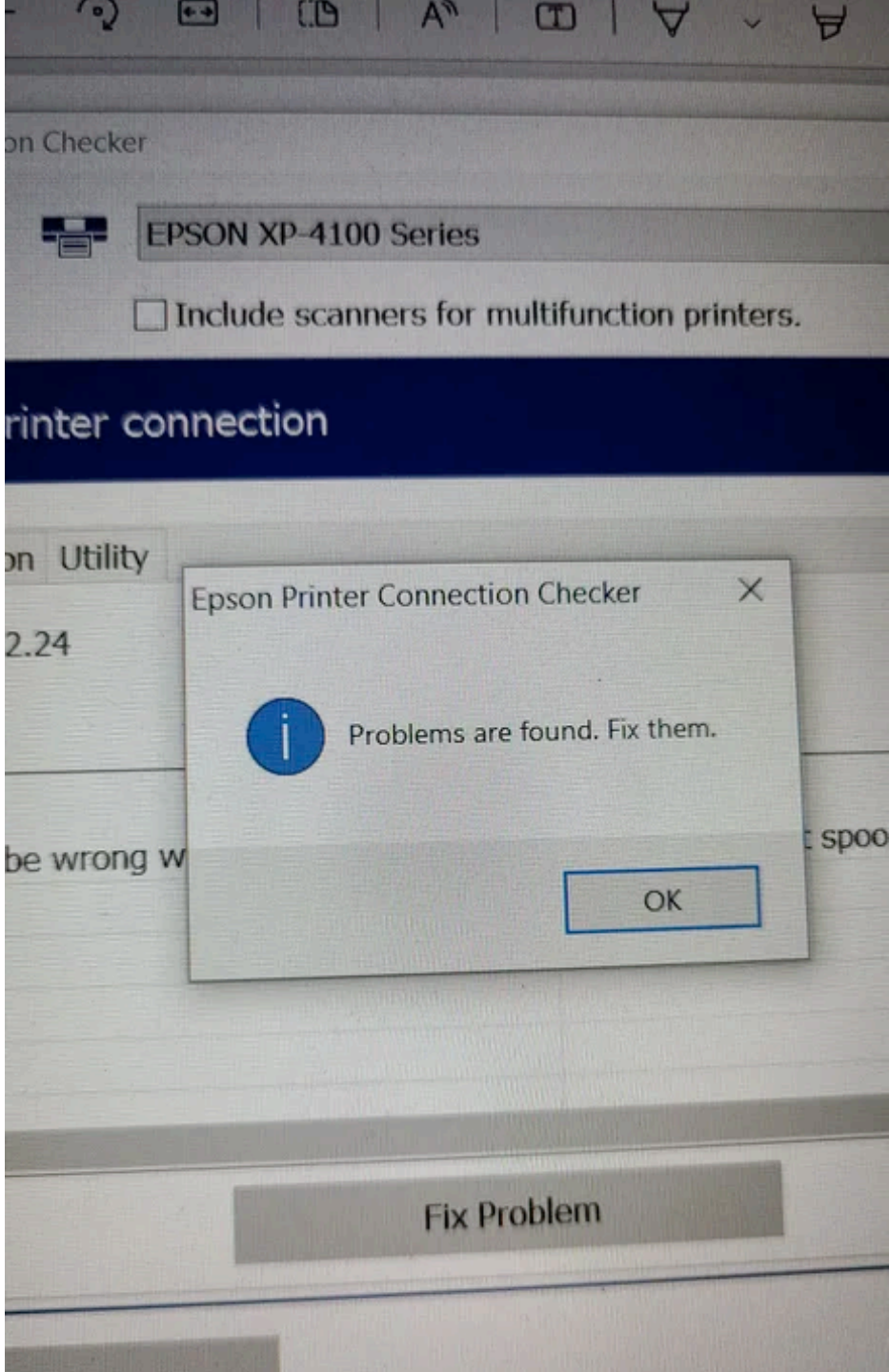
most important properties

that are

the most difficult to realise

Group assignments

- Architecting is a responsibility
  - take decisions
  - provide directions
- Refuse to be bullied
  - stay focused
- Technical writing
  - concise, specific, readable
  - learnable skill



Design of Software Architectures

Dr. Vadim Zaytsev aka @grammarware, September..October 2022

UNIVERSITY OF TWENTE.