HSLU Hochschule Luzern

Software Architecture and Techniques

Evolution Of Software Architecture Over The Last Decades

Marcel Baumann, tangly llc

Truths (1/2)

All architecture is design but not all design is architecture.

Architecture represents the significant design decisions that shape a system, where **significant** is measured by **cost of change**

Grady Booch, 2006

Truths (2/2)

Software development does not have economies of scale.

Development has **diseconomies of scale**.

– Allan Kelly

Architecture Definitions

The fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution.

— *ISO/IEC/IEEE 42010:2011*

The structure of components, their interrelationships, and the principles and guidelines governing their design and evolution over time.

— *The TOGAF Standard, Version 9.2*

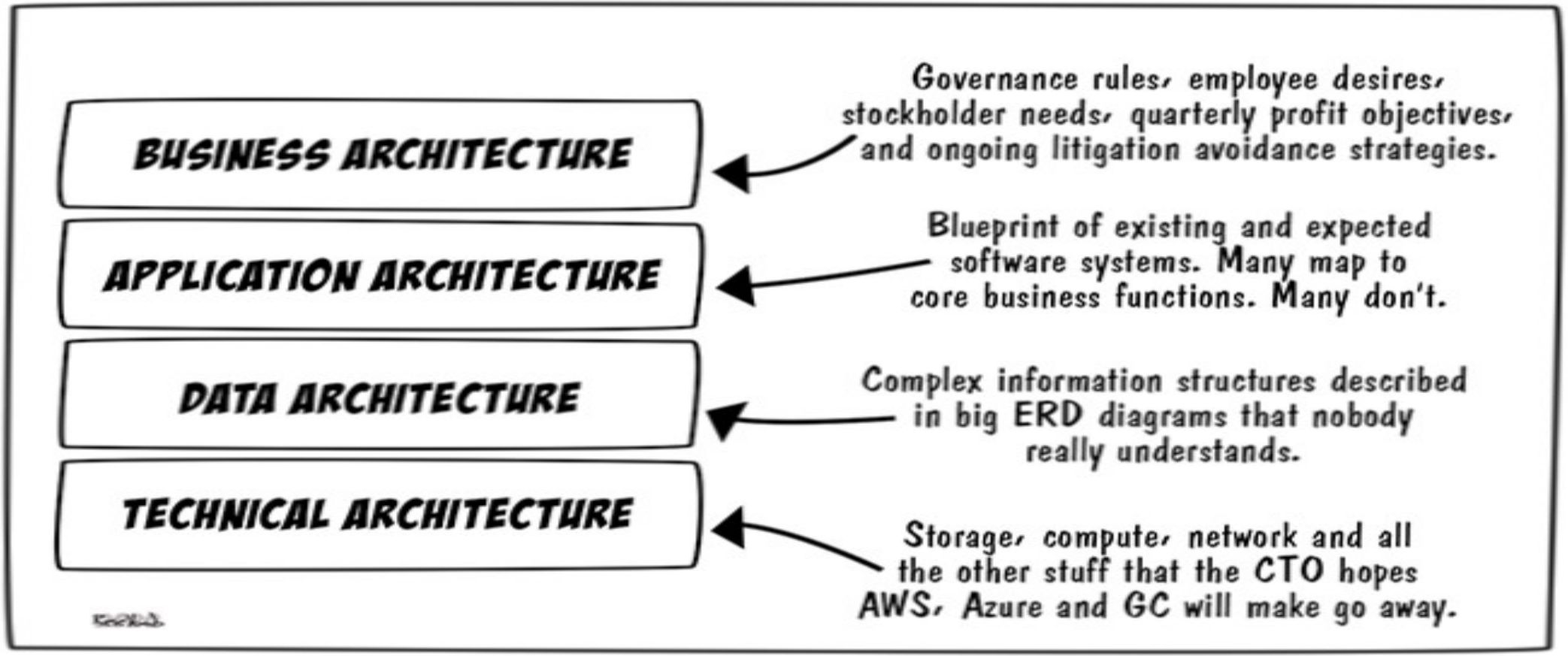
Old School Architect

- Separate position with highest status
- Decide how the architecture will be
  - Architects are smart
  - Developers are dump
- *Ivory tower syndrome*
- *Powerpoint architect syndrome*
- *Think about Conway Law – Hierarchy vs Meritocracy*

Architecture Kinds (1/2)

- Design → developer
- Application Architecture → within team
- Solution Architecture → within product
- Enterprise Architecture → whole company
  every traditional architect wants to be an enterprise architect!

Architecture Kinds (2/2)



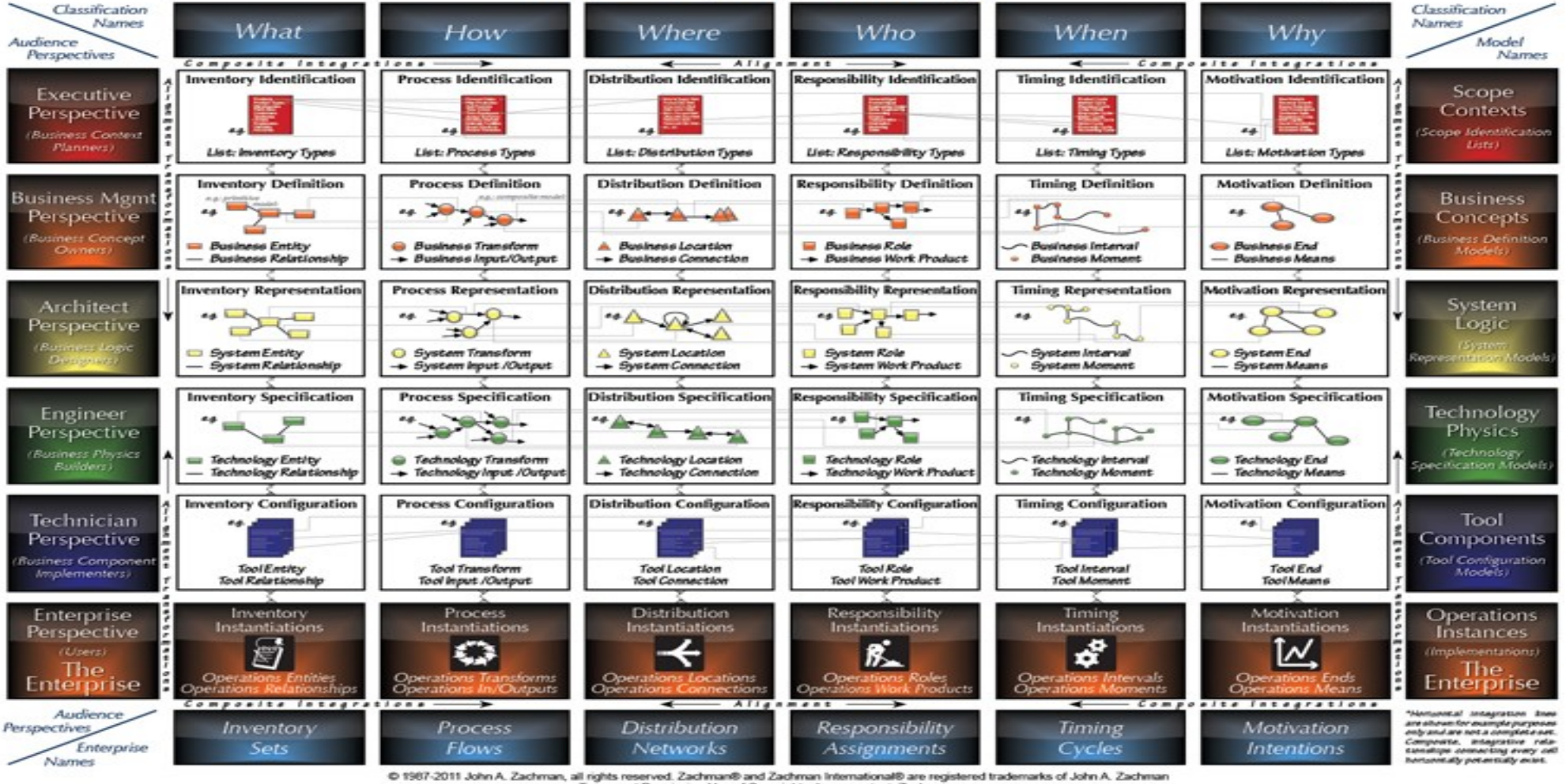
History 1960 - 2000

- Structured Programming – goto are evil -
- Structured Design – Yourdon, DeMarco -
- Structured Analysis & Design – SASD, SADT -
- Object Oriented Approach – Booch, Rumbaugh, Jacobson -
- Enterprise Architecture – Zachmann -

The Zachman Framework for Enterprise Architecture™

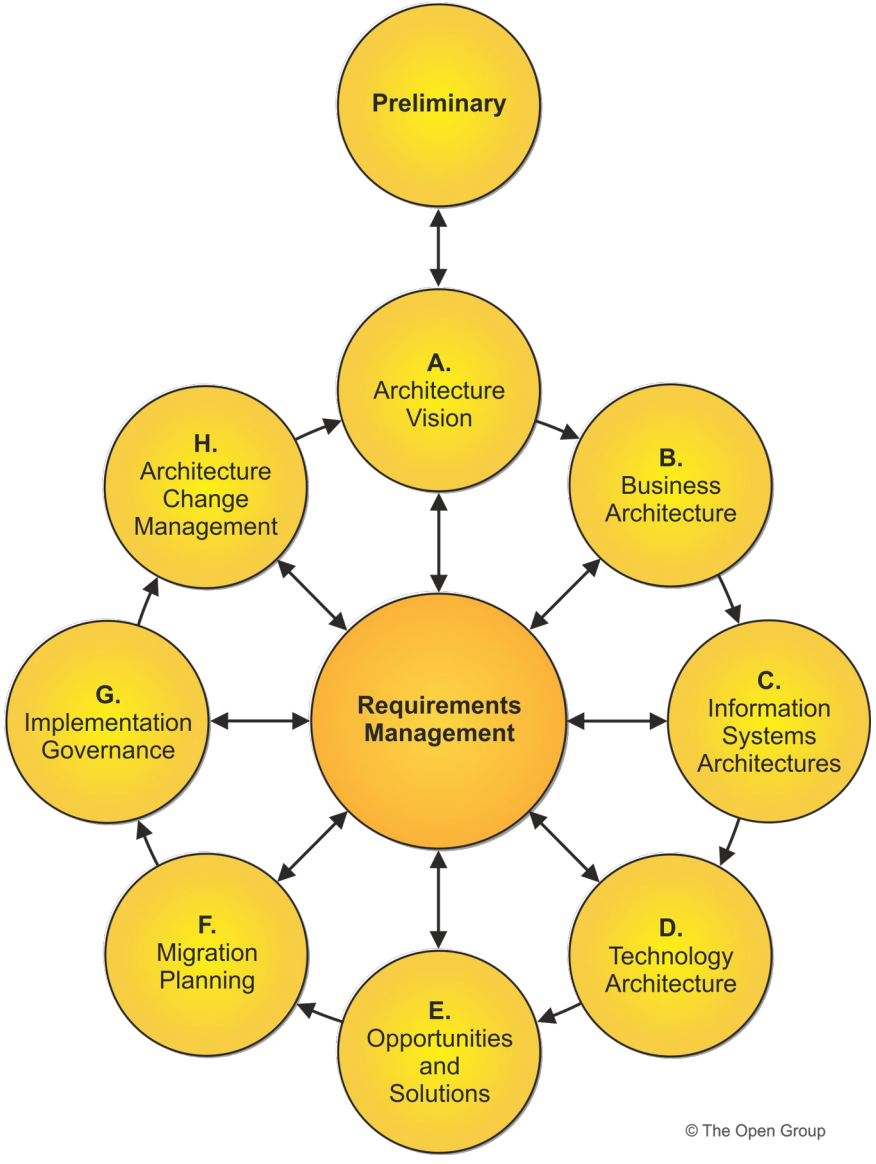
The Enterprise Ontology™

Version 3.0



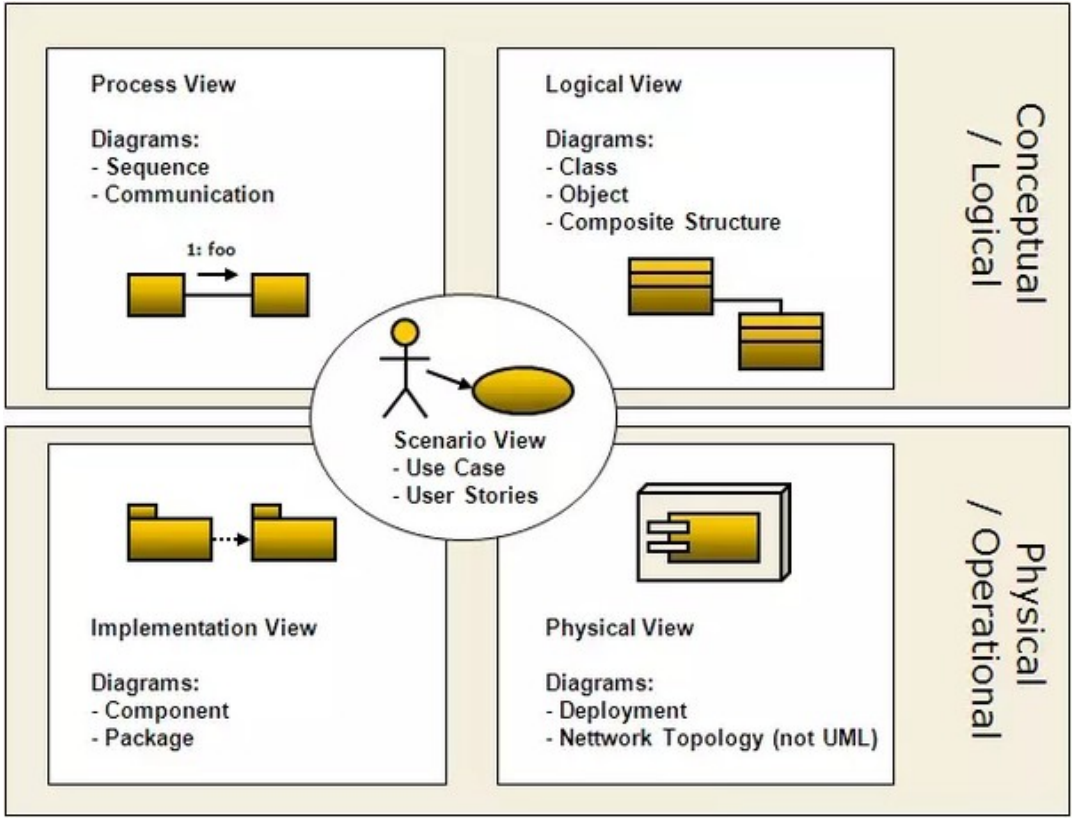
Standards (around 2000)

- TOGAF
- Arc42
- RUP – Inception, Elaboration, Construction, Transition
- Hermes
- IEEE



UML – 4 + 1 View

Evaluate the views in the context of a modern development project and environment

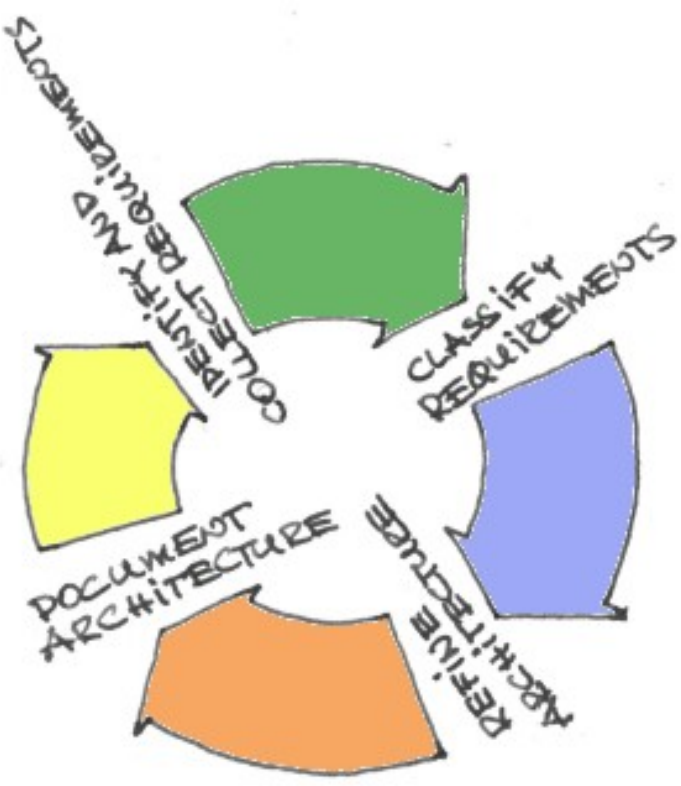


First Findings

- Architecture shall fulfill customer needs
  - Functional requirements
  - Non-functional requirements
- Dichotomy Analysis and Design
  - Analysis is requirement engineering – understand the problem
  - Design is architecture – identify a solution
  - Modern approaches killed up-front requirements documents and analysis

Understand the Problem

- Understand the domain
- Functional requirements
- Non-functional requirements
- User interface
- Process improvements



Requirements - SMART

- **S** – Specific
- **M** – Measurable
- **A** – Assignable (who will do it?)
- **R** – Realistic
- **T** – Time-related (when should it be done?)

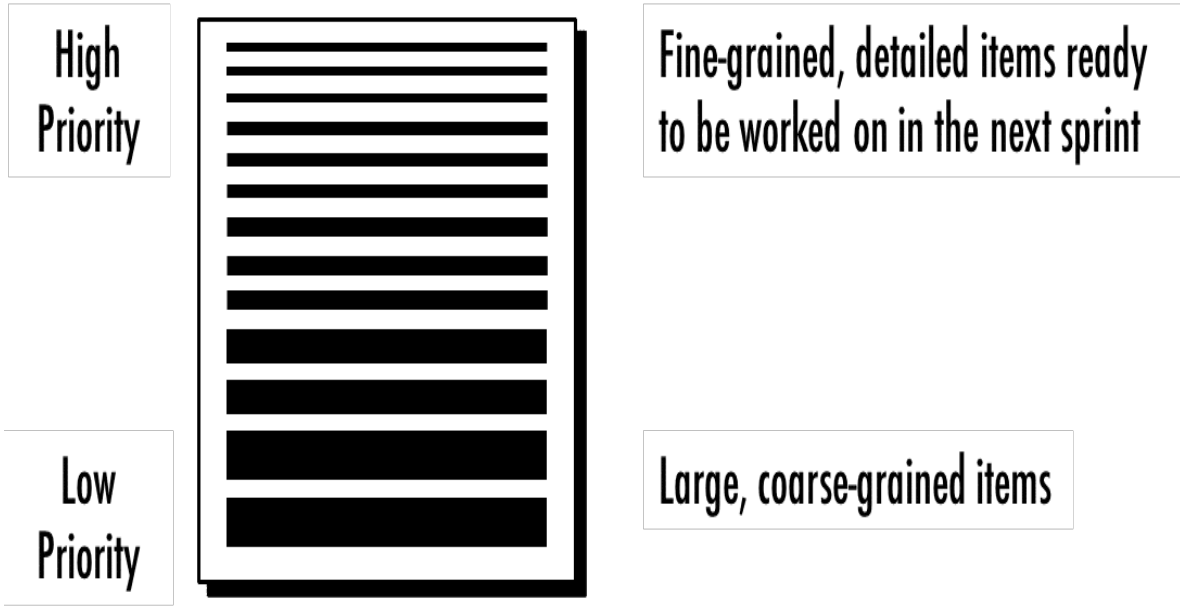
Look at SMART requirements in agile world

Stories - INVEST

- **I** – Independent
- **N** - Negotiable
- **V** - Valuable
- **E** - Estimate-able
- **S** - Small
- **T** - Testable

Backlog - DEEP

- **D** – Detailed Appropriately
- **E** – Estimated
- **E** – Emergent
- **P** – Prioritized

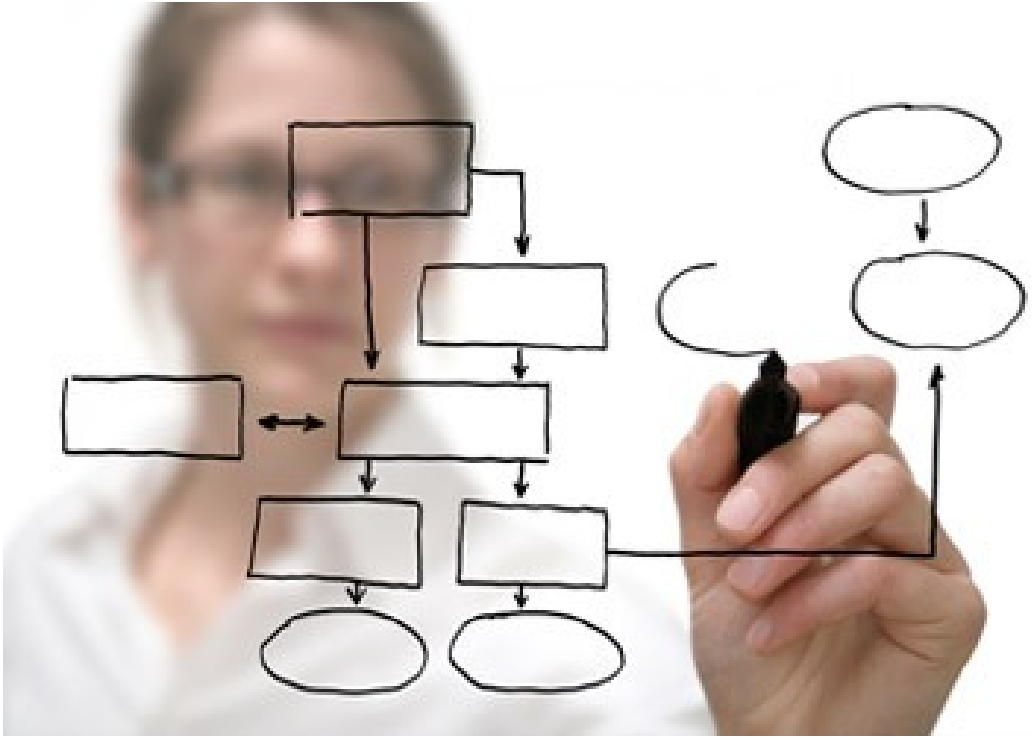


Backlog Item

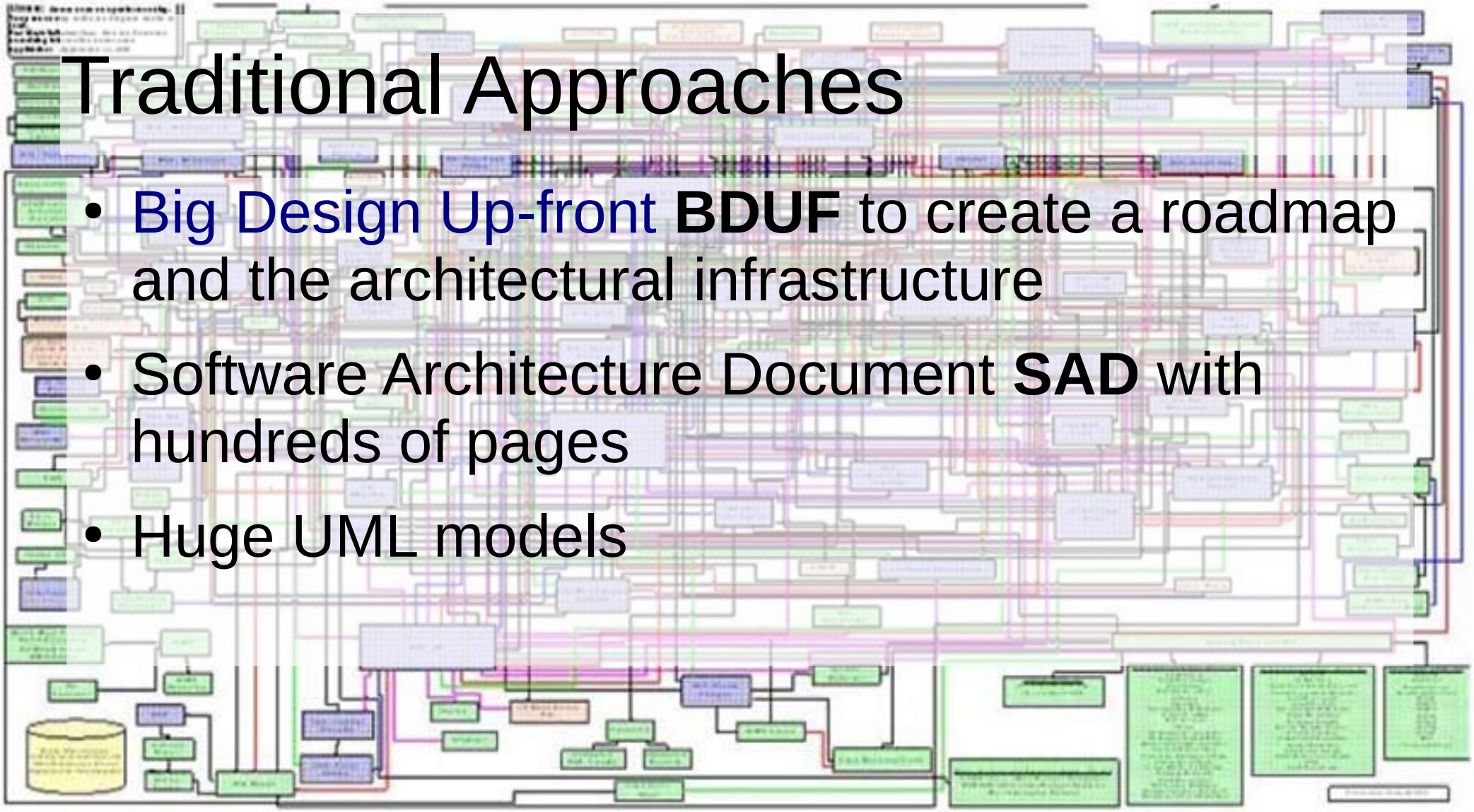
- What is a product backlog item *PBI*?
- Is a product backlog item a story?
- Why do you estimate a PBI?
- How do you know when a PBI is completed?

Create an Architecture

- Define an architecture
- Validate it
- Verify it
- Document it
- Evolve it



Traditional Approaches



- Big Design Up-front **BDUF** to create a roadmap and the architectural infrastructure
- Software Architecture Document **SAD** with hundreds of pages
- Huge UML models

DDD and Event Storming

- Customer Language
- Domain Knowledge
- Workshop and Discussion

- UX Workshop
- Design Thinking

Agile Approach (1/2)

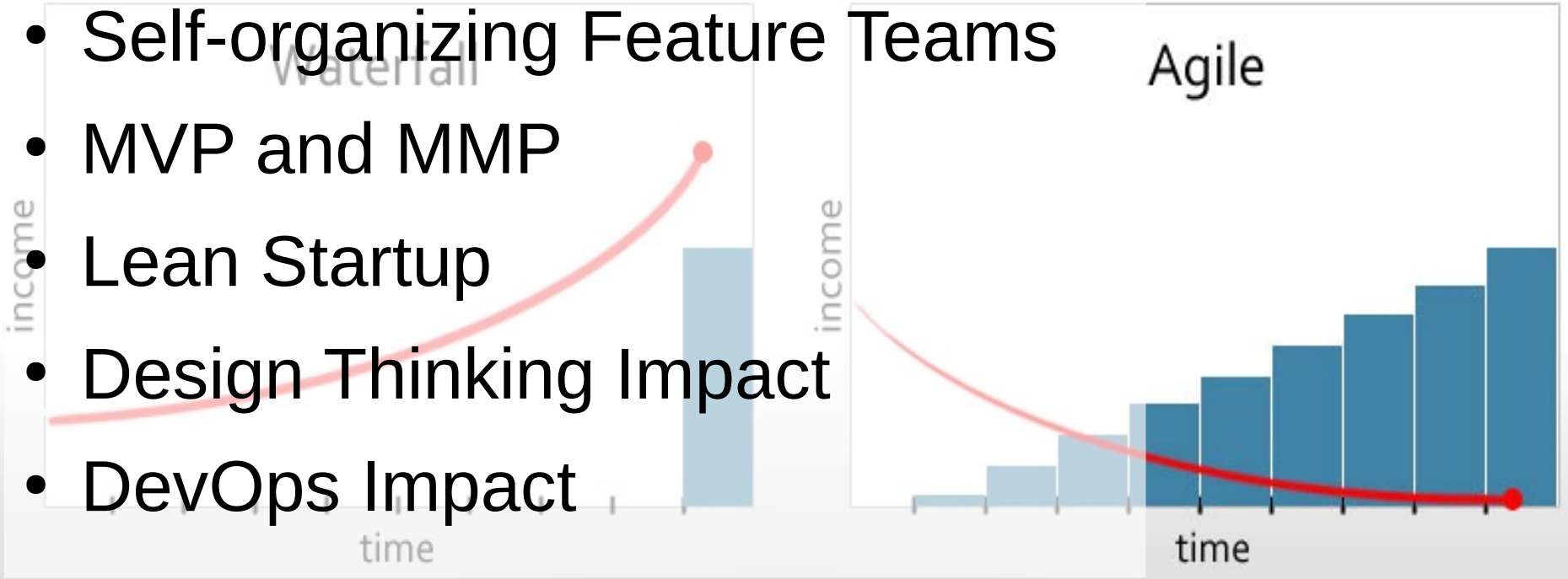
- Vision – Why?
- Roadmap – What do we get the next 9 -18 months?
- Release Planning – Story Map
- Sprint Backlog – What do we do the next 1-2 weeks?
- MVP Minimum Viable Product
- MMP Minimum Marketable Product

Agile Approach (2/2)

- *Agile Manifesto Principle 6*: The most efficient and effective method of conveying information to and within a development team is **face to face** conversation
- *Agile Manifesto Principle 11*: The best architectures, requirements, and designs emerge from **self-organizing teams**

Agile Impact to Architecture

- Self-organizing Feature Teams
- MVP and MMP
- Lean Startup
- Design Thinking Impact
- DevOps Impact



Agile Impact On Success

Standish Group, Chaos Report, 2018

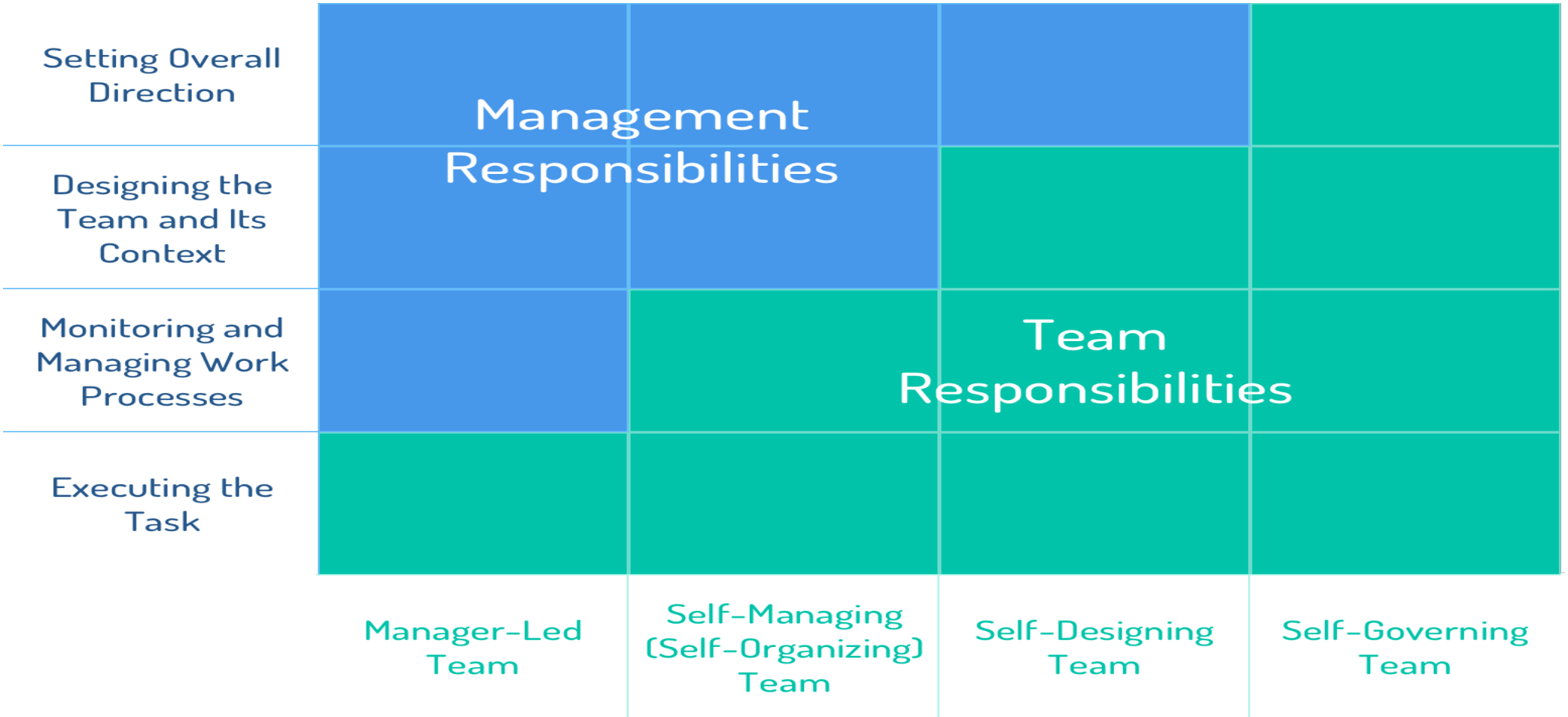
Architect and Developers

- Team Work
- Craftsmanship
- Team Dynamics

Craftsmanship Approach

- Architect is a **domain expert**
- Architect is a **software craftsmanship**
- Architect is a **lean leader** – teacher, coach, mentor
- Architect discuss with stakeholders and C-level representatives

Self-Organizing Teams

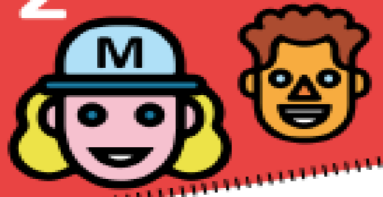


1

Tell

I will tell them

2



Sell

I will try and sell it to them

3

Consult

I will consult and then decide

4

Agree

We will agree together

5

Advise

I will advise but they decide

6

Inquire

I will inquire after they decide

7

Delegate

I will fully delegate

DELEGATION POKER

These cards are part of the Management 3.0 materials. They represent the 7 delegation levels for empowering organizations. You can find a description of their use at:
www.management30.com/delegation-poker

MANAGEMENT 3.0
CHANGE AND INNOVATION PRACTICES

Exercises (1/3)

- Create an UML diagram of your application
  - Why, when and how do you do it,
- Write a functional requirement – SMART -
  - How do you insure your requirement is testable?
  - Look at SPIDR for stories (Mike Cohn Video)
- Write a non-functional requirement
  - How do you insure your requirement is testable?
- Reflect changes introduced with agile and lean
  - Quality, speed, costs, success

Exercises (2/3)

- Read article “Agile Architecture in the Digital Age”
- Code examples of students
- Write unit tests, execute them in IDE, improve code coverage
  - Java: JUnit 5, Mockito, AssertJ
- Execute SonarLint on the fly on your source code

Exercises (3/3)

- Explore the refactoring features of the IDE
  - IDEA configure code style, copyright, etc.
  - IDEA Analyze Menu
    - Inspect Code, Clean Code
    - Find Usage, Find Declaration
  - Refactor Menu
    - Refactor (more than 10 operations)
  - Run IDEA “Analyze/Inspect Code…”, Spotbugs, SonarLint
  - Use Git integrated client – commit, amend, push,
  - Use local history feature