Inf. Colloquium

# Emergent Design in Agile Software Development Practices

Bruno Rossi
*CASE (Center for Applied Software Engineering)*
*Free University of Bolzano-Bozen*

## Introduction

- **Agile Methodologies** (AMs) for software development are based on an incremental process that gives value to the customer during each iteration
- There is **less up-front planning** compared to traditional techniques
- With respect to the architecture of the system, agile methodologies describe it as **emerging design**, that is the architecture of the system will emerge from several iterations over time
- During this presentation, I will address the following questions:

  → **How do we ensure that this will not lead to mere chaos?**

  → **How do we know whether the approach could be appropriate for our application/domain?**

## Outline

- I will start delving into **architectural concerns** in traditional software development
- I will then delve **into the principles of the agile methodologies**, and how those can lead us towards principles for architecture in AMs
- I will then provide evidence from a “real project” about the **emerging design process**, as well about how the principles work in practice

## The software architecture

- In early computing days, **algorithms** and **data structures** constituted the major design focus
- As size of software systems started growing in the 70's and 80's, it appeared evident that the organization of a software system - the **software architecture** – deserved more attention
- **Composition** of design elements, **scaling & performance**, and the selection among **design alternatives** all became relevant

## The software architecture

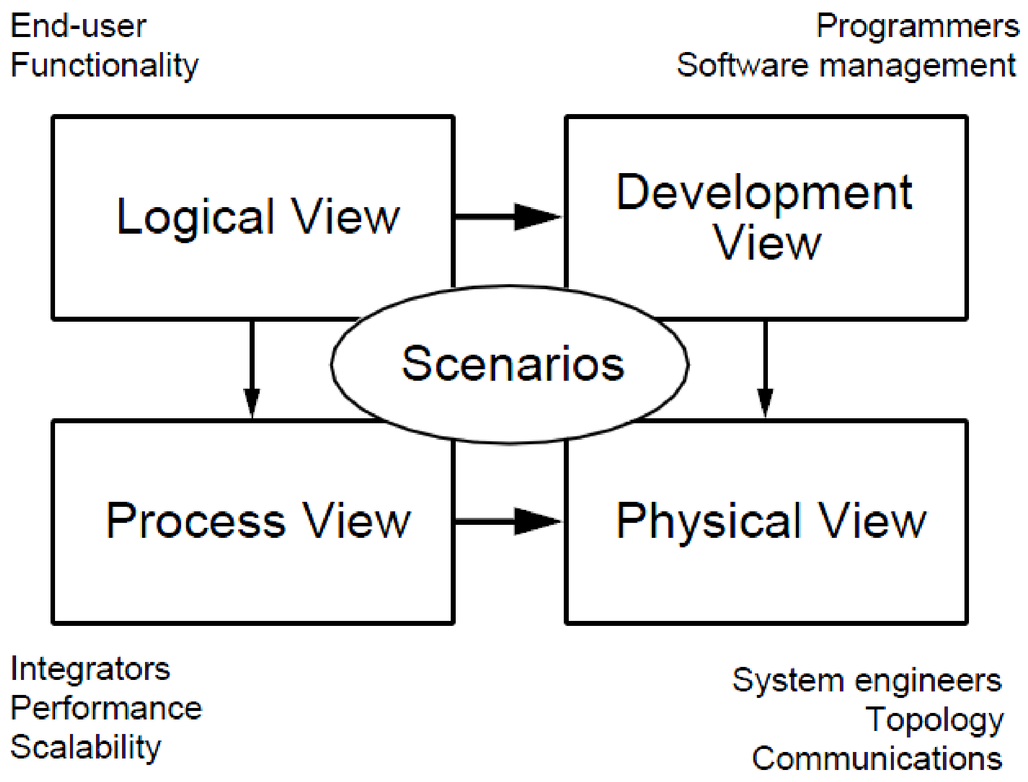
- The **architecture** of a software system
  - is about the **structure of a system** in parts (components, modules, files, objects, classes)
  - deals with the **parts**, their **properties**, and their **mutual relations**
  - facilitates the **comprehension** of the system
  - supports the **communication** among the different stakeholders of the development process

## The software architecture

- An architecture is formed by **several structures** (Bass, Clemens, Katzman, 1998) that we call **“projections”**
- It can be useful to analyze architectures at different levels of abstraction, from different points of view: package, class, and object, physical
- For each of these **“points of view”** (we call them “projections”) it is possible to find so called “architectural styles”
- This is similar to the view in IEEE1471 Standard

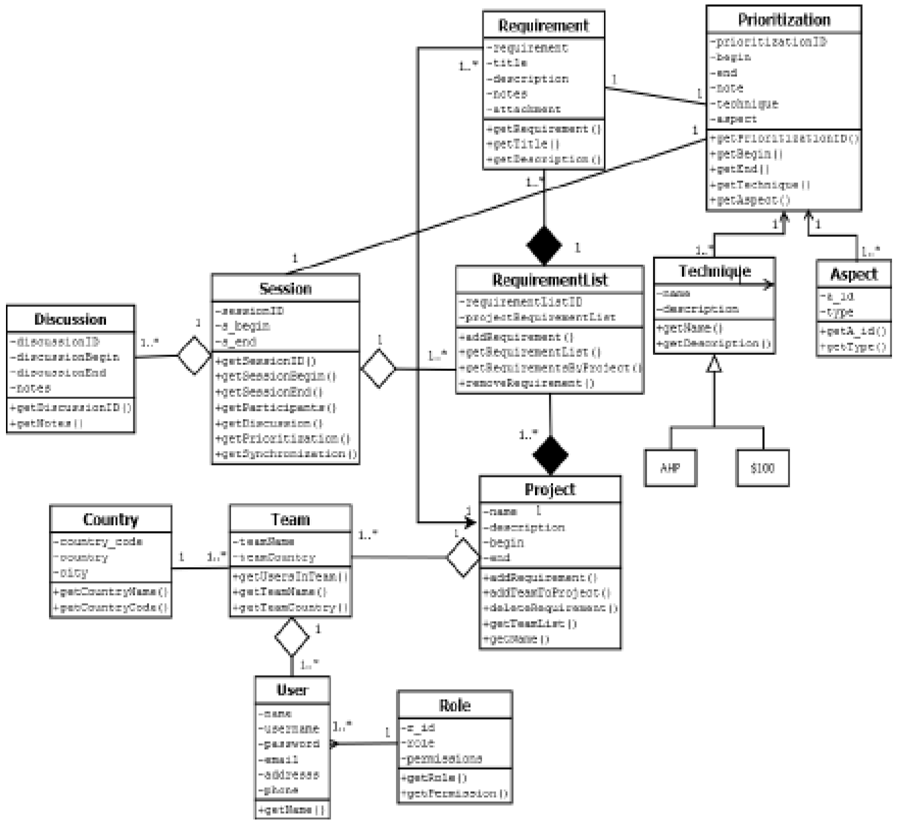
# The software architecture

- An Instance of IEEE1471 Standard is Kruchten 4+1 model



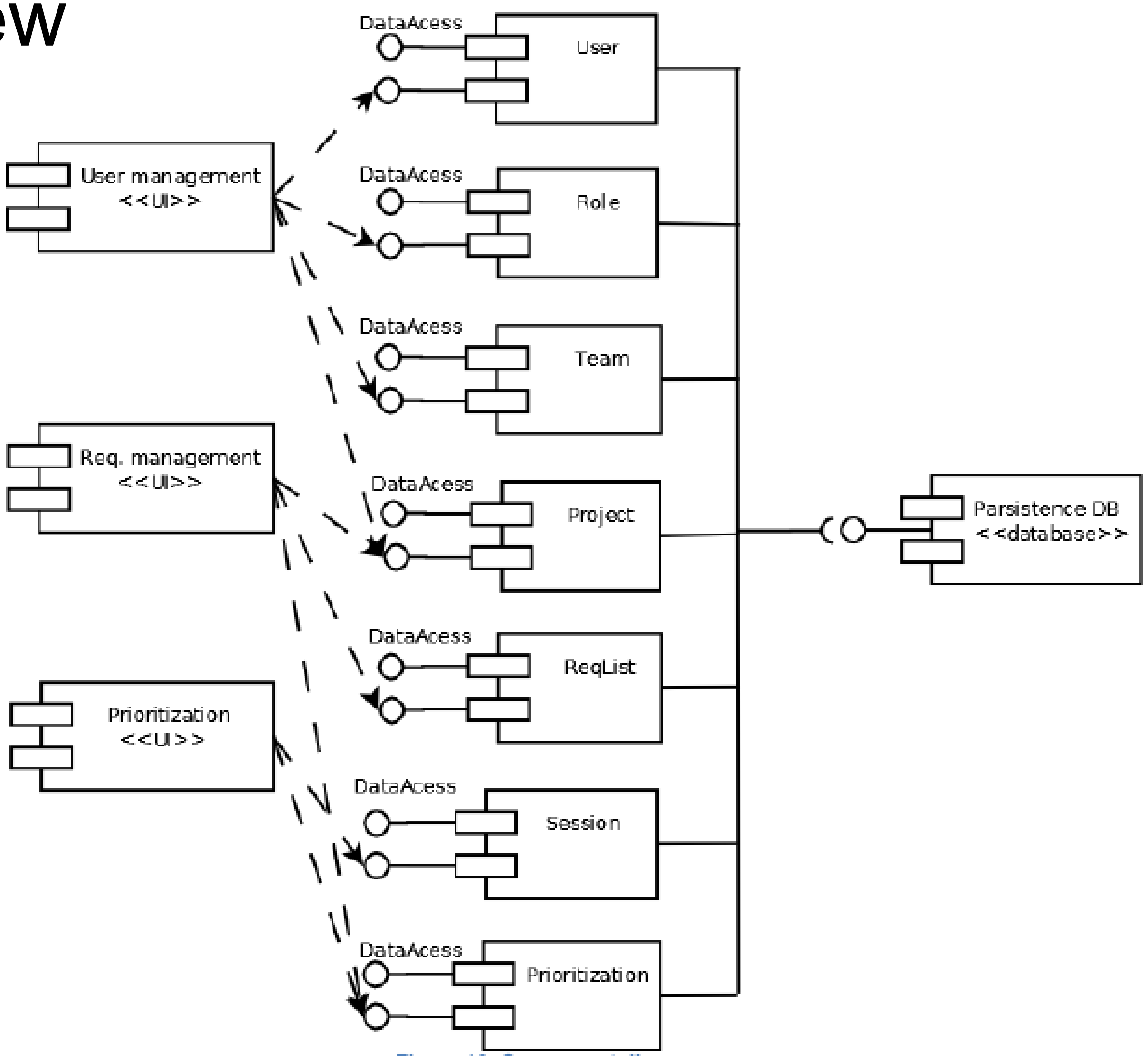
# The software architecture

- Logical View



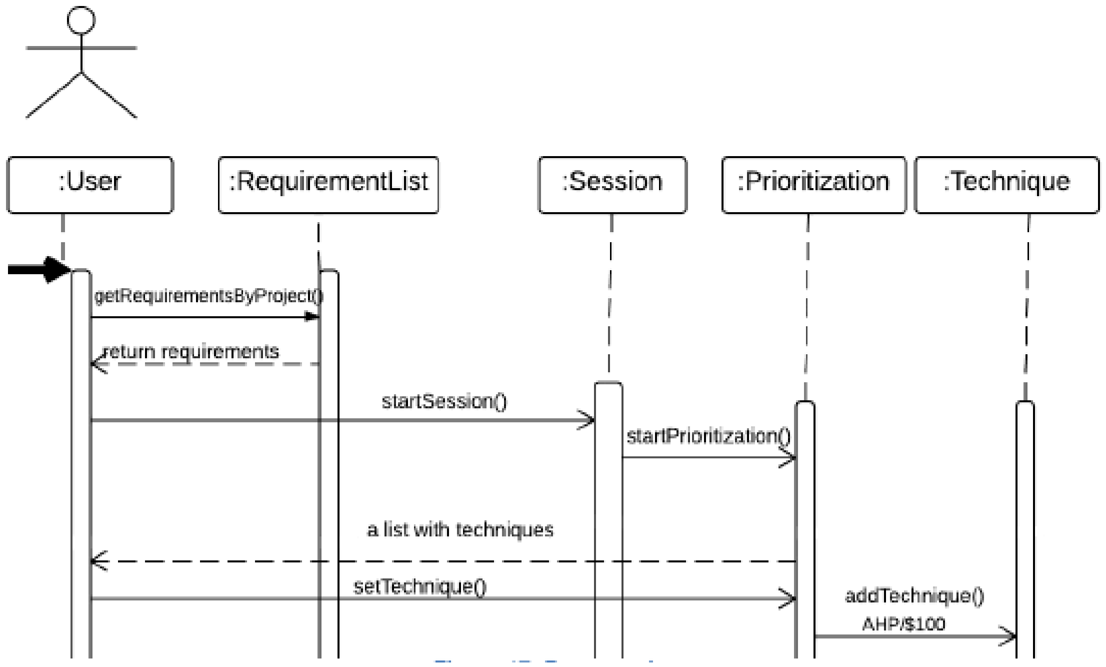
# The software architecture

- Development View



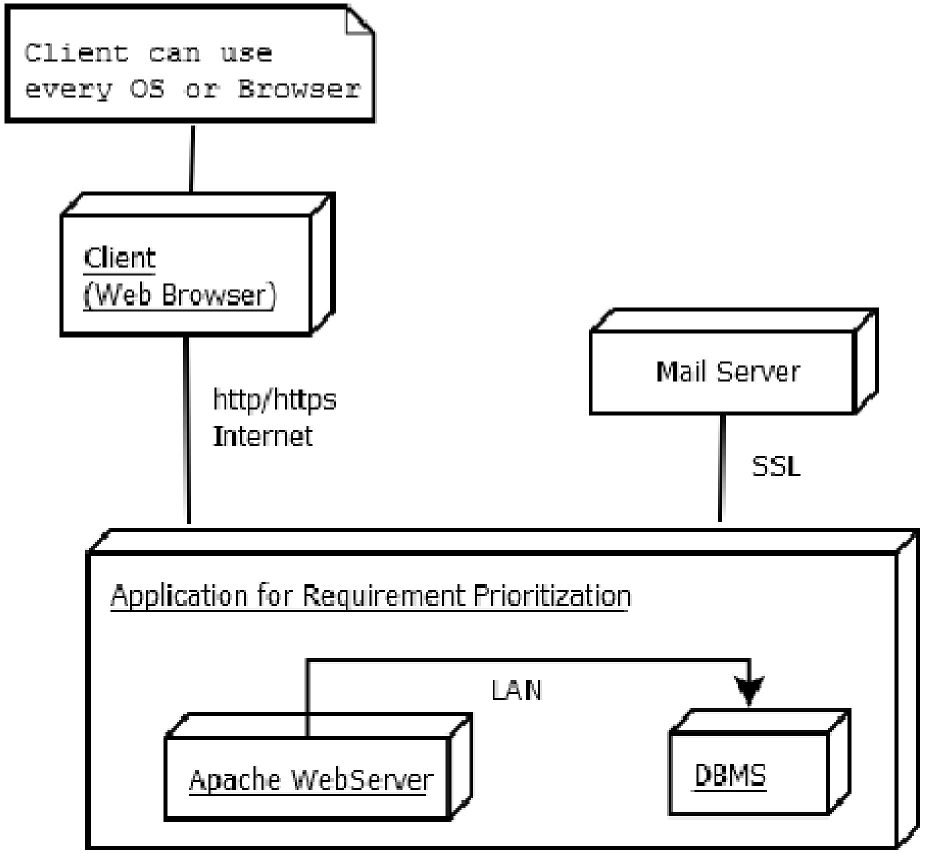
# The software architecture

- Process View



## The software architecture

- Deployment View



## Quick Recap on AMs (1/3)

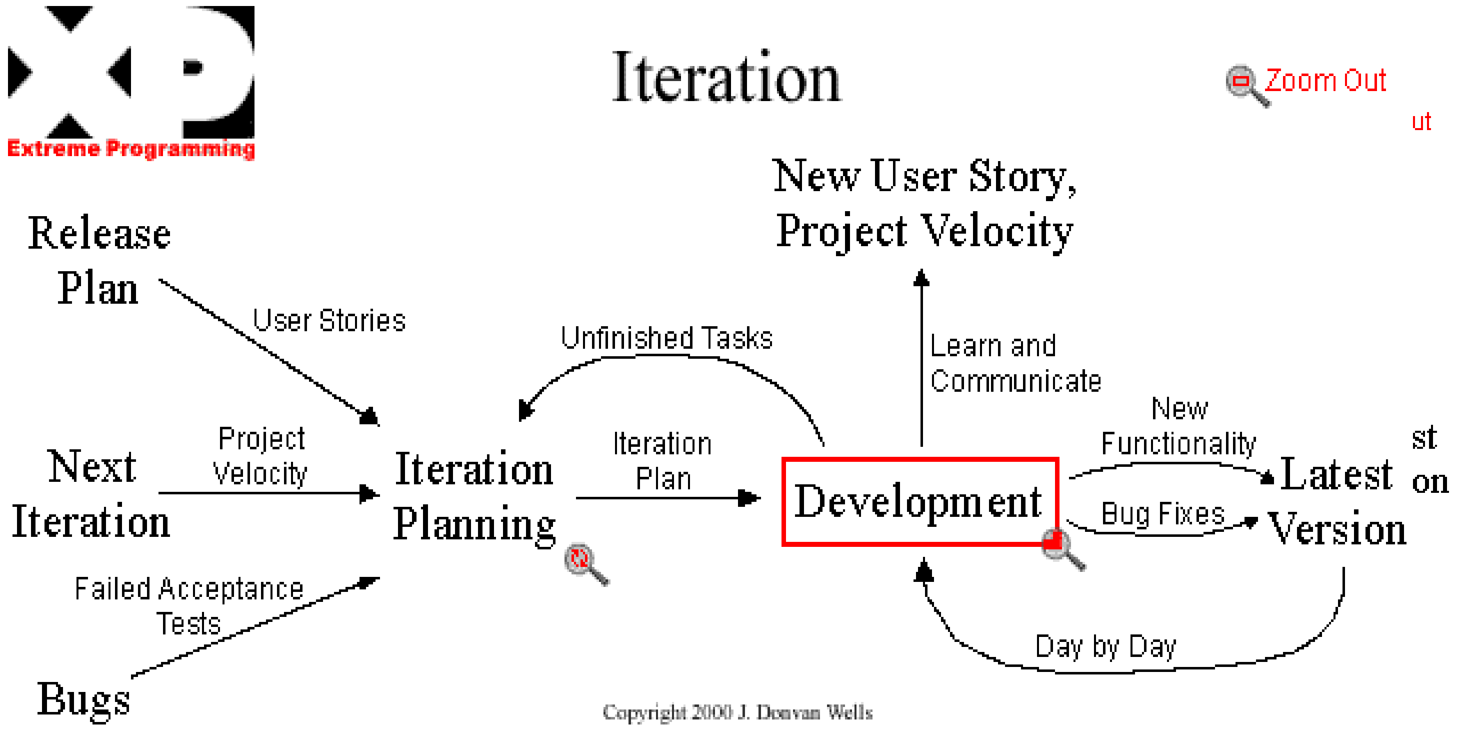
- Agile Development practices derived from the **failure of plan-based development processes** and the idea to **apply lean practices** from industry to software development
- Lean software development is based on a set of principles
  - Eliminate waste
  - Amplify learning
  - Decide as late as possible
  - Deliver as fast as possible
  - Empower the team
  - Build integrity in
  - See the whole

# Quick Recap on AMs (2/3)

- **Eliminate waste** : everything that does not add value to the customer is discarded as waste (**muda**)
- **Amplify learning**: the development process is a learning process, short iterations and learning are key factors
- **Decide as late as possible**: to counteract possible changes of requirements
- **Deliver as fast as possible**: as in the just-in-time production system
- **Empower the team**: more bottom-up approach, developers are not mere resources
- **Build integrity in**: the customer must perceive integrity of the system, refactoring is a way to build simplicity into the process
- **See the whole**: in short, "Think big, act small, fail fast; learn rapidly"

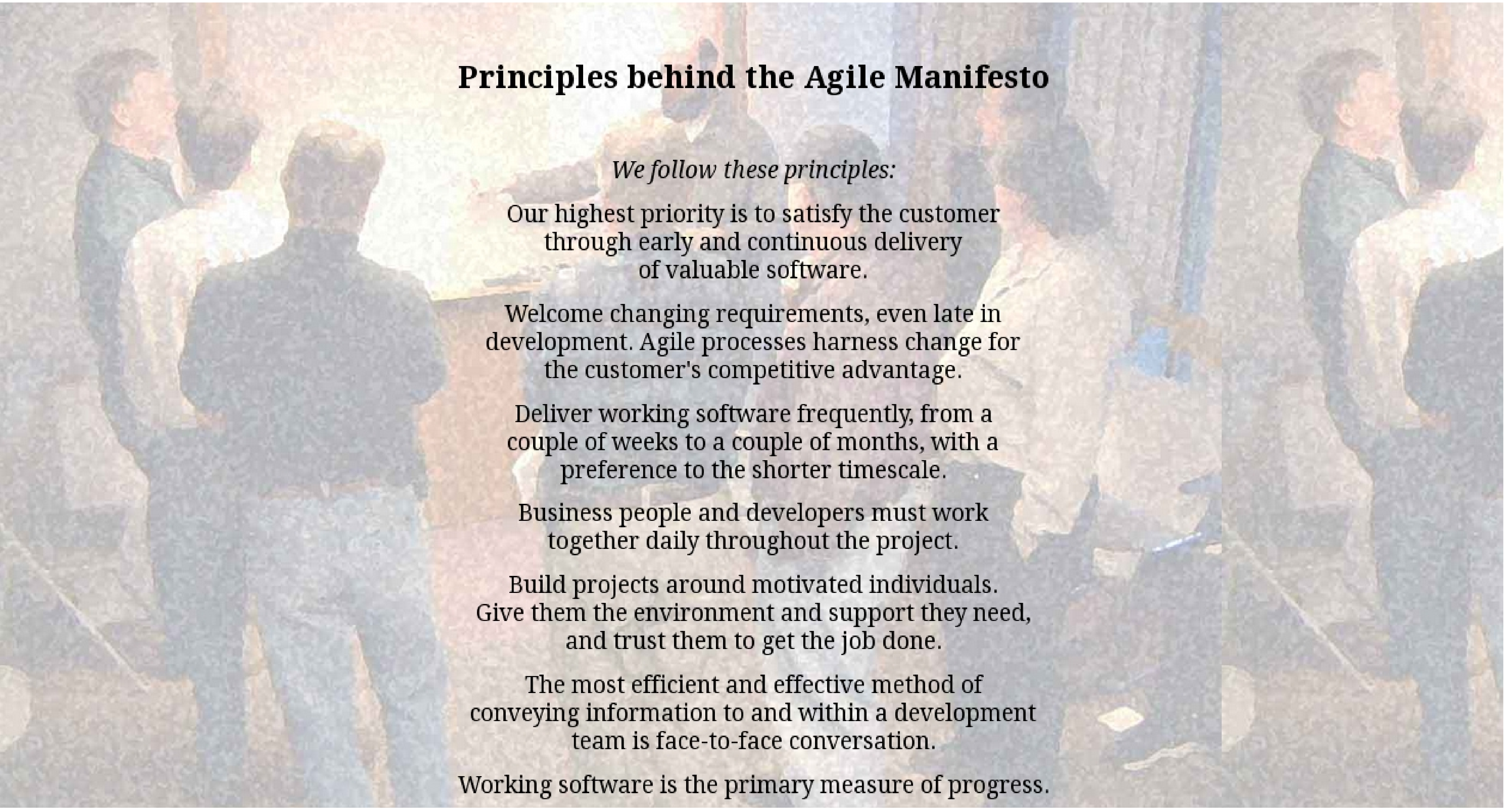
# Quick Recap on AMs (3/3)

- The eXtreme Programming process description (just one instance of AMs)



- Where are the **architectural concerns**?

## The Agile Manifesto principles – relation to the software architecture?



**Principles behind the Agile Manifesto**

*We follow these principles:*

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

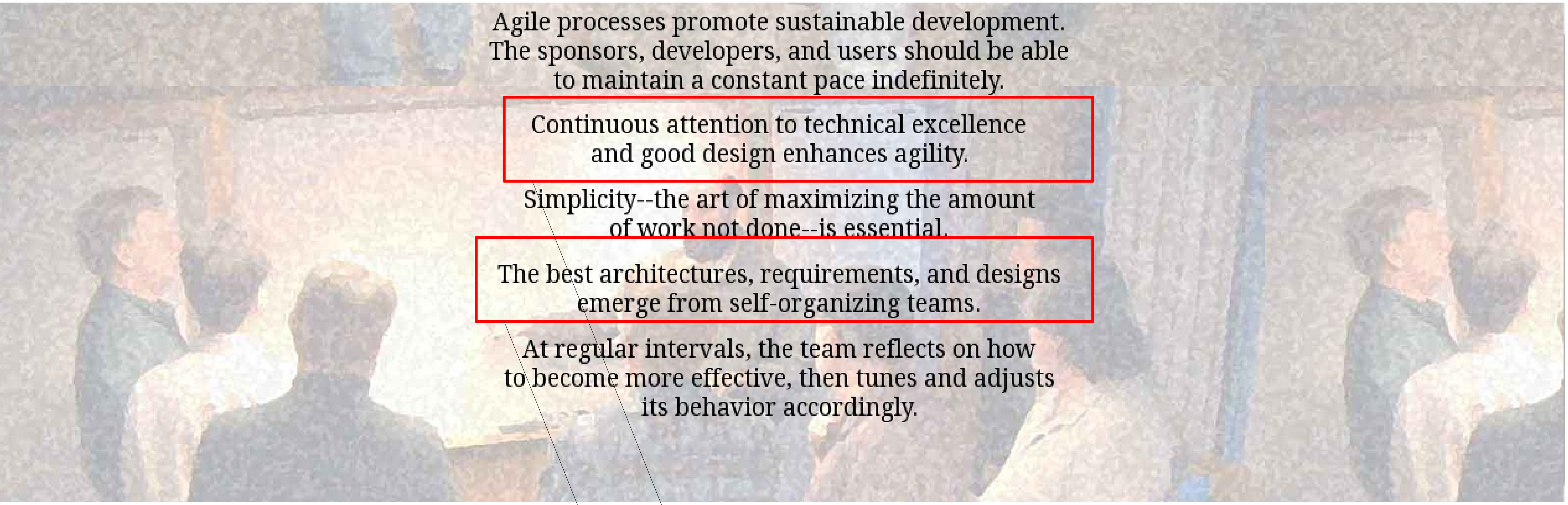
Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

# The Agile Manifesto principles – relation to the software architecture?



Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

This is what we have from the Agile Manifesto underlying principles

## Emergent Design

- What is an **emergent design**?
- *An emergent design is a proper program structure that derives directly from the coding process*
- *The idea is that by focusing on the problem at hand, a proper architecture will emerge naturally*

- Leffingwell reported the 8 principles of the software architecture in AMs

* D. Leffingwell, Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise, 1st ed. Addison-Wesley Professional, 2011.

## The 8 Principles of the Agile Architecture

- **Principle 1:** the teams that code the system also design the system
- Driven by the Agile Manifesto (*the best architectures, requirements, and designs emerge from self-organizing teams*)

  Teams **must be considered accountable** for the decisions taken, not those taken by somebody else!

## The 8 Principles of the Agile Architecture

- Accountability and responsibility

| | **Pre-Agile** | **Agile** |
|---|---|---|
| **Architect's Responsibility** | Analyze requirements<br>Design the system<br>Interface to key business stakeholders and customers<br>Bid the work for teams<br>**Only one that understands how the whole works** | Analyze architectural epics<br>Collaborate with stakeholders and development teams<br>Get implementation feedback and development estimates from teams<br>Maintain system models ad model future state based on new epics |
| **Team Responsibility** | Inherits the plan and the work estimates<br>Inherits the architecture<br>Left "holding the bag" and executes on a "best effort" basis | Interface in business and customers via product owner role<br>Tech leads participate in virtual, extended system architecture team<br>Responsible for subsystem design<br>Estimates work for their area of concern<br>Commits of behalf of themselves<br>Accountable for the results |

Source: D. Leffingwell, Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise, 1st ed. Addison-Wesley Professional, 2011.

## The 8 Principles of the Agile Architecture

- **Principle 2:** Build the simplest architecture that can possibly work
- Driven by the Agile Manifesto *(Simplicity - the art of maximizing the amount of work not done - is essential)*

## The 8 Principles of the Agile Architecture

- **Principle 3:** When in doubt, code it or model it out
- Having a visible process by means of **stories and prototypes** allows all the stakeholders to see the reasoning that is behind the architectural building
- You can use the 4+1 views to model, **but remember about throw-away prototyping**!

## The 8 Principles of the Agile Architecture

- **Principle 4:** They Build It, They Test It
- It is the **responsibility of the development teams themselves to develop, test, and maintain a system-testing framework** that continually assesses the system’s ability to meet its functional and non-functional requirements.
- **Evolving the architecture means to evolve also the testing framework**. A key principle of agile methodologies is that if the team design one part and built it, then the team must also test it

## The 8 Principles of the Agile Architecture

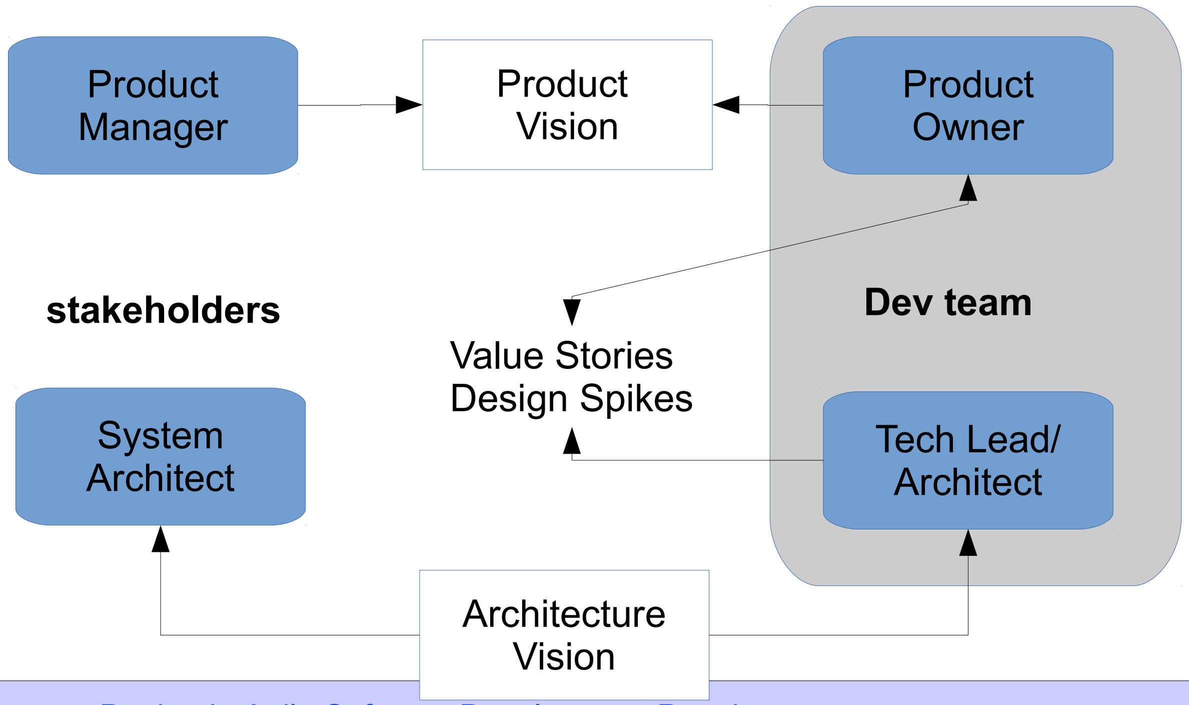
**Principle 5:** The bigger the system, the longer the runway

- The assumption of a delivery commitment **depends on how much reliable the foundations** are
- So there must be an **architectural runway** that constitutes a baseline for solid iterations – without it missed deadlines are going to happen

## The 8 Principles of the Agile Architecture

**Principle 6:** System architecture is a role collaboration

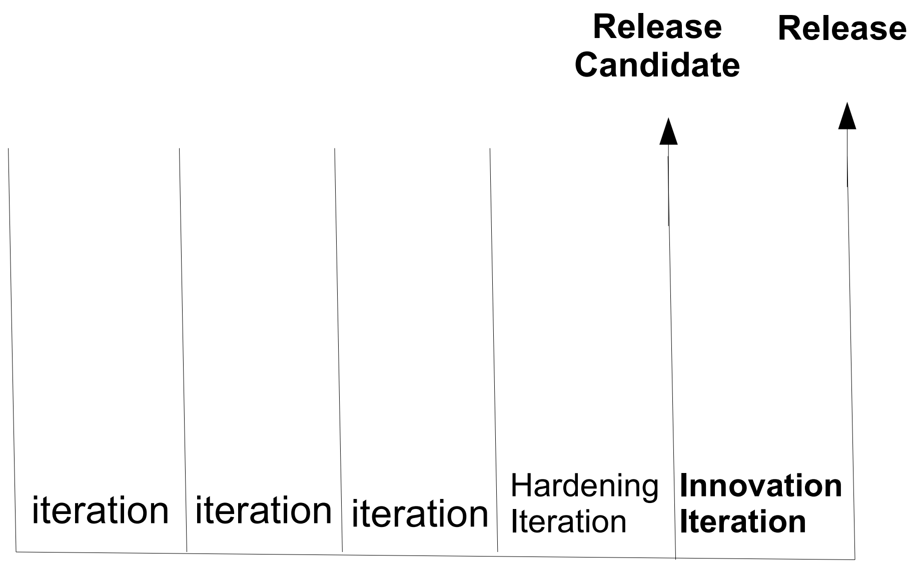
- system architects work with the team technical leaders **to define the design spikes** they will be use to get the common architectural view - if in doubt, **coding or modeling** (**principle 3**) can be used



# The 8 Principles of the Agile Architecture

**Principle 7:** There is no monopoly on innovation.

- One way to foster innovation at the team level is by **managing the backlog to have iterations that include spikes to explore new ideas**



Source: D. Leffingwell, Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise, 1st ed. Addison-Wesley Professional, 2011.

# The 8 Principles of the Agile Architecture

- **Principle 8:** Implement architectural flow
- Just as with the stories in the backlog, **also for the architectural part there must be an active management**, that is the **process must be visible and transparent**, and all must be continuously updated to maintain a flow for the architectural decisions
- This is a key part: how do we maintain the **system architecture if we need to perform changes**? One of the key aspects of AMs is to **always have a running system**
- In agile, we do not branch and then merge later, we need to apply **continuous refactoring supported by unit tests**... Still, for some of the largest architectural stories, implementing them in the context of refactoring sessions can be impossible

## Principle 8 – implement architectural workflow

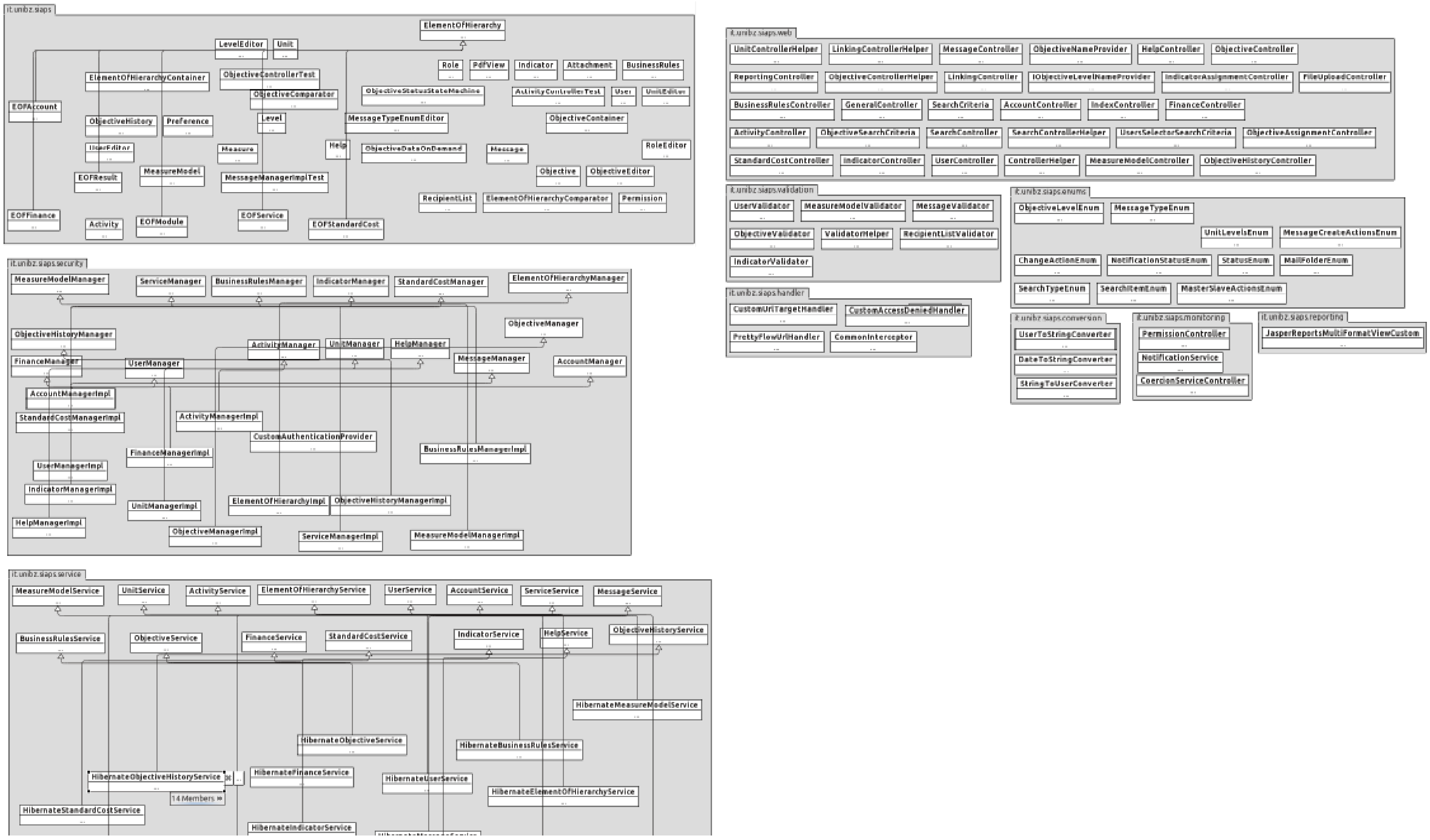
- Three cases foreseen
- **Case A**, big but incremental, the system always runs
- **Case B**: Big, but Not Entirely Incremental → the System Takes an Occasional Break
- **Case C**: Really Big and Not Incremental; the System Runs When Needed; Do No Harm

## Experience from the trenches

- I will provide some considerations from a large development project to manage strategical planning
- Medium-Large codebase ~7000 Java LOCs plus JSP and Javascript code
- ~5 developers, 1 year time
- No software architect external to the team

- The approach is Action research-based, as the presenter was one of the participants to the project

# Experience from the trenches -the Architecture



## Experience from Practice

- How did the design emerge? Two detailed ways

  → **Agreement within the team**

  → The **frameworks and technologies used imposed some architectural choices** (e.g. security layers)

Major lesson learnt: there must be a **senior developer** (in absence of an architect) that takes into account the **architectural integrity of the system** – warning other team members about parts of the system that  violate the architectural view common to the team

## Experience from Practice

- How to maintain architectural integrity over time? There are some key points:
- **Stakeholders must be aware of the fact that some iterations are for innovation,and refactoring** – asking to do refactoring and coding at the same time is not feasible
- Must **convince the stakeholders** about the importance of those iterations
- **Without a testing system** in place mere architectural **reconstruction is not possible**
- When you **start losing the architectural runway**, **changes will require more code duplication, more time, and as well lead to more bugs** (if we assume that more code leads to more bugs)

## Experience from Practice

- How to communicate the architecture to other team members?
- The **composition of the teams is very likely to change over time – generation of architectural views from the code** is a key part in letting new team members understand the current system's architecture
- **Throw away modeling** is not a problem in this case, and typically initial models if not updated can lead to problems

## Experience from Practice

- **Refactoring sessions**
- If **refactoring sessions** are not allowed by the stakeholder, the system architecture will start becoming a sort of compromise
- This is especially relevant if you have **changing requirements** and **quite large system**

## Derivation of Research Questions from the Study

- Some research questions we derived from the pilot study:
  - How is the emerging design process **distributed among team members**?
  - **How much rework** is implied in this kind of process? That is how many changes are there in the structural parts of the system?
  - Do team members have an **exact view of the architecture**, and how much such view is **aligned with the one implemented**?

## Conclusions

- We could also answer our initial questions
- How do we ensure that this will not lead to mere chaos? **Many practices of AMs** are of **paramount importance** in this sense: **refactoring**, **continuous integration**, **collective code ownership –** without those practice it is impossible to build a proper architectural runaway
- How do we know whether the approach could be appropriate for our application/domain? **Does the customer request constantly for architectural information – do they want to be part of the architecture building process?** If this is the case, you **need** to have an external software architect

## Questions

- Thank you!
- I will be happy to answer your questions, for additional enquires → bruno.rossi@unibz.it

WE'RE GOING TO TRY SOMETHING CALLED AGILE PROGRAMMING.
THAT MEANS NO MORE PLANNING AND NO MORE DOCUMENTATION. JUST START WRITING CODE AND COMPLAINING.
I'M GLAD IT HAS A NAME.
THAT WAS YOUR TRAINING.
www.dilbert.com
scottadams@aol.com
11-26-07 ©2007 Scott Adams, Inc./Dist. by UFS, Inc.

I'LL GO UP AND FIND OUT WHAT THEY NEED AND THE REST OF YOU START CODING!