

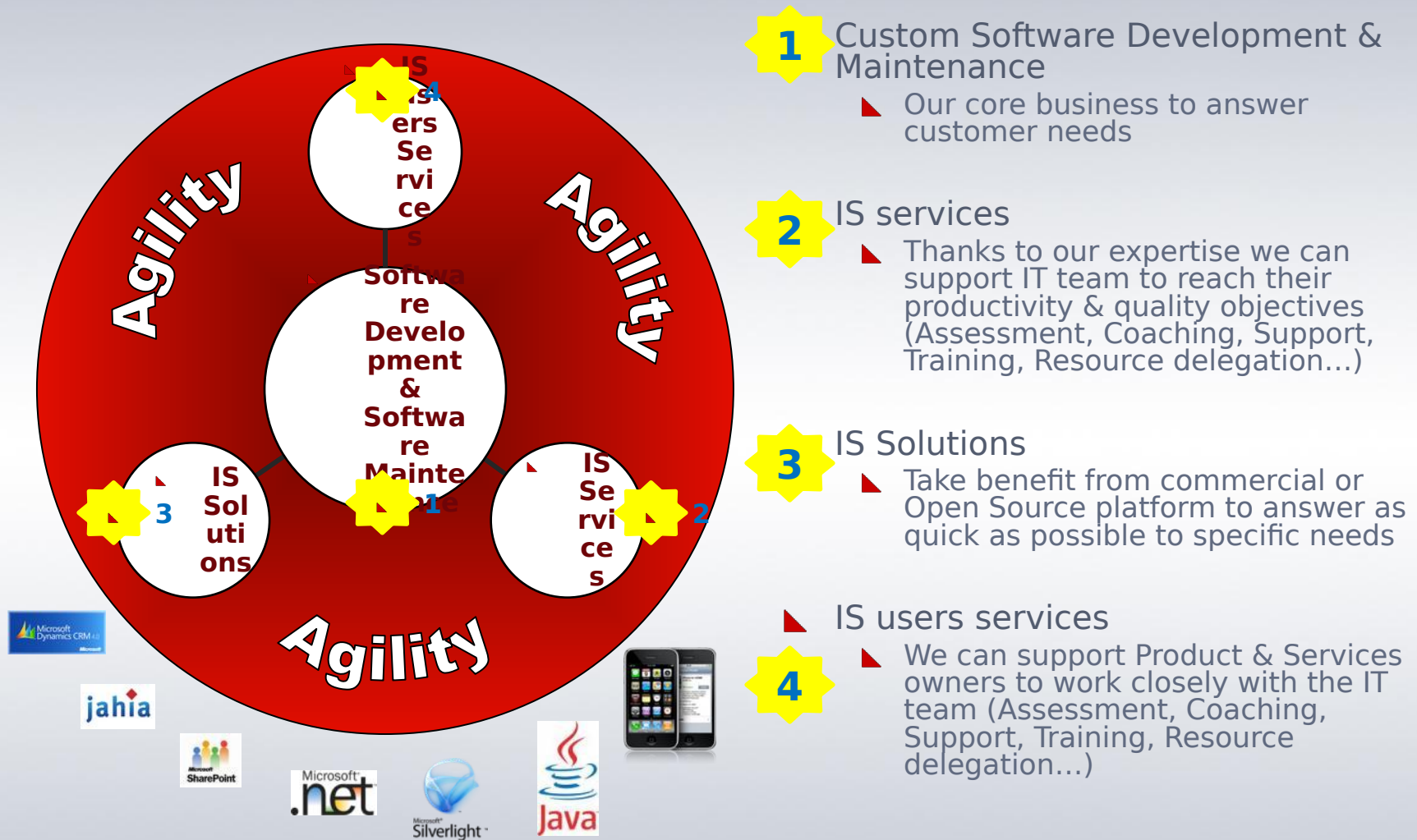


agilepartner

Agile architecture & programming

Agile Mötteg - 16 September 2010

- ▶ Understand the implications of agile development on **architecture**, **design** and **coding practices**
- ▶ Explain some **myths** about software architecture and agility
- ▶ Show some best practices



▴ About Me

▲ Agenda

- ▲ Myths about agility and architecture
- ▲ What do we mean by *Architecture*?
- ▲ What do we mean by *Agile*?
- ▲ How does agility affect architecture?
- ▲ Introduction to Domain-Driven Design
- ▲ Introduction to Component-Based Design
- ▲ Coding practices

MYTHS ABOUT AGILITY AND ARCHITECTURE

- ▲ Agile = Fragile
- ▲ No Architecture
- ▲ No Design
- ▲ Only works for small projects



WHAT DO WE MEAN BY ARCHITECTURE?

- ▀ Architecture embodies the critical design decisions that classify a system
 - ▀ Enterprise Architecture (relates to organizational structure, cost of change, ...)
 - ▀ Technical Architecture (infrastructure)
 - ▀ **Software Architecture** (capabilities of the system, structure of code ...)
 - ▀ etc.

- ▶ The importance of decisions needs to be well understood and assessed
 - ▶ Heavy-weight approaches are likely to reduce the understanding and the ability to evaluate
 - ▶ Keep it simple!

- ▶ A good architecture is one in which the significance of decisions is reduced
 - ▶ This means key decisions are the ones that reduce the significance of other decisions
 - ▶ Makes changes easy
 - ▶ Reduces cost of change
- ▶ A stable and robust architecture does not mean it is “frozen”

- ▶ Base for further decisions
- ▶ Influence on team structure
- ▶ Foundation for GUI
 - ▶ “The interface is the program” (Raskin)
- ▶ Reflects system design
 - ▶ Web, Smart Client, Client/Server, Embedded ...

WHAT DO WE MEAN BY AGILE?

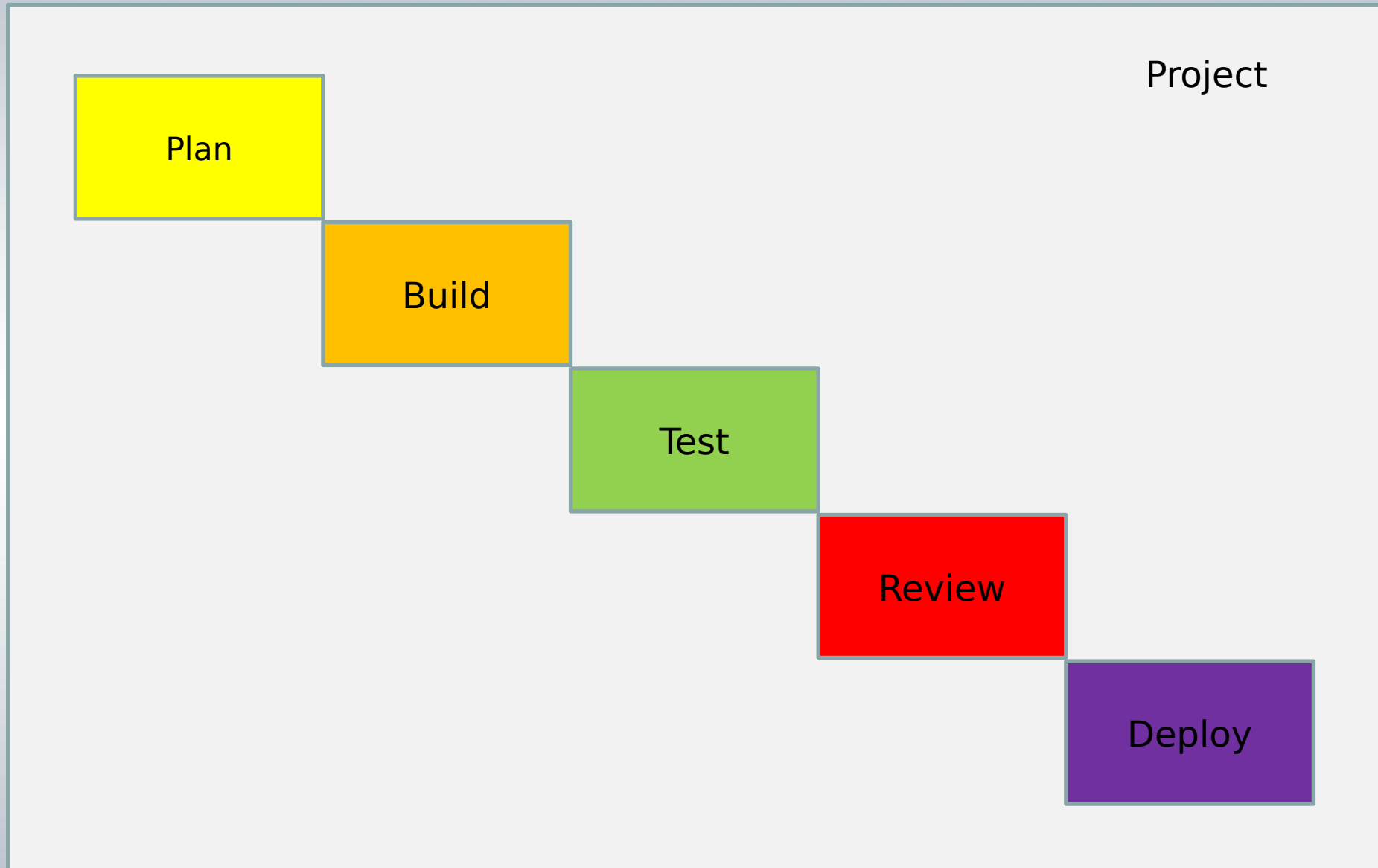
- ▶ Individuals and interactions over processes and tools
- ▶ Working software over comprehensive documentation
- ▶ Customer collaboration over contract negotiation
- ▶ Responding to change over following a plan

- ▶ What makes development agile is ...
 - ▶ Sustainability of process and code.
 - ▶ Feedback at different levels of scale
 - ▶ Awareness of what is being built and how
 - ▶ The right detail at the right time and in the right place
 - ▶ The engagement of people in the process
 - ▶ Economy

HOW DOES AGILITY AFFECT ARCHITECTURE?

- ▶ Requirements for an “agile” design
 - ▶ The design should improve inter team communication.
 - ▶ It must be comprehensive to everyone without a lot of documentation
 - ▶ It must support testing
 - ▶ The design must support customer communication and collaboration
 - ▶ It must respond to change

BUFD, NUFD OR RUFD?



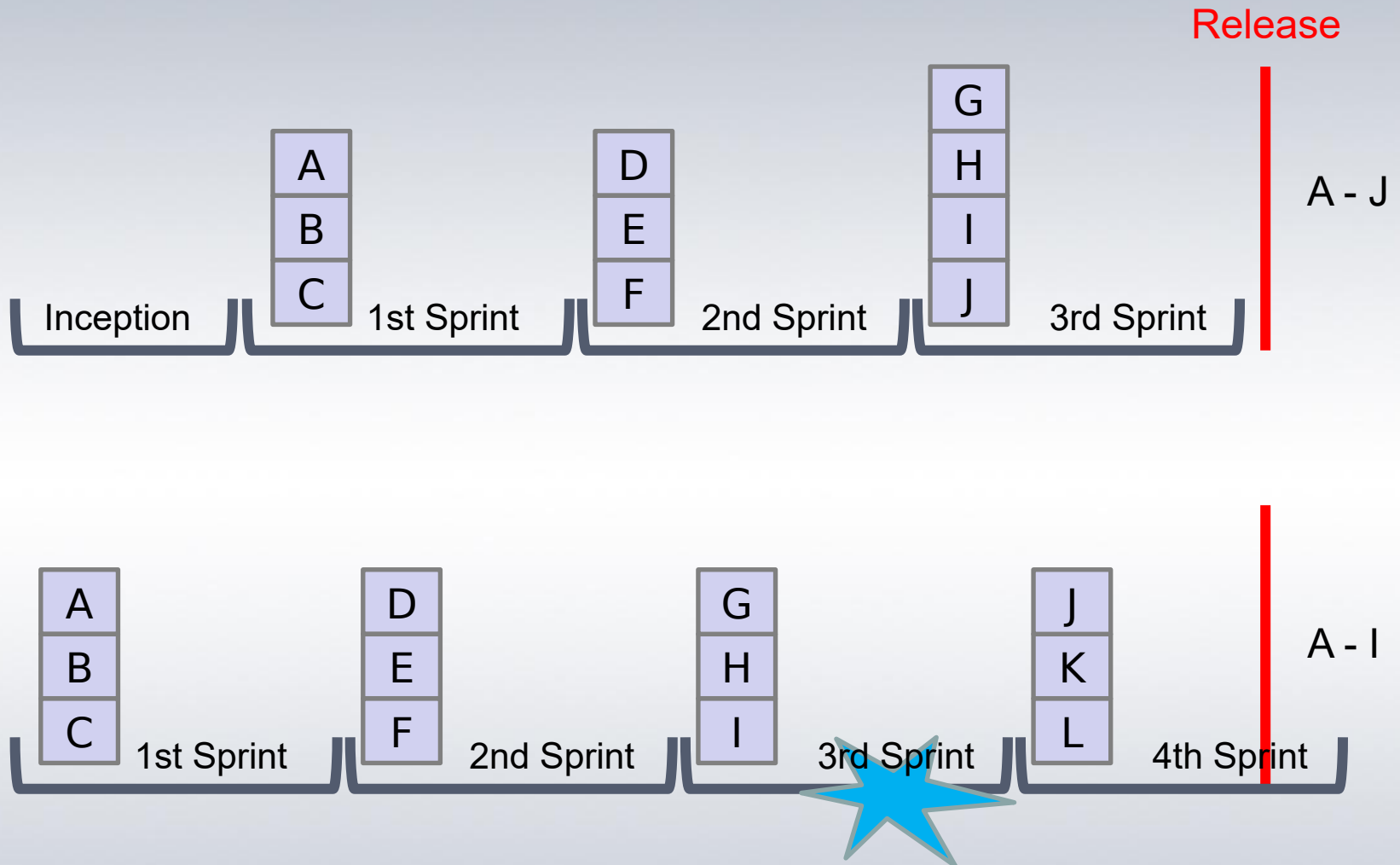
- ▴ Big upfront design
 - ▴ Impossible to know everything in advance
 - ▴ Lack of flexibility
 - ▴ Lack of extensibility
 - ▴ Lack of adaptability
 - ▴ “Ivory Tower” metaphor
 - Architecture might be ignored because team members were not involved in its creation

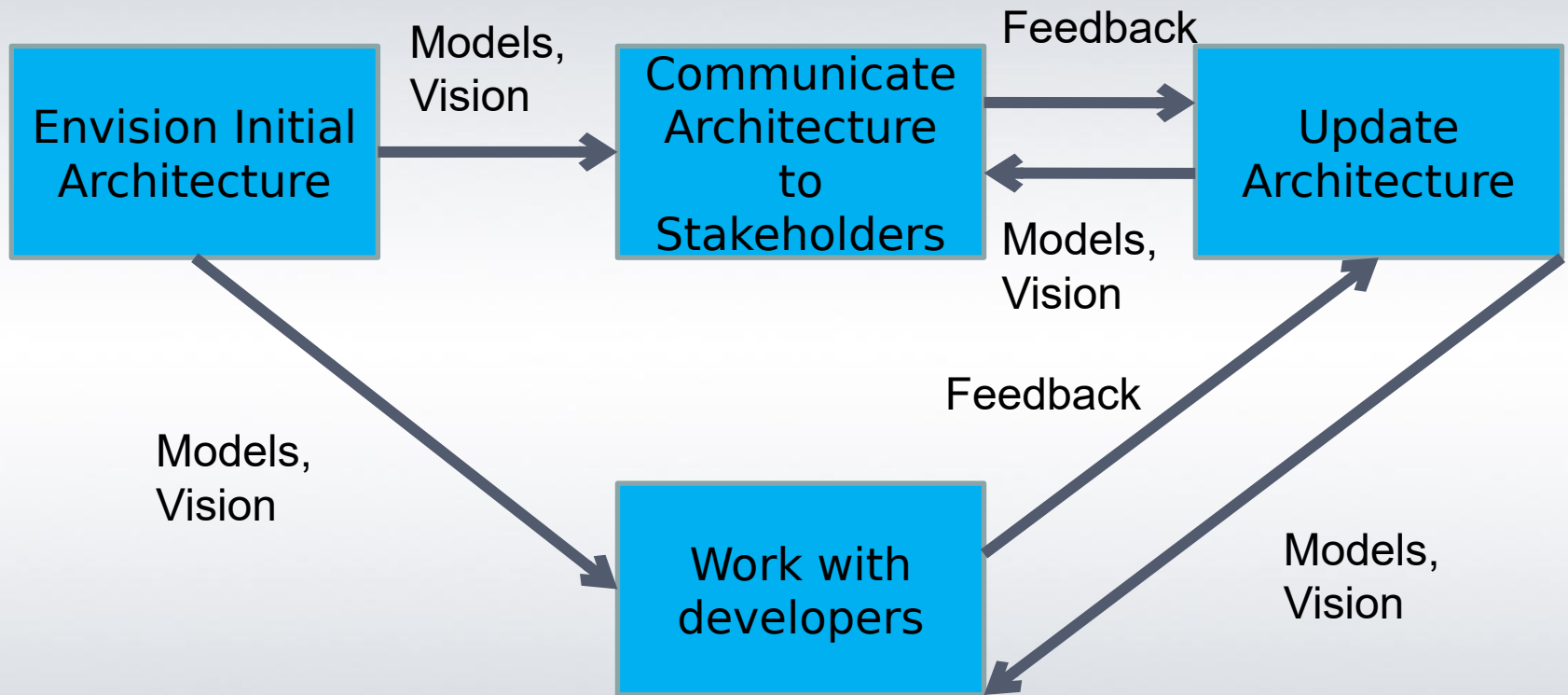
Sprint

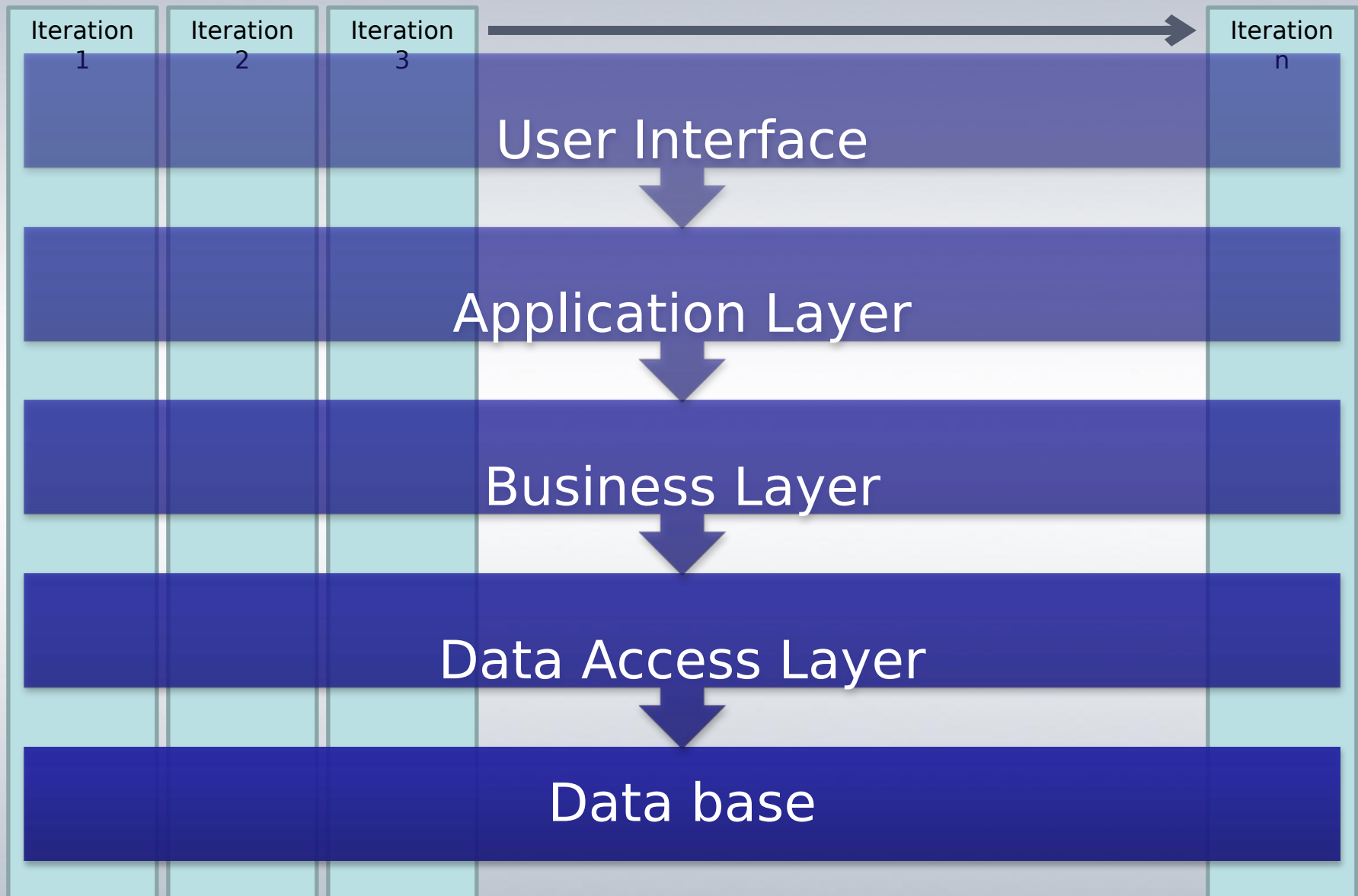


- ▶ No Up-Front Design
 - ▶ Team members have a very high technical expertise
 - ▶ Team members have in-depth knowledge of the domain

- ▲ Keep up-front design short
 - ▲ But not too short
 - You do not have to **know precisely what to build before you can start building it.**
 - A sprint/iteration 0 can help defining key architecture concepts, a vision and initial architecture
 - ▲ Use existing expertise
 - Delaying critical design decisions does not mean we ignore existing knowledge
 - ▲ Use proof of concepts to backup the architecture and document it







INTRODUCTION TO DOMAIN-DRIVEN DESIGN

- ▀ Domain-Driven Design (Eric Evans)
 - ▀ Focus on the core domain and domain logic
 - ▀ Model is the base of complex designs
 - ▀ Initiating a creative collaboration between technical and domain experts to iteratively the conceptual heart of the problem.

INTRODUCTION TO COMPONENT-BASED DESIGN



- ▴ Domain-Driven Design Concepts
 - ▴ Just in time modeling
 - ▴ Ubiquitous language
 - ▴ Bounded Contexts
 - ▴ Decoupling business logic from other aspects of the system
 - Separation Of Concernes
 - Cohesiveness

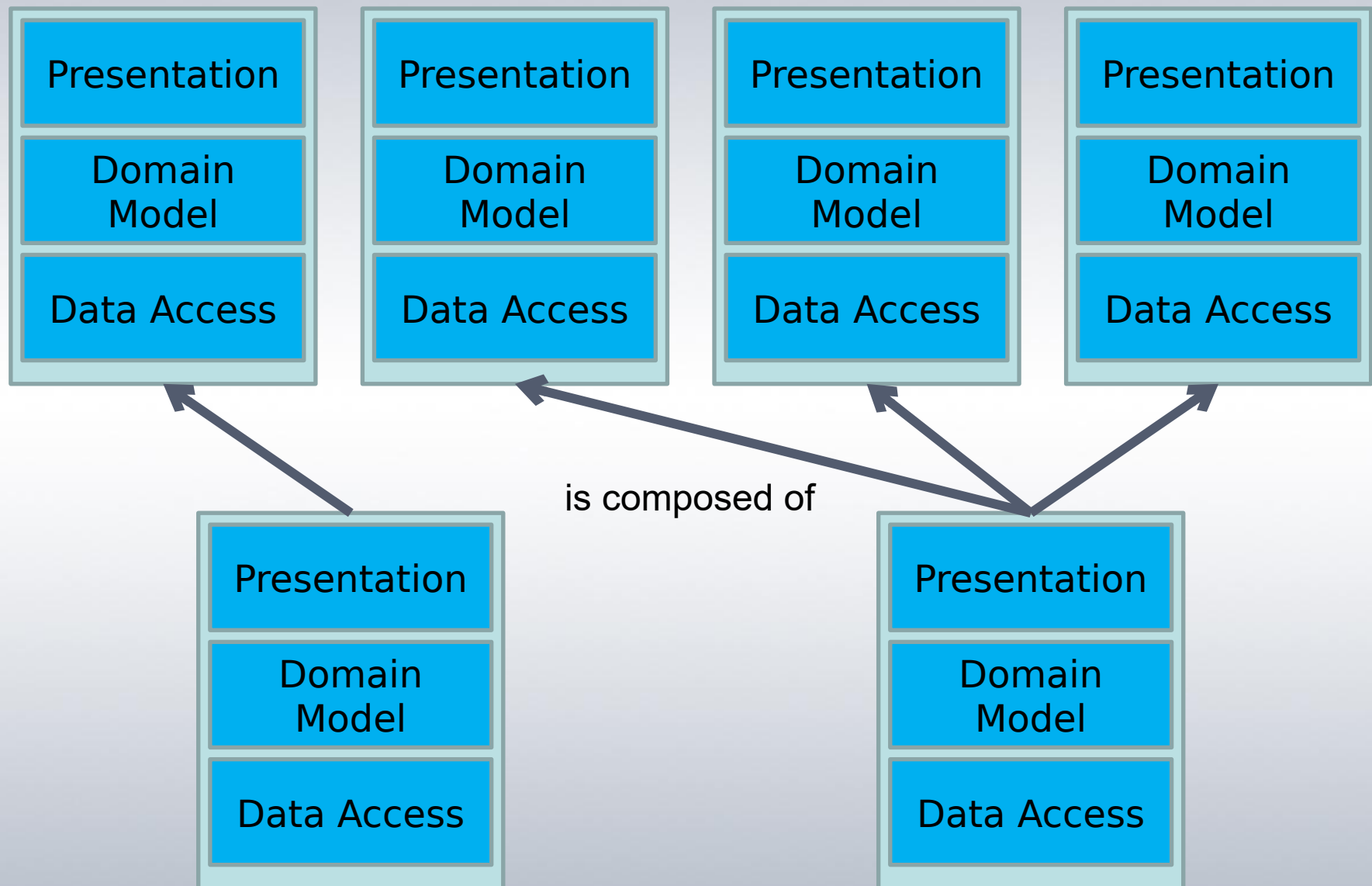
■ Component-Based Design

- An individual component is a software package, a web service, or a module that encapsulates a set of related functions (or data).
- All system processes are placed into separate components so that all of the data and functions inside each component are semantically related (just as with the contents of classes)
- Software as a circuit board

- ▶ Component-Based Design Concepts
 - ▶ Modularity
 - ▶ Reusability
 - ▶ Composition
 - ▶ Encapsulation
 - Separation Of Concerns
 - Cohesiveness

- ▶ Using Domain-Driven Design within a component
 - ▶ To create domain model and logic
 - ▶ Bounded Contexts
 - ▶ Anti-Corruption layers
 - ▶ Improve communication with UL

- ▶ Using Component-Based design to achieve
 - ▶ Modularity
 - ▶ Composition
 - ▶ Encapsulation and Decoupling
 - The component defines a bounded context
 - Enables us to focus on one problem at a time



CODING PRACTICES

- ▶ Test-Driven Development
 - ▶ Design tests BEFORE implementing a feature
 - ▶ Tests are testing the expected behavior of a component or class
 - ▶ Implementation is done step by step until all tests pass
 - ▶ Supports thinking of a design that can be tested

- ▶ Behavior-Driven Development
 - ▶ Get business specialists involved in testing!
 - ▶ Focuses on collaboration between developers, quality assurance and business users
 - ▶ Might be used for User Acceptance testing

▀ Version Control

- ▀ A version control system allows the user to simultaneously work on the code base and it automatically merges differences and reports conflicts.

▀ Continuous Integration

- ▀ Regular builds of the code base
- ▀ Early discovery of introduced bugs
- ▀ Automatic execution of test base
- ▀ Reduces Regression

▀ Refactoring

- ▀ Improve code quality without changing its behavior
- ▀ Code towards design patterns
- ▀ Consider separation of concerns and cohesiveness
- ▀ TDD, BDD, Version Control and CI are the foundation for refactoring (a safety-net), they reduce the fear of change

▴ Decoupling

▴ How?

- Inversion of control
- Dependency Injection
- Encapsulation

▴ Helps with

- Modularity
- Extensibility
- Maintainability
- Testability

- ▴ Design Patterns
 - ▴ Do not re-invent the wheel
 - ▴ Design Patterns provide solutions for common problems
 - ▴ Domain-Driven Design defines a set of architectural patterns

DEMO

▴ Intentionally Blank(ish)

- ▶ Agile Partner: www.agilepartner.net
- ▶ Agile Interest Group Luxembourg:
www.aiglu.org
- ▶ Agile Alliance: www.agilealliance.org
- ▶ Scrum alliance: www.scrumalliance.org
- ▶ Scrum.org

Thank You

Johannes
Täuber

Software Architect /
CSM

jtaeuber@agilepartner.net

[http://
www.taeuber.org](http://www.taeuber.org)