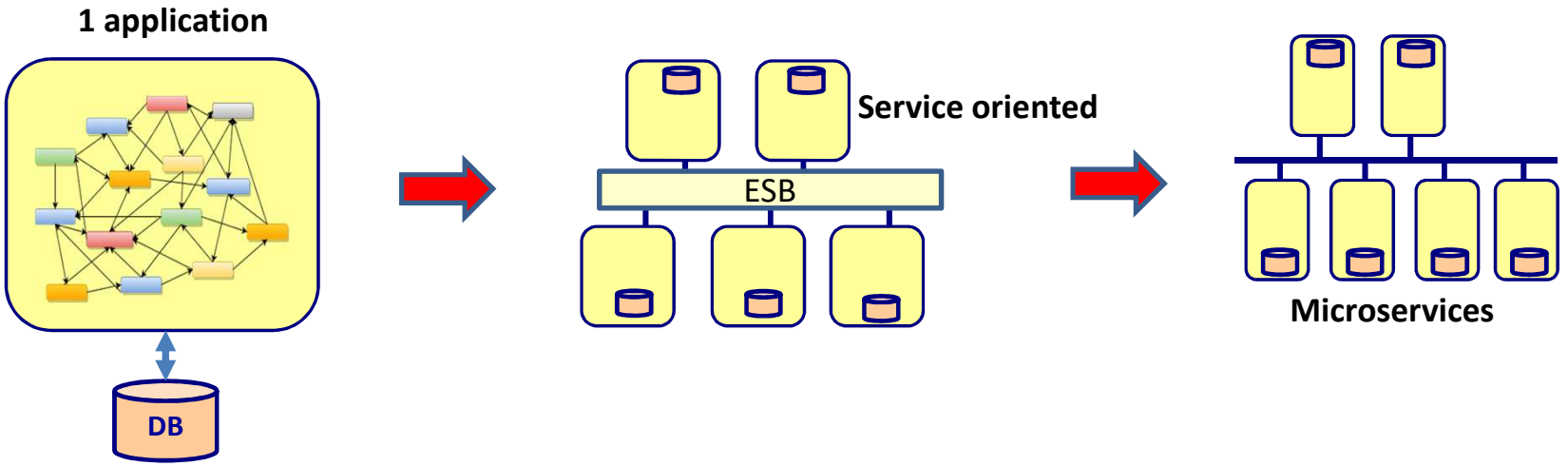
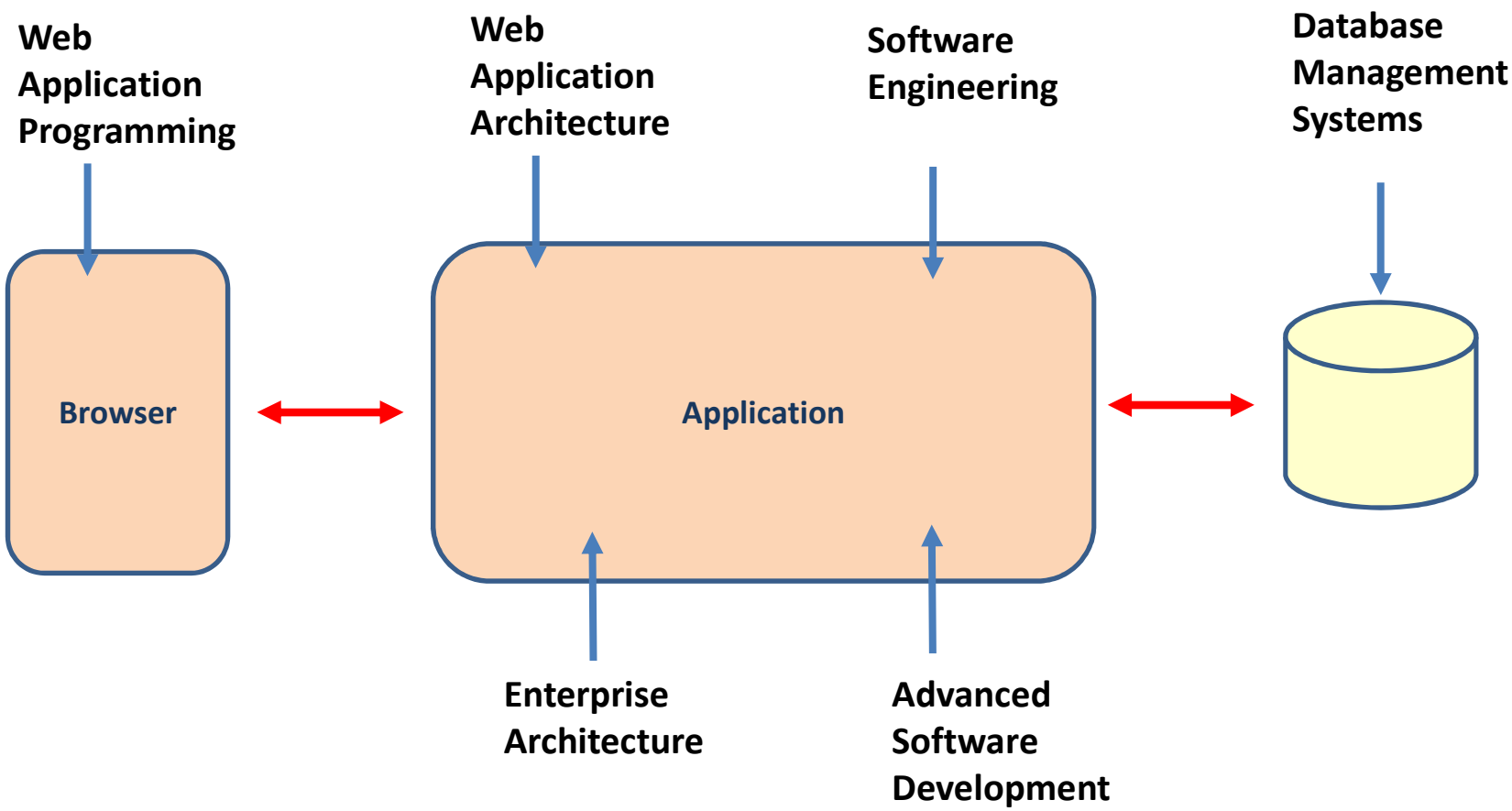
SOFTWARE ARCHITECTURE COURSE OVERVIEW

CS 590 Software Architecture

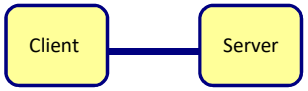
- Architecture styles
- Architecture patterns
- Architecture principles and best practices



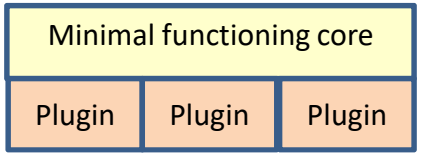
Relationship with other courses



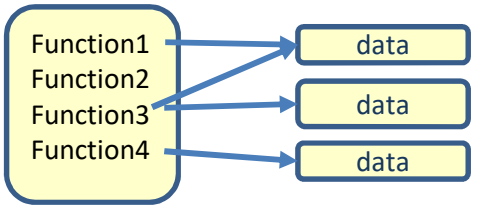
Architecture styles



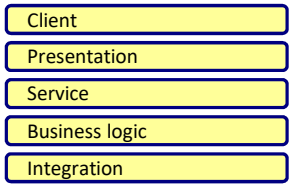
Client-server



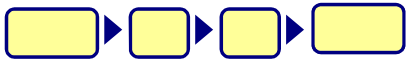
Microkernel



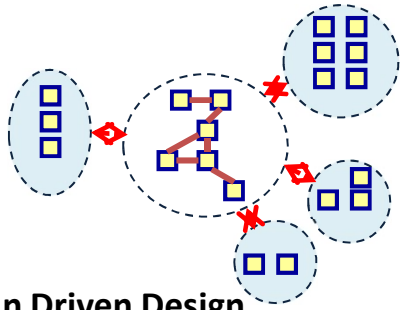
Procedural



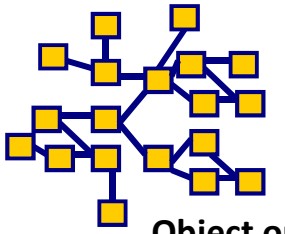
Layering



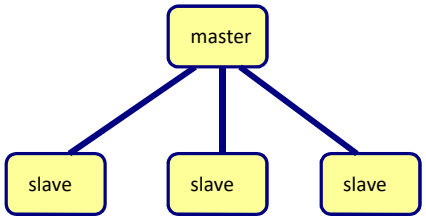
Pipe-and-Filter



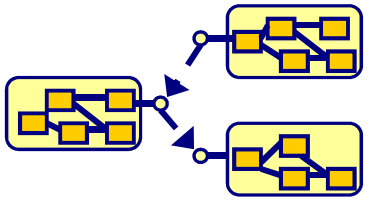
Domain Driven Design



Object oriented

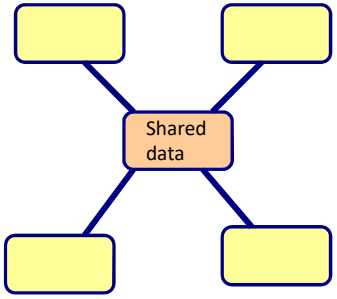


Master-Slave

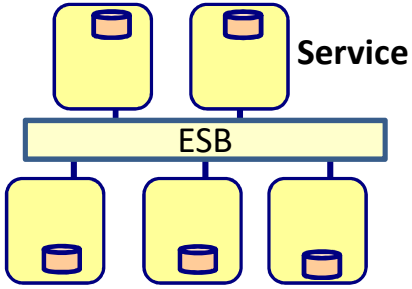


Component based

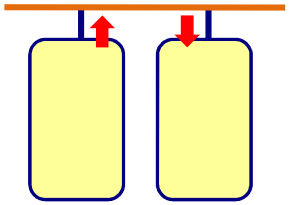
Architecture styles



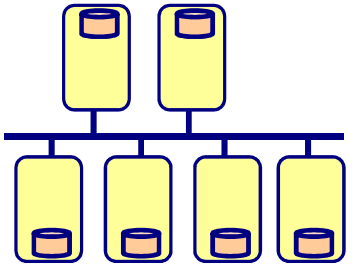
Blackboard



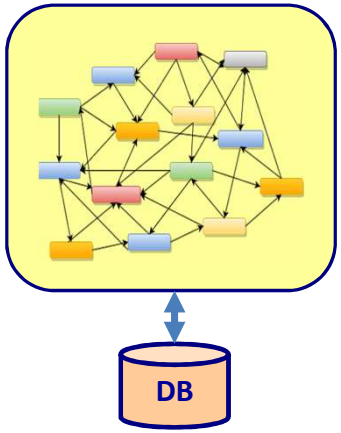
Service oriented



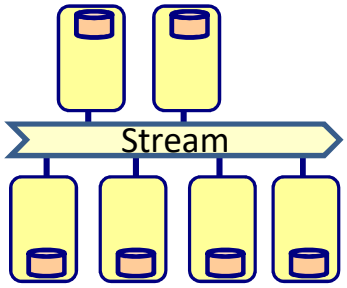
Event Driven



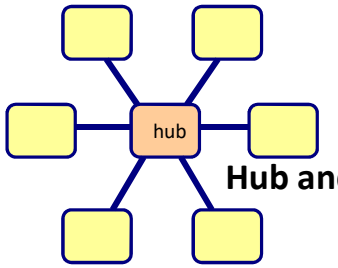
Microservices



Monolith



Stream based



Hub and Spoke

LESSON 1
SOFTWARE ARCHITECTURE INTRODUCTION

Why architecture?

More complexity asks for:

- More abstraction and decomposition
- More principles and guidelines
- More communication
- More proces
- More powerful tooling

More complexity asks for more architecture

Why architecture?

- Winchester “mistery” house
- 38 years of construction work– 147 builders 0 architects
- 160 rooms– 40 bedrooms, 6 litchens, 2 basements, 950 doors
- No architecture description
- 65 doors that don’t go anywhere, 13 stairs that don’t go anywhere, 24 skylights where you cannot see the sky

What is software architecture?

- 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

  - ANSI/IEEE std 1471-2000

What is architecture?

- Modules, connections, dependencies and interfaces
- The big picture
- The things that are expensive to change
- The things that are difficult to change
- Design with the bigger picture in mind
- Interfaces rather than implementation
- Aesthetics (e.g. as an art form, clean code)
- A conceptual model
- Satisfying non-functional requirements/quality attributes
- Ability to communicate (abstractions, language, vocabulary)
- A degree of rigidity and solidity
- A blueprint
- Systems, subsystems, interactions and interfaces
- Governance
- The outcome of strategic decisions
- Structure (components and interactions)
- Technical direction
- Strategy and vision
- Building blocks
- Standards and guidelines
- The system as a whole
- Tools and methods
- A path from requirements to the end-product
- Guiding principles
- Technical leadership
- The relationship between the elements that make up the product
- Foundations
- An abstract view
- The decomposition of the problem into smaller implementable elements
- The skeleton/backbone of the product

What is architecture?

Shared understanding of the system

The decisions that are hard to change

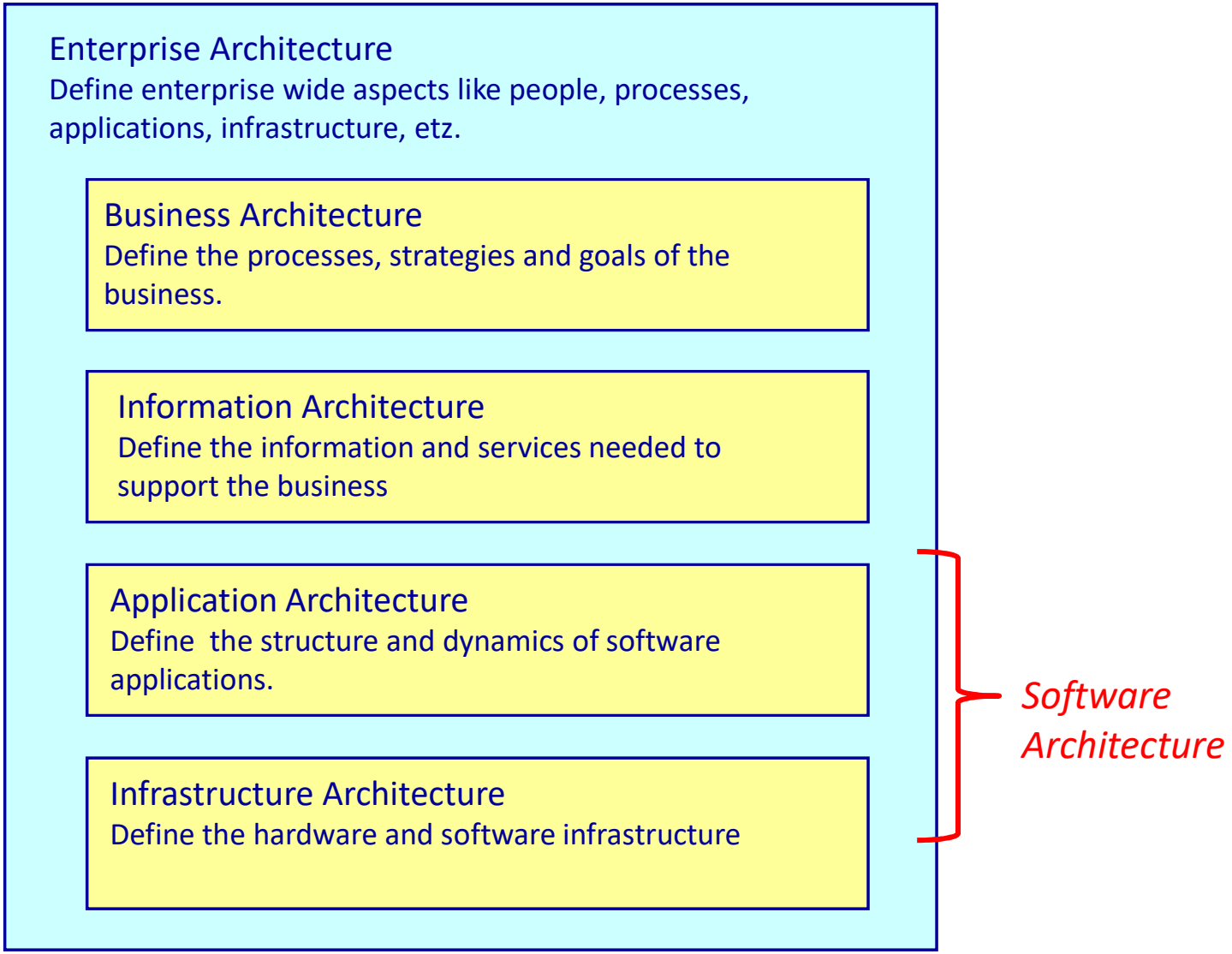
The important stuff

Whatever that might be

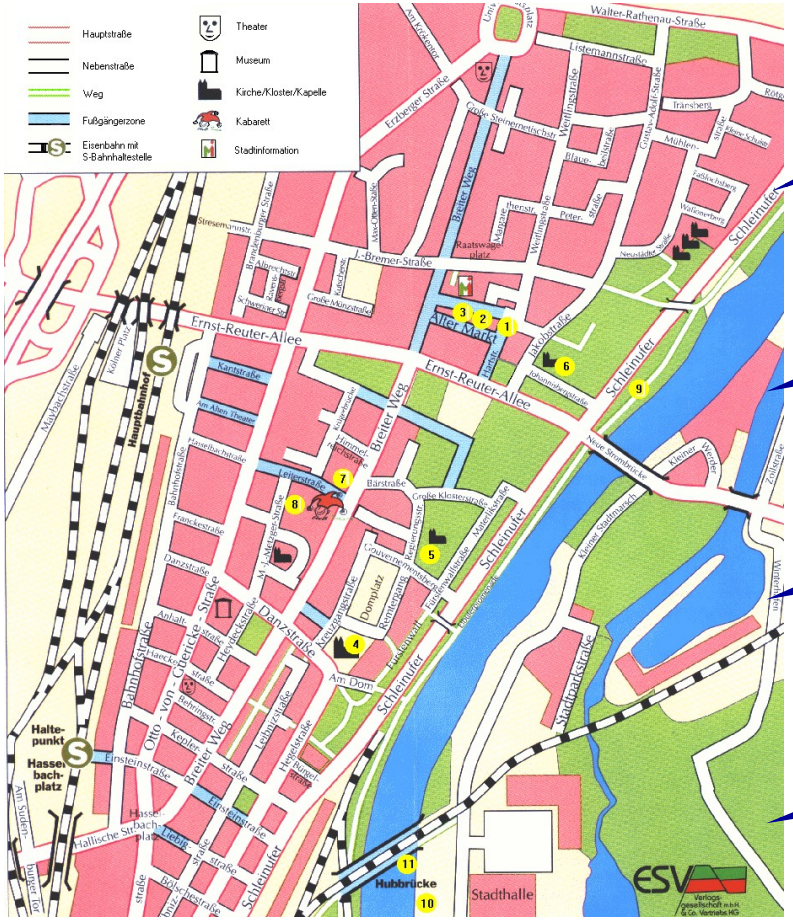
Different kinds of architecture

- Infrastructure
- Security
- Technical
- Solution
- Network
- Data
- Hardware
- Enterprise
- Application
- System
- Integration
- IT
- Database
- Information
- Process
- Business
- Software

Different kinds of architecture



City planning



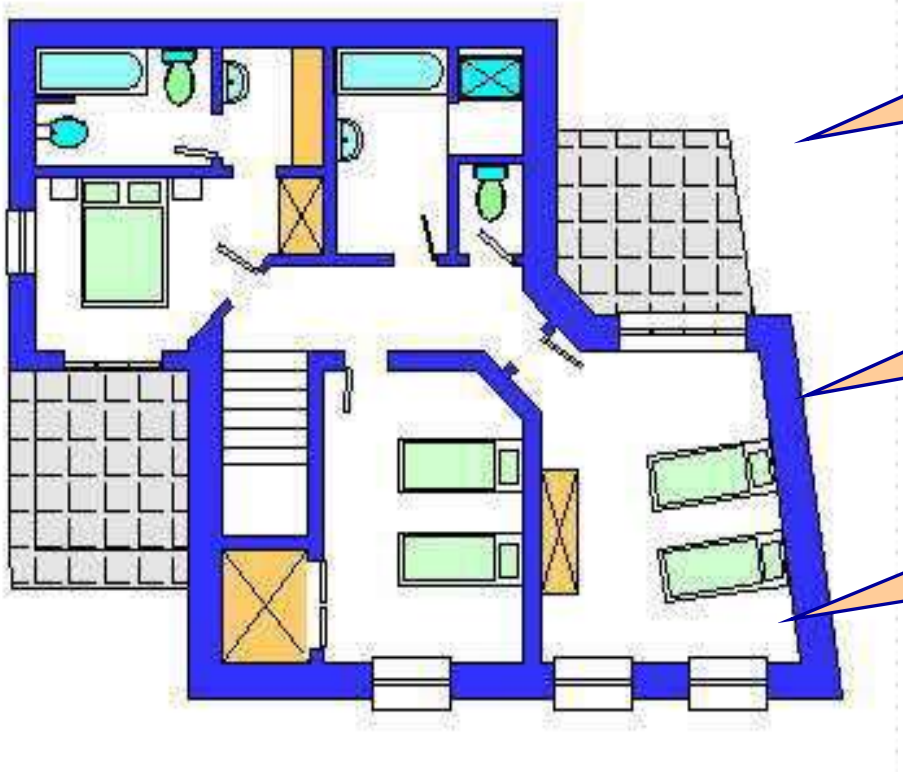
Business Architecture :
Goals of the city and the processes in a city

Information Architecture:
healthcare, education, water, electricity, transport, …

Application Architecture :
hospital, police, library, schools, …

Infrastructure Architecture :
roads, railroads, harbour, airport, …

House planning



Information Architecture:
water, electricity, heating, …

Application Architecture :
Living room, bedrooms, kitchen, …

Infrastructure Architecture :
Electrical wires, water pipes, driveway,…

Warship Vasa

- Customer
  - King of Sweden
    - Gustav II Adolf
- Requirements:
  - 70 m long
  - 300 soldiers
  - 64 guns
  - 2 decks
- Architect
  - Hendrik Hybertson

Software architecture is hard!

- Complexity
  - No physical limitations
  - Huge state-space
- Constant change of
  - Business
  - Technology
- The architecture is never ideal

The work of an architect: Make non-optimal decisions in the dark.

Defining the architecture

- Define components
- Define component interfaces
- Define platform and language(s)
- Define architectuur styles
- Define architectuur patterns
- Define layers and packages
- Define presentation architecture
- Define persistency architecture
- Define security architecture
- Define transaction architecture
- Define distribution architecture
- Define integration architecture
- Define the deployment architecture
- Define the clustering architecture
- Define the hardware
- Define tools to use

- Decide on solutions for
  - Logging
  - Error management
    - Error detection
    - Error reporting
    - Fault tolerance
  - Event management
  - File handling
  - Printing
  - Reporting
  - Resource management
  - Internationalization
  - Licence management
  - Debugging
  - …

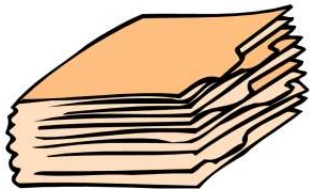
Main point

- Software architecture is based on architecture styles, principles and patterns
- The human physiology has the same structure as the structure of the Veda and Vedic literature who are expressions of the structure of pure consciousness.

AGILE ARCHITECTURE

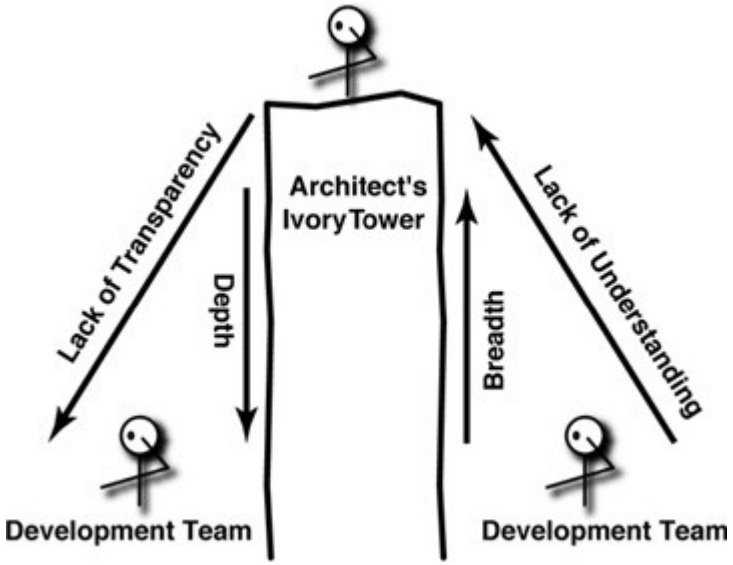
Traditional architecture

- The architect designs the system
  - Big upfront architecture
    - Large Software Architecture Document (SAD)
  - Ivory tower architect
    - The architecture is not understood and implemented by the developers
    - The architect does not understand the current technology
  - The architect is only available in the beginning of the project



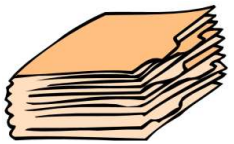
Ivory tower architect

- It is very hard to truly know the best solutions for design problems if you are not working (coding) on the project
- It takes many iterations of a solution before it finally works - so you can't suggest a solution and then leave



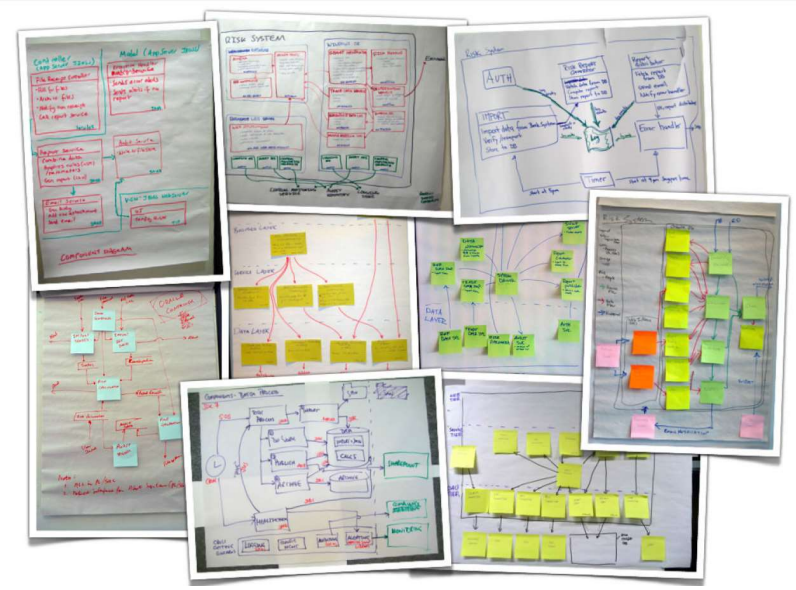
Documenting architecture

- Traditional process
  - Software Architecture Document (SAD)



Software Architecture Document

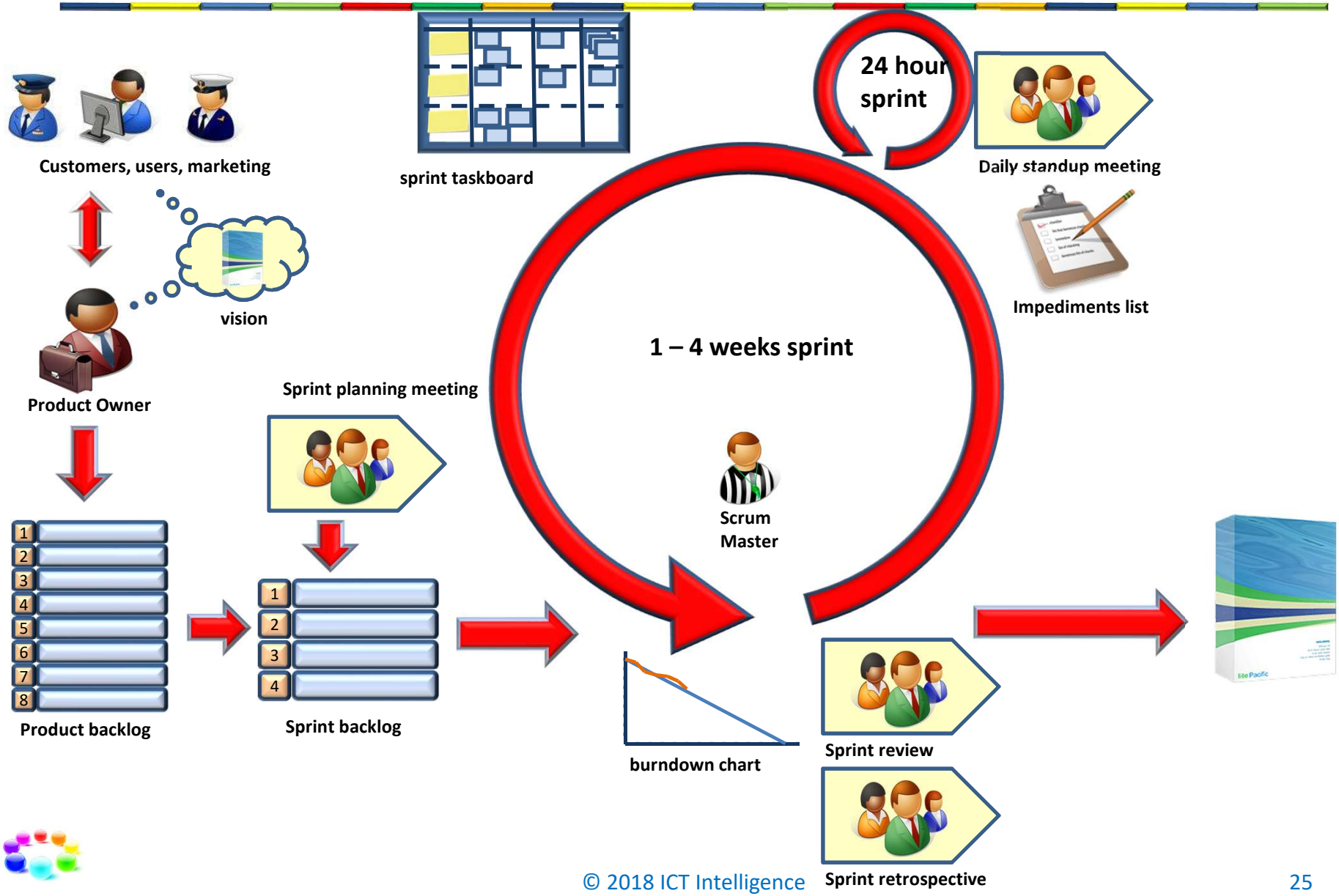
- Agile process
  - 4C model



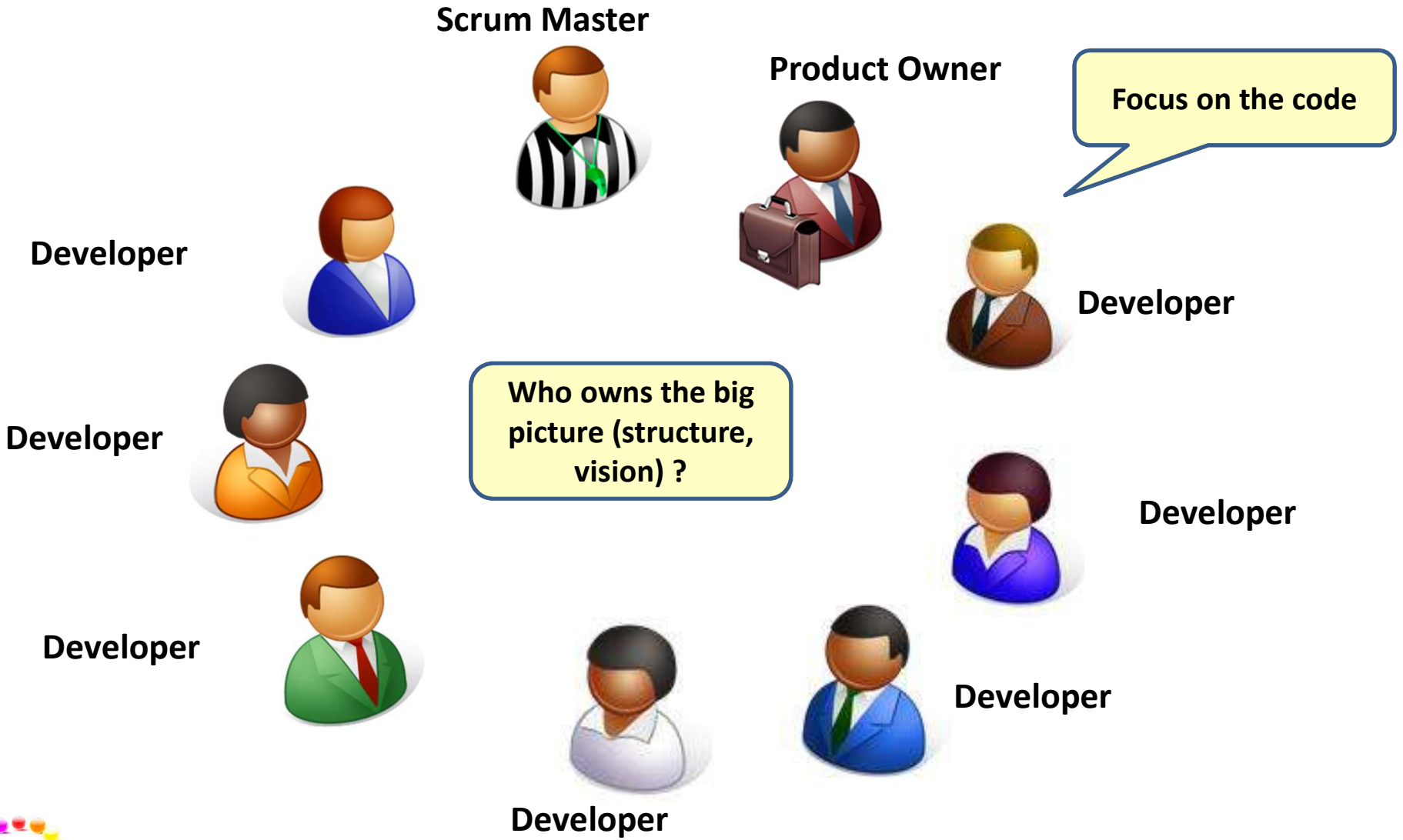
Agile architecture

- Just enough upfront architecture
  - Refine the architecture later
- Keep your architecture flexible
- The architect is available during the whole project
- The architect also writes code
  - But not all the time
  - The architecture is grounded in reality
  - Works together with the developers
  - Architects should be master builders
- Proof the architecture in the first iteration(s)

Scrum



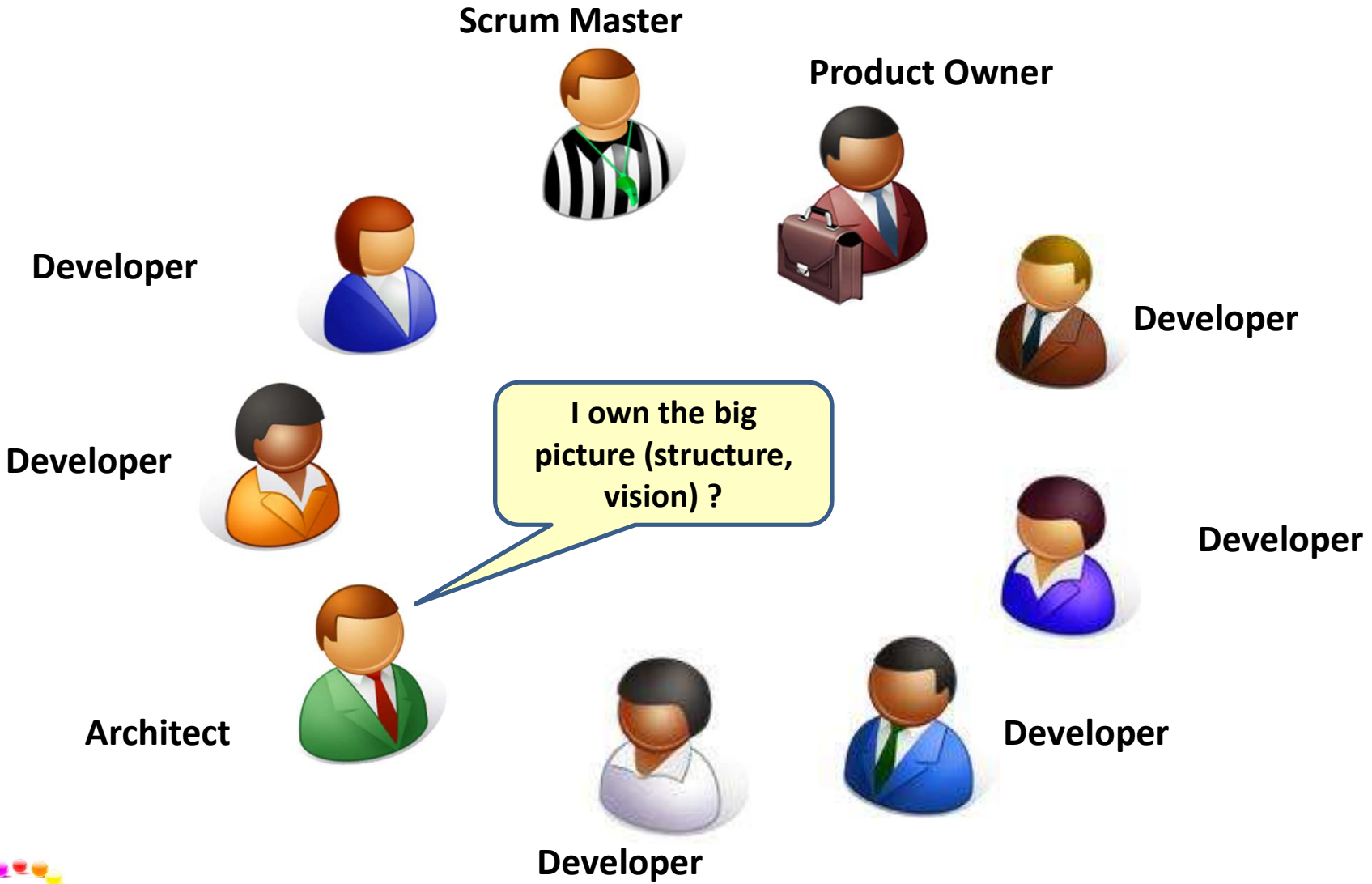
Scrum team


Scrum Master
Product Owner
Focus on the code
Developer
Developer
Developer
Who owns the big picture (structure, vision) ?
Developer
Developer
Developer
Developer

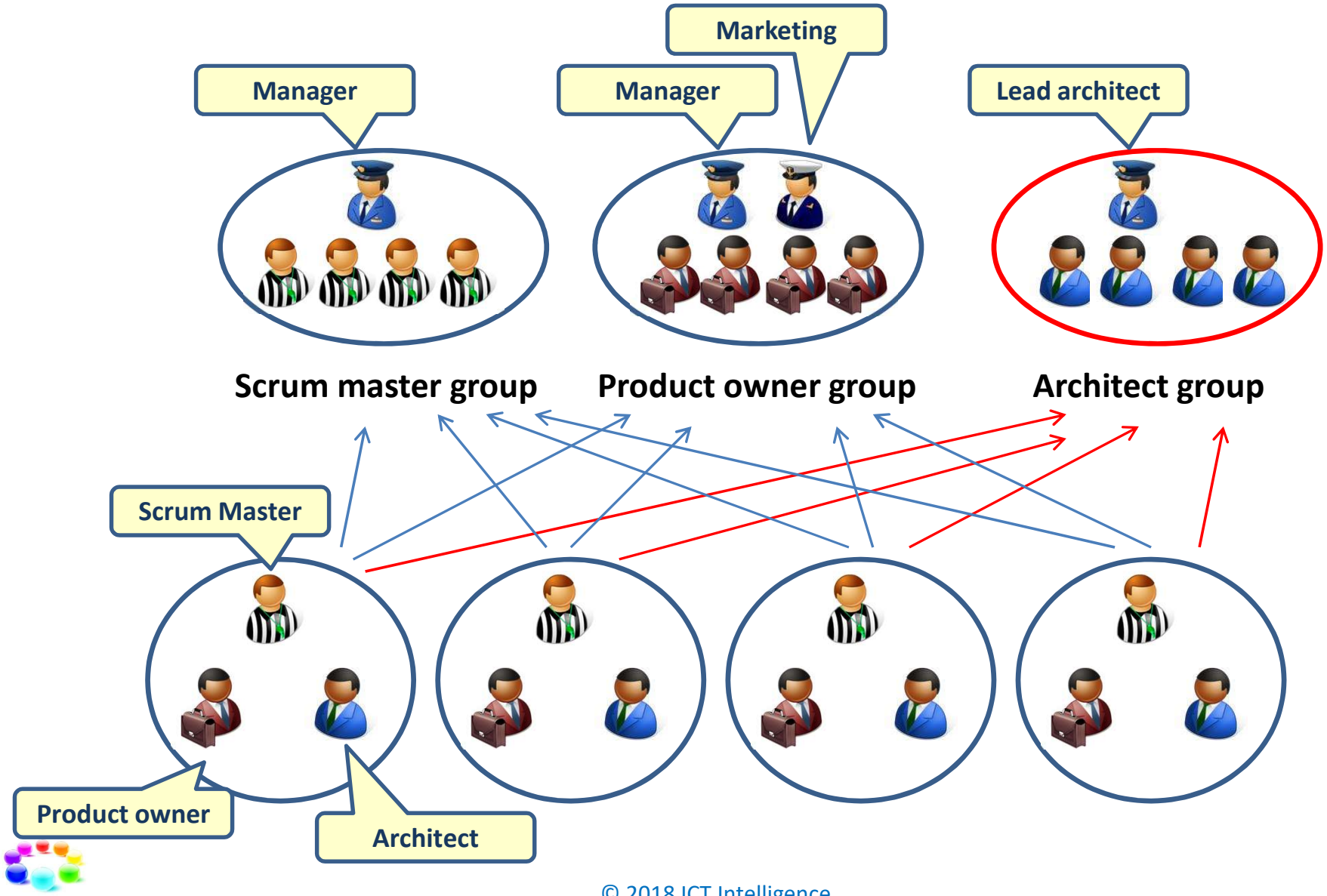
Why software architecture in agile?

- We need a clear vision and roadmap for the team to follow.
- We need to identifying and mitigating risk.
- We have to communicate our solution at different levels of abstraction to different audiences.
- We need technical leadership
- We have to make sure our architecture is consistent, correct and fits within the context

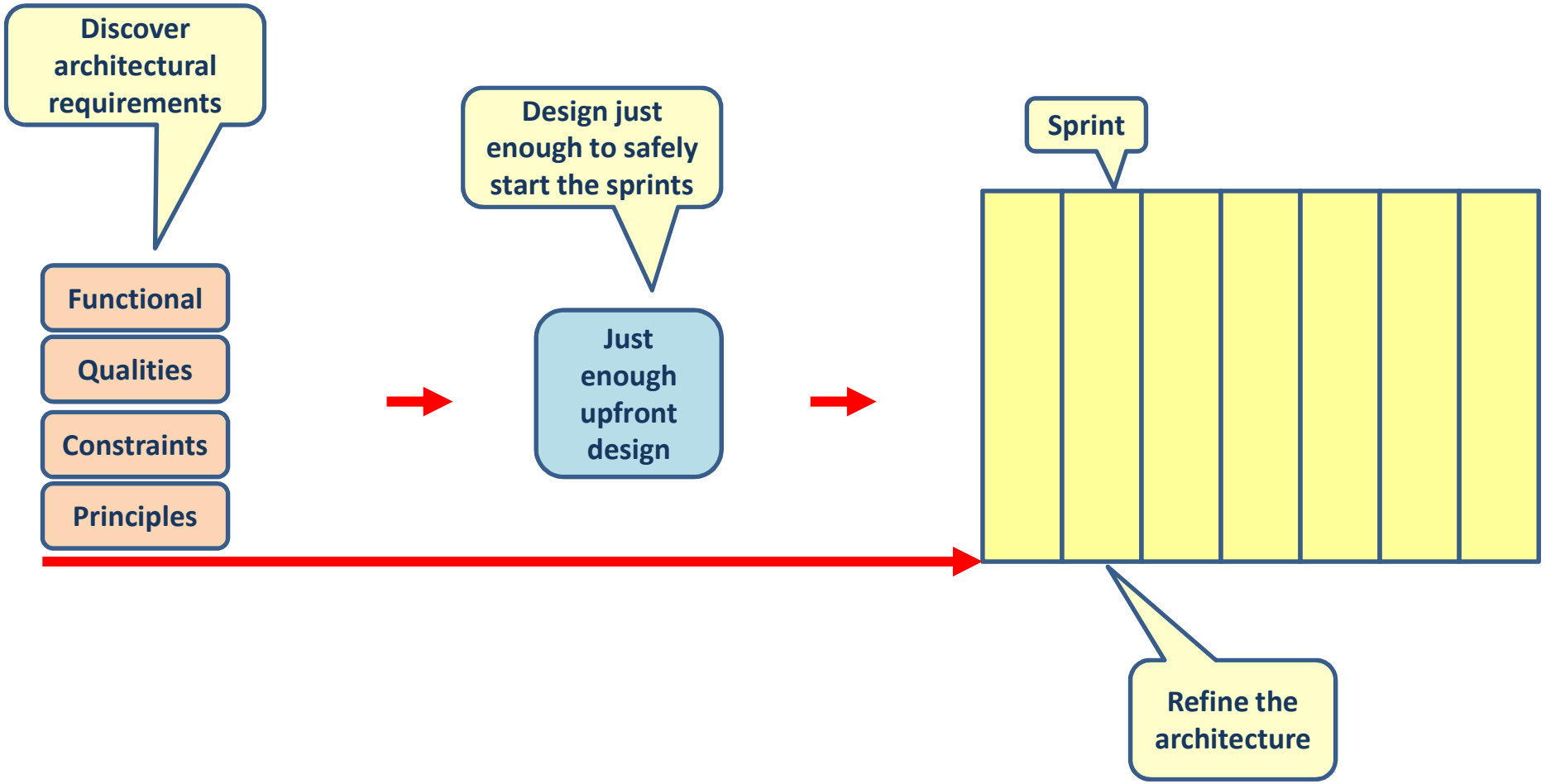
You need an architect

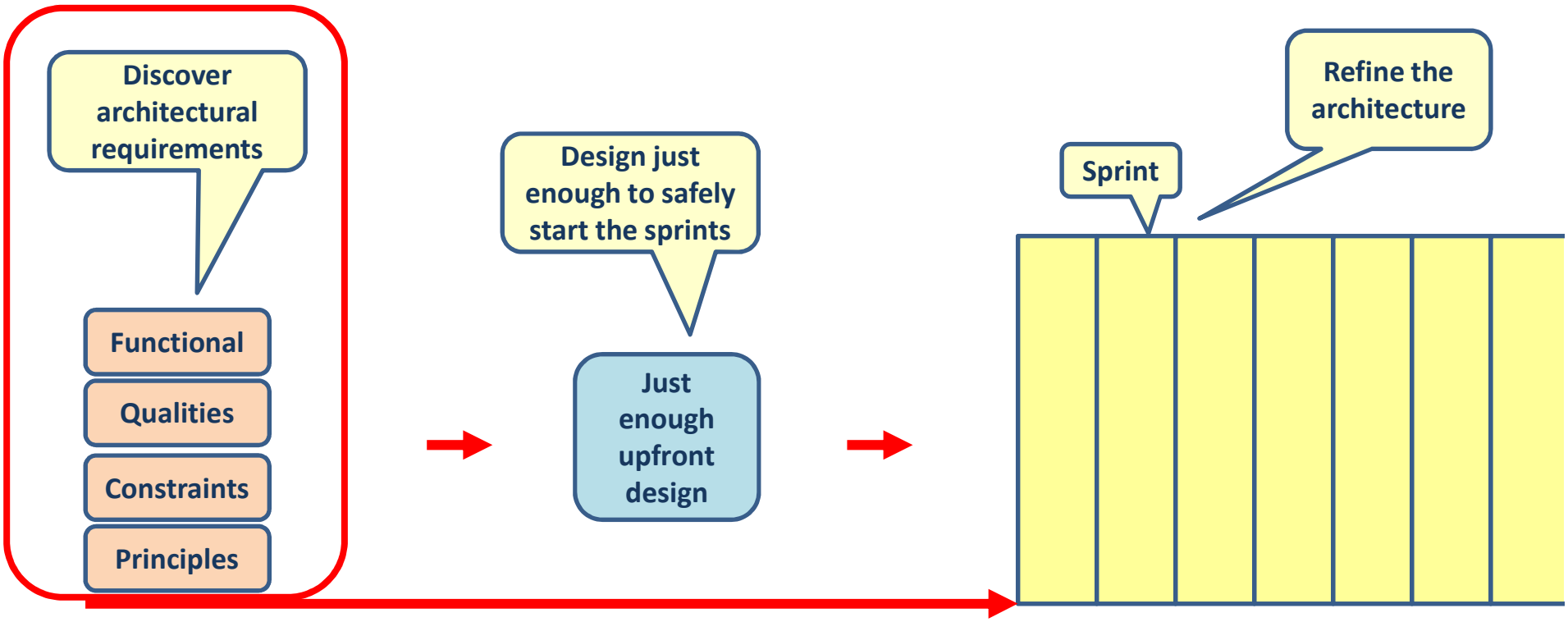


Architect group



Agile architecture





ARCHITECTURAL REQUIREMENTS

Functional requirements

- What should the system functionally do?
  - Use cases
  - User stories

As a customer

I can view my account history

so that I know all transactions on my account

- **Functional**
- **Qualities**
- **Constraints**
- **Principles**

Architectural constraints

- Constraints from the business (or enterprise architecture
  - We do everything in .Net
  - We always use an Oracle database
  - Our maintenance engineers all know Java
  - All applications talk with the Oracle ESB
- Budget
- Deadlines

SOFTWARE QUALITIES

NFR characteristics

- Which qualities are available?
- We need to balance the qualities
- A quality itself is not precise enough
- Stakeholders have different interests

SEI quality model

Qualities noticeable at runtime

Performance	**Responsiveness of the system**
Security	**Ability to resist unauthorized usage**
Availability	**Portion of time the system is available**
Functionality	**Ability to do intended work**
Usability	**Learnability, efficiency, satisfaction, error handling, error avoidance**

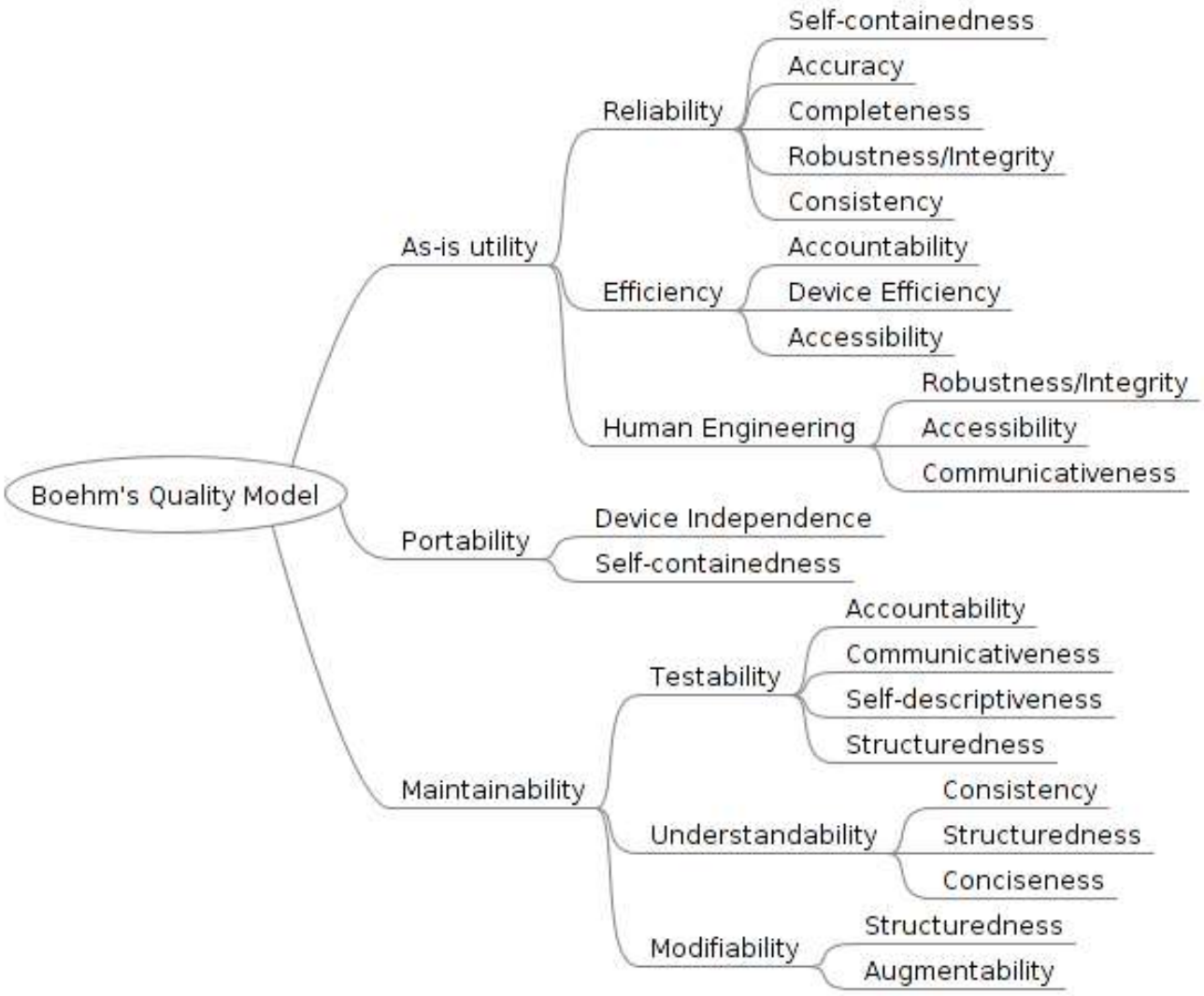
Qualities not noticeable at runtime

Modifiability	**Cost of introducing change**
Portability	**Ability to operate in different computing environments**
Reusability	**Ability to reuse components in different applications**
Integrability	**Ability that components work correctly together**
Testability	**Ability to systematic testing to discover defects**

FURPS model

- **Functionality** - evaluate the feature set and capabilities of the program, the generality of the functions delivered and the security of the overall system
- **Usability** - consider human factors, overall aesthetics, consistency, and documentation
- **Reliability** - measure the frequency and severity of failure, the accuracy of outputs, the ability to recover from failure, and the predictability
- **Performance** - measure the processing speed, response time, resource consumption, throughput and efficiency
- **Supportability** - measure the maintainability, testability, configurability and ease of installation

Boehm



ISO 25010

NFR characteristics

- Which qualities are available?
- We need to balance the qualities
- A quality itself is not precise enough
- Stakeholders have different interests

Balance the qualities

- More security through encription lowers performance
- More scalability through clustering lowers performance
- More scalability through clustering increases the cost

Find the top 5(+/- 2) qualitities

NFR characteristics

- Which qualities are available?
- We need to balance the qualities
- A quality itself is not precise enough
- Stakeholders have different interests

Quality scenario’s

- A quality on itself has little meaning
- Create scenario’s for the top qualitities
- Make scenario’s measurable
  - The should be able to scale to 1000 concurrent users
  - The system should be available 24/7
  - All user actions should give a response within 3 seconds.
- Prioritize the scenario's
- Write acceptance tests for NFR scenario’s

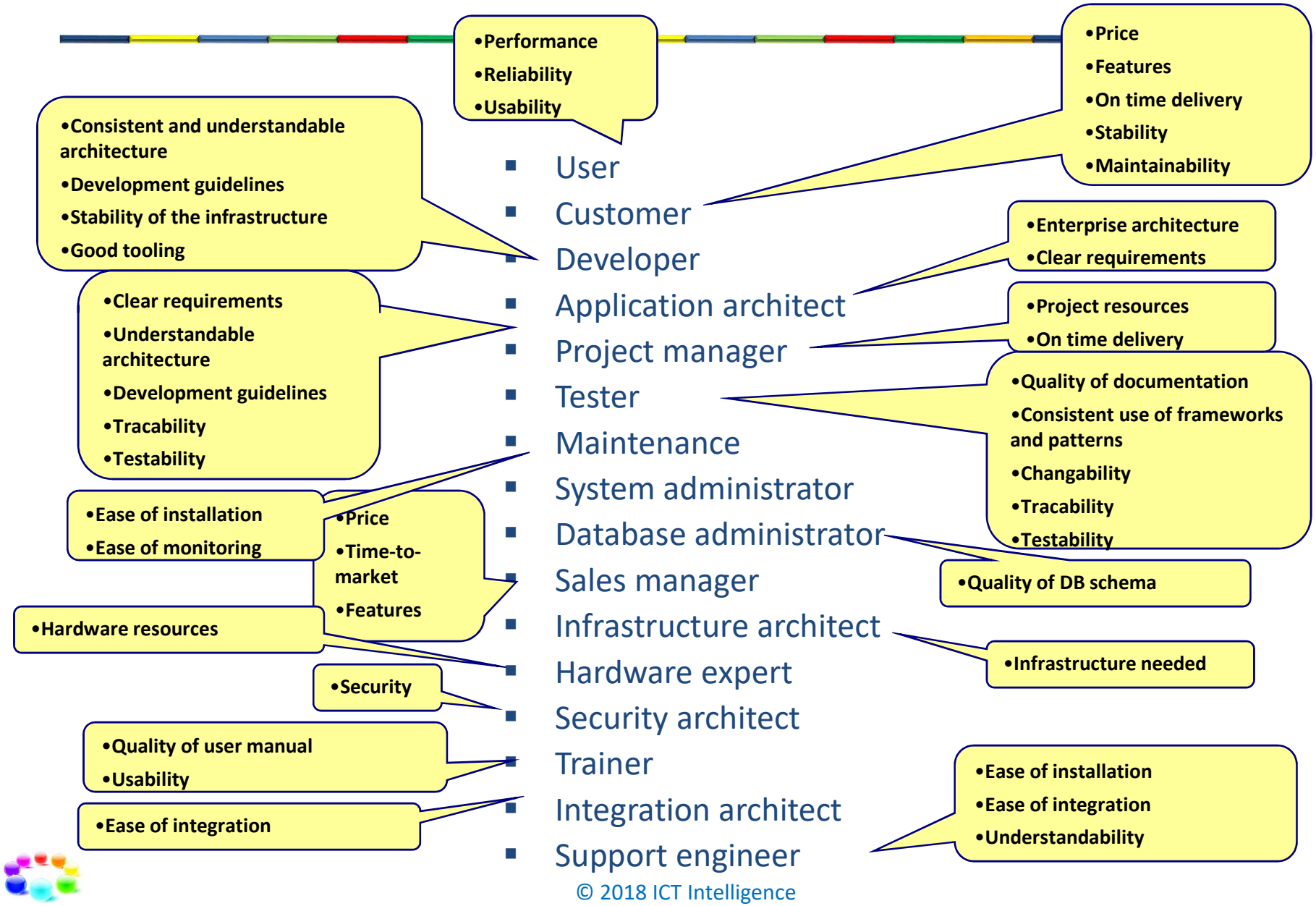
NFR characteristics

- Which qualities are available?
- We need to balance the qualities
- A quality itself is not precise enough
- Stakeholders have different interests

Stakeholders

- User
- Customer
- Developer
- Application architect
- Project manager
- Tester
- Maintenance
- System administrator
- Database administrator
- Sales
- Infrastructure architect
- Hardware expert
- Security architect
- Trainer
- Integration architect
- Support engineer

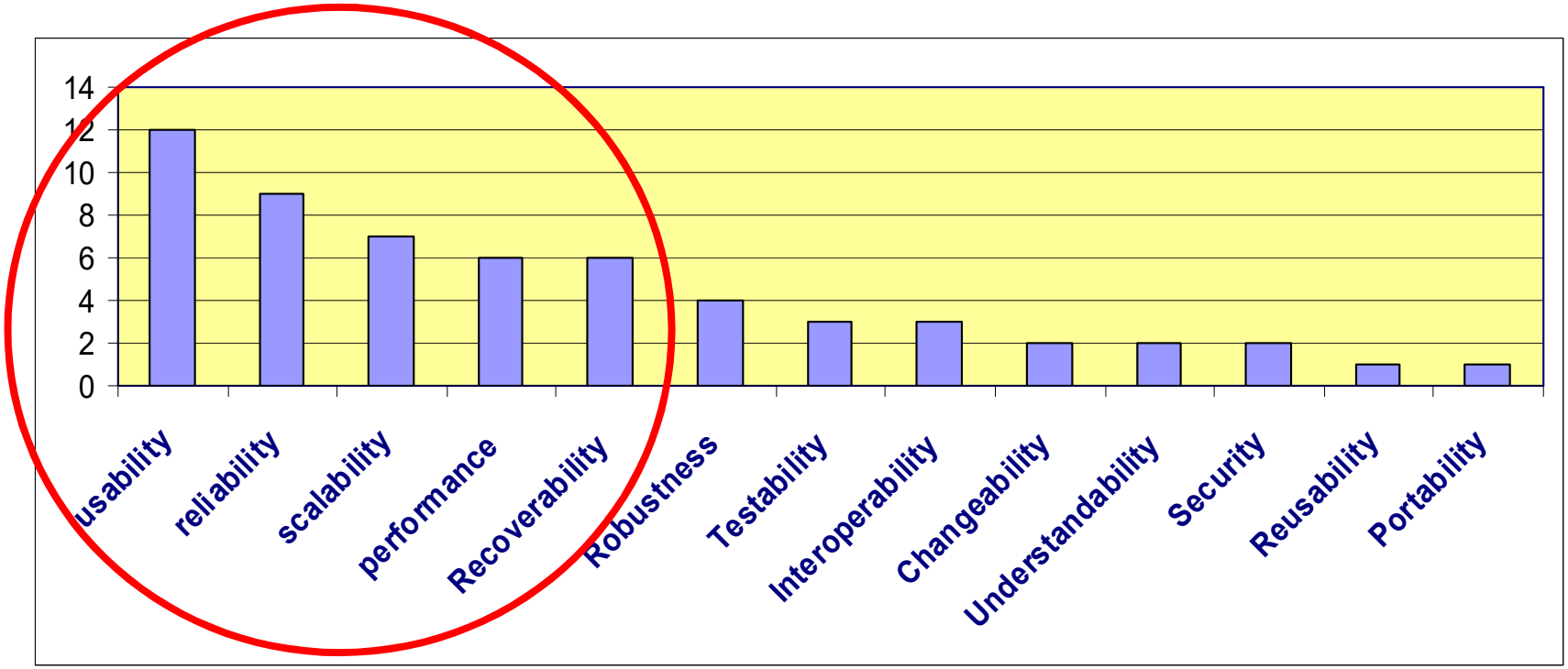
Stakeholders and their interest



Quality workshop

- Goal:
  - Find the prioritized scenario's of the top qualities.
  - Communicate the qualities between stakeholders
- Workshop agenda
  - Explain the vision of the system
  - Explain the different qualitities
  - Everone votes (everyone gets $10 to divide)
  - Discuss the result
  - Vote again
  - Create scenario's for the top qualitites
  - Prioritice the scenario's (vote)

Qualitity workshop result



Most important qualities

- Performance
- Scalability
- Availability
- Reliability
- Maintainability
- Security
- Interoperability
- Usability

Performance

- The responsiveness of the application to perform specific actions in a given time span.
  - Latency
    - The time taken by the application to respond to an event.
  - Throughput
    - The number of events scored in a given time interval.

- Scenario’s
  - All actions must respond in 3 seconds
  - Complex actions must respond in 5 seconds

Scalability

- The ability to handle an increase in the workload without impacting the performance
- Scenario's
  - The system should scale up to 100.000 users in a year
  - The system should be able to handle 5.000 concurrent users

Availability

- The probability that the system is operating properly when it is requested for use
- Calculated as uptime/total time
- Scenario's
  - The critical part of the system should be available 99.5% of the time (24 * 7)
  - The non critical part of the system should be available 98.3% of the time (24 * 7)

Reliability

- The probability that the system is operating properly for a desired period of time without failure.
- Mean Time Between Failure (MTBF)
  - Total time in service / number of failures
- Scenario's
  - Mean Time Between Failures = 100.000 hours
  - Mean Time To Recovery = 10 hours
  - Data is consistent all the time

Availability vs. Reliability

- High reliability contributes to high availability
- But you can achieve high availability with an unreliable system, if the recovery time is low.

Maintainability

- The ability of any application to go through modifications and updates with a degree of ease.
- Scenario’s
  - The system should be highly configurable
  - The system should have 80% test coverage
  - All errors should be logged

Security

- 3 aspects
  - Authentication: are you who you say you are?
  - Authorization: what are you allowed to do?
  - Secrecy: encrypt the data
- Scenario's
  - Only authorized users may access the system
  - All secret data that is sent to other systems should be encrypted

Interoperability

- The ability to exchange information and communicate with internal and external applications and systems.
- Scenario's
  - All communication with external systems should use standards like SOAP, REST or messaging.
  - All interfaces should be defined and published.

Usability

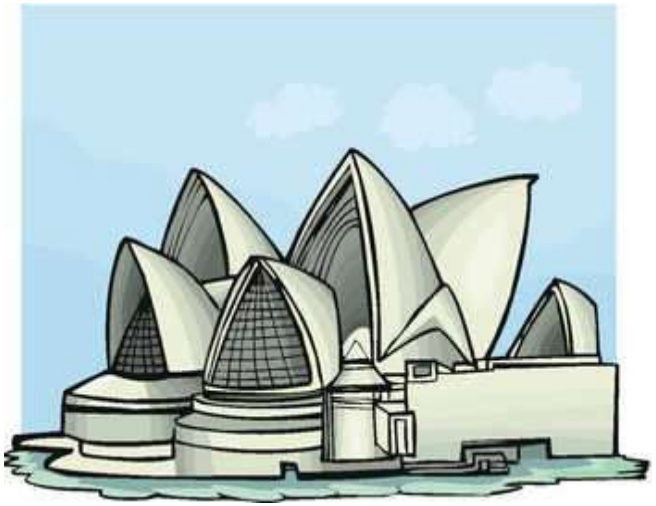
- The ease at which the users operate the system and make productive use of it.
- Aspects:
  - Efficiency: How efficiently can a end user perform a task once they have learned the UI design
  - Learnability: Ease for end users to accomplish primitive processes the first time they encounter the UI.
  - Memorability: When users return to the system after a time frame, how quickly can they re-establish expertise
  - Errors: How many errors do end users make, how severe are these, and how easily can they recover from these errors?
  - Satisfaction: How pleasing is it to use the system
- Scenario's
  - The system should support the following browsers: Internet Explorer, Chrome, Mozilla Firefox, Safari, Netscape
  - The user can change the language to English, Chinese, French and Spanish.

ARCHITECTURE PRINCIPLES

Architecture (design) principles

- Keep it simple
- Keep it flexible
- Loose coupling
- Separation of concern
- Information hiding
- Principle of modularity
- High cohesion, low coupling
- Open-closed principle

Keep it simple



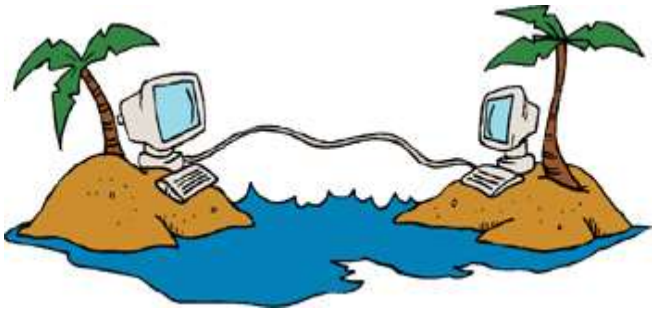
- The more complexity, the more change on failure
- Simple applications, frameworks, tools, etc. remain to be used
  - Complex ones will be replaced by something simple
- Gold plating

Keep it flexible

- Everthing changes
  - Business
  - Technical
- More flexibility leads to more complexity

Loose coupling

- Different levels of coupling
  - Technology
  - Time
  - Location
  - Data structure
- You need coupling somewhere
  - Important is the level of coupling



Separation of concern

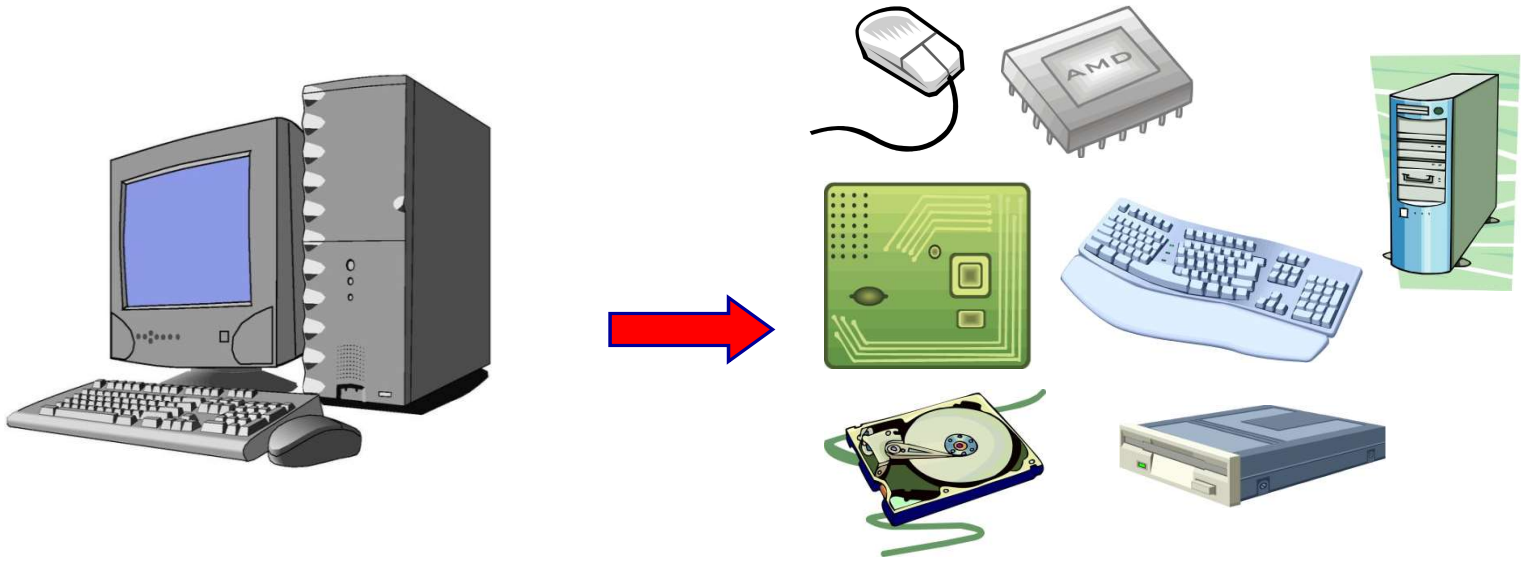
- Separate technology from business
- Separate stable things from changing things
- Separate things that need separate skills
- Separate business process from application logic
- Separate implementation from specification

Information hiding

- Black box principle
- Hide implementation behind an interface
- Hide the data structure behind stored procedures
- Hide the data structure behind business logic

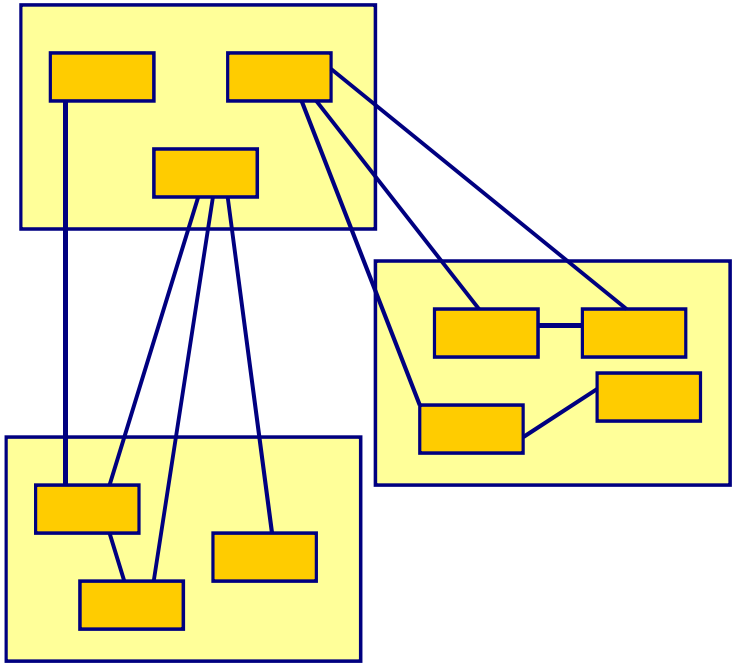
Principle van modularity

- Decomposition
- Devide a big complex problem is smaller parts
- Use components that are
  - Better understandable
  - Independent
  - Reusable
- Leads to more flexibility
- Makes finding and solvings bugs easier

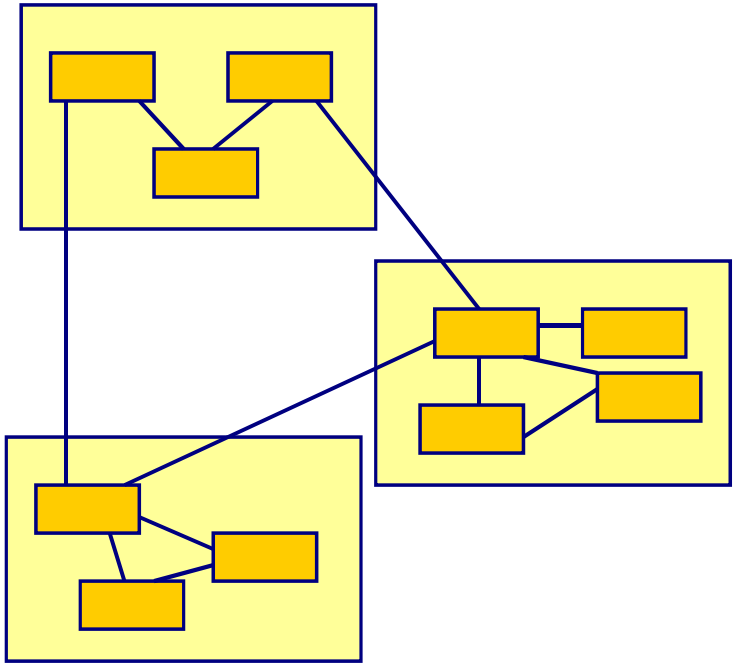


High cohesion, low coupling

- High coupling, low cohesion



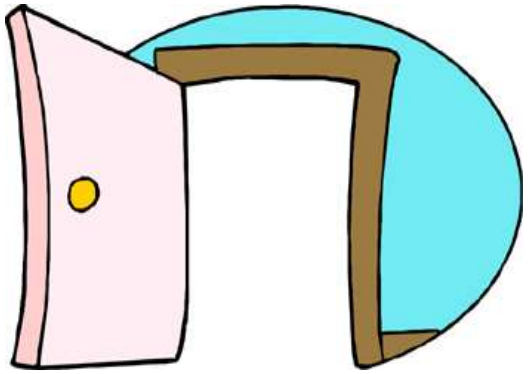
- High cohesion, low coupling



Open- closed principle

- The design should be “open” for extension, but “closed” for change.

- You want to add new functionality instead of changing existing, working and tested code.



Abstraction

- Focus on only the important aspects.
- Interface
- Abstract class

Library customer:
name
address
phone
age

Shopping customer:
name
shipping address
phone
credit card number
billing address

Hospital customer:
name
address
phone
insurance company
birth date

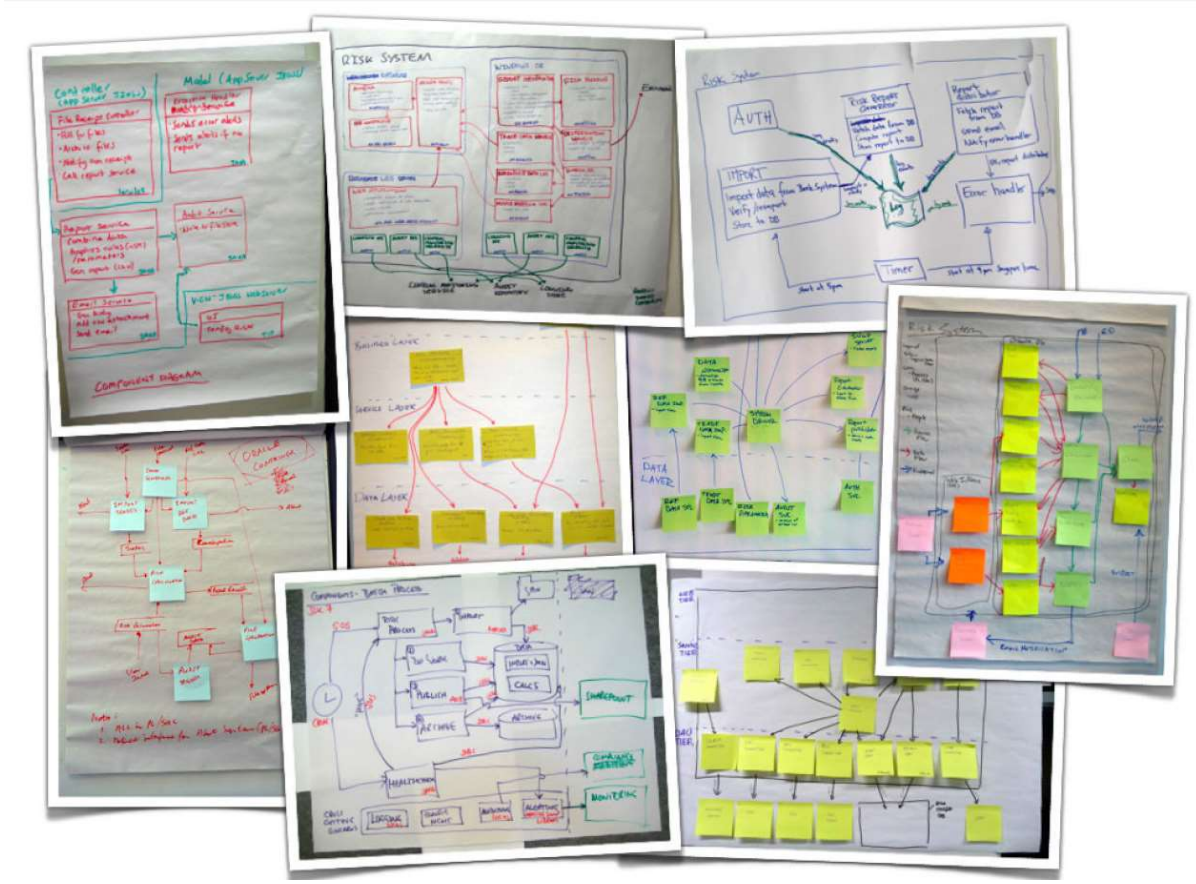
Main point

- Software architecture is never ideal. We have to find the right balance between the different software qualities and architecture principles
- Nature always takes the path of least resistance so that the perfection of the unified field can express itself in the relative creation

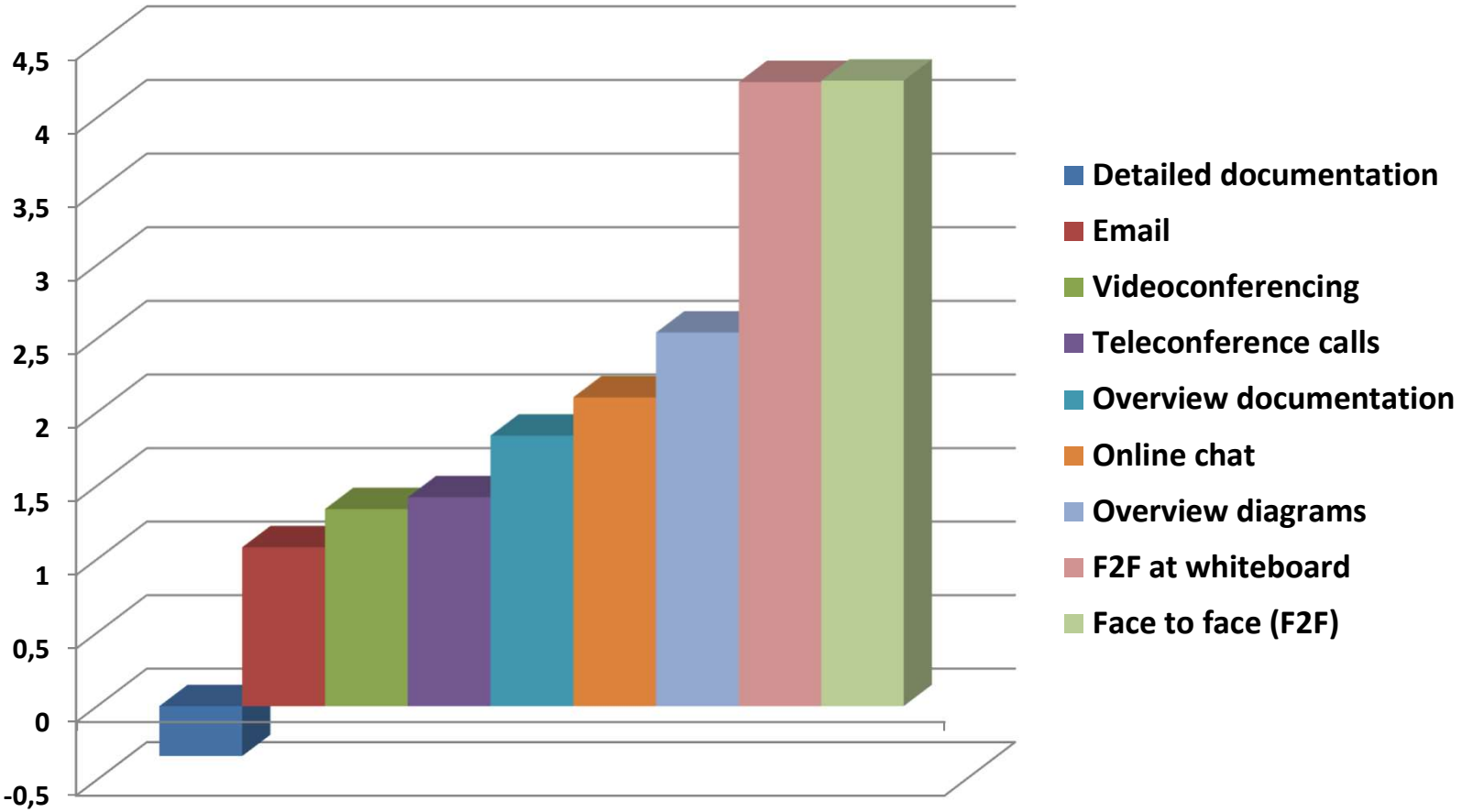
COMMUNICATING ARCHITECTURE

Architecture visibility

- Make the architecture visible with diagrams
  - Information radiator



Effectiveness of communication



Structure and Vision

Context diagram: Shows the big picture

↓

Container diagram: shows the high-level shape of the software architecture and how responsibilities are distributed across it. It also shows the major technology choices and how the containers communicate with one another.

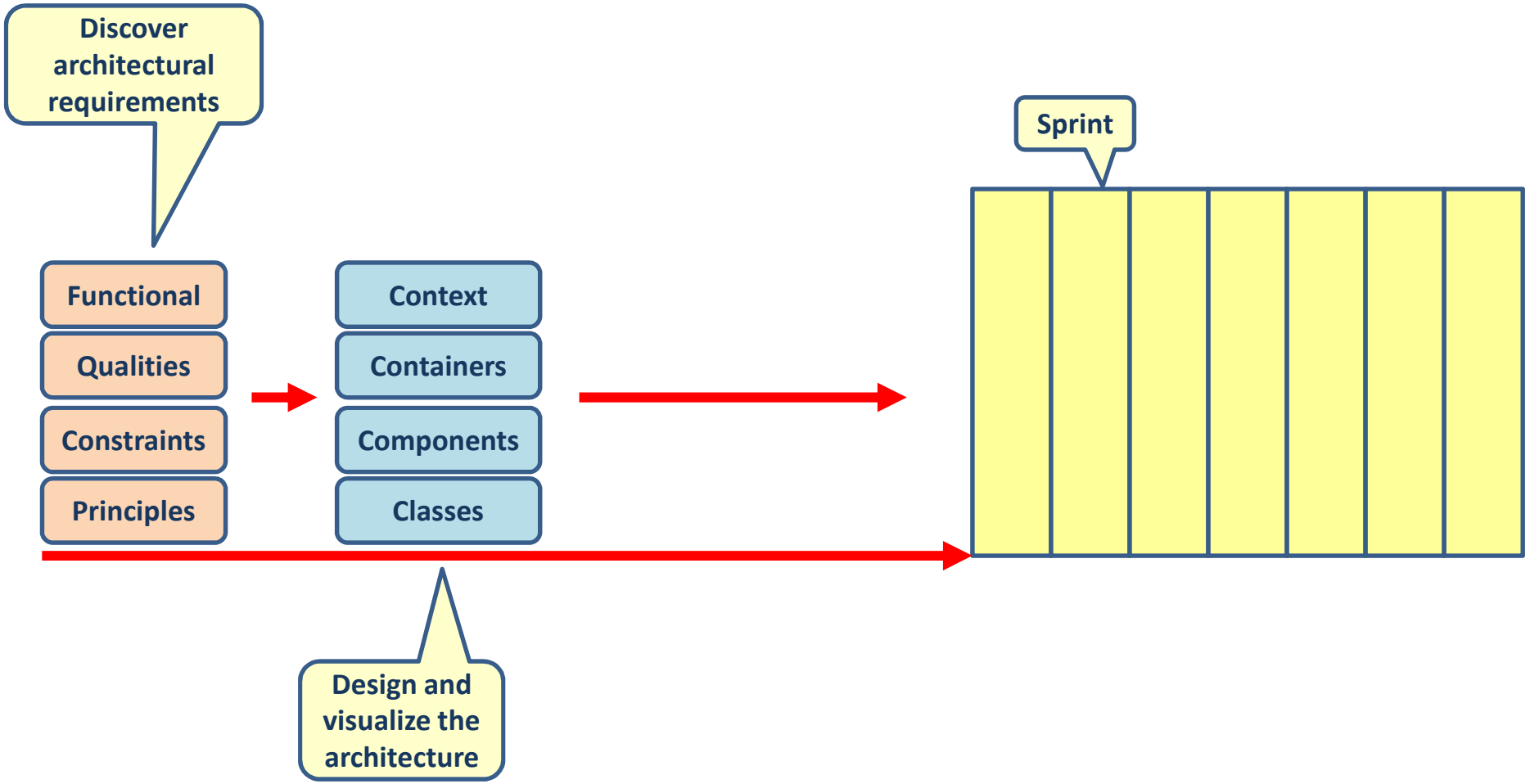
↓

Component diagram: Decompose each container further into a number of distinct components, services, subsystems, layers, workflows, etc.

↓

Detailed diagrams: Class diagram, sequence diagram, deployment diagram

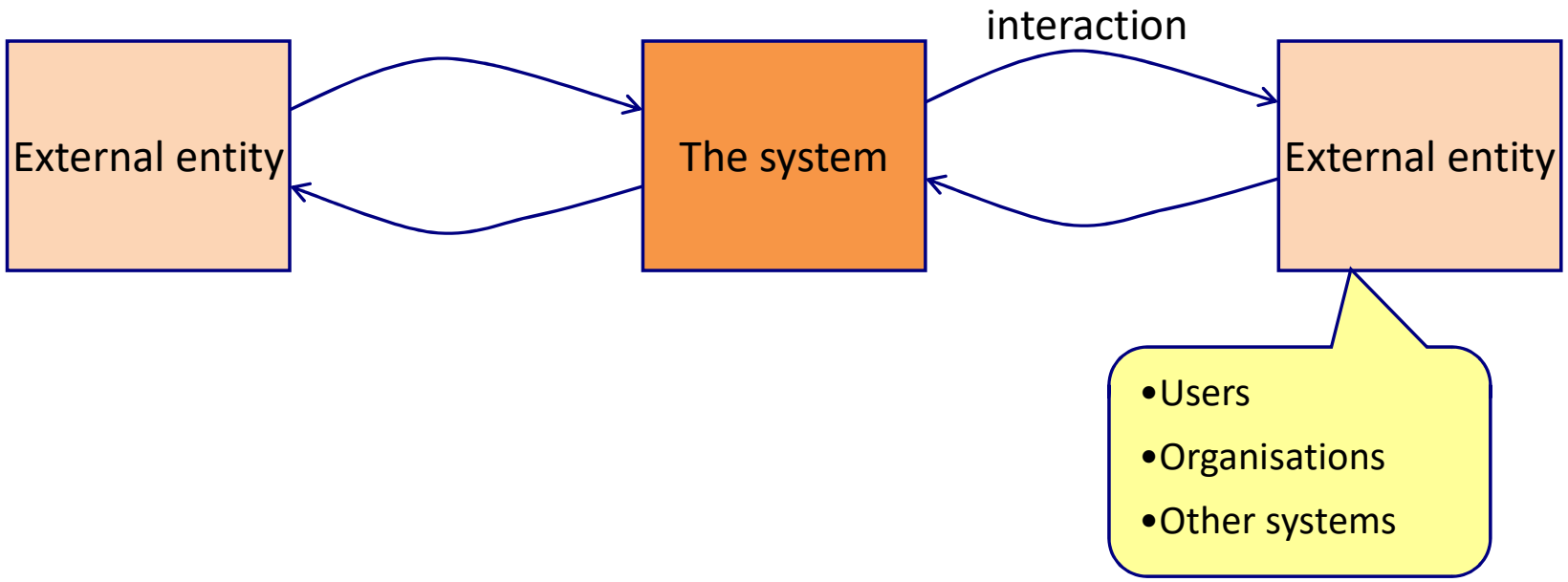
Agile architecture



Context diagram

- Shows the big picture
  - 10 km view

Context
Containers
Components
Classes



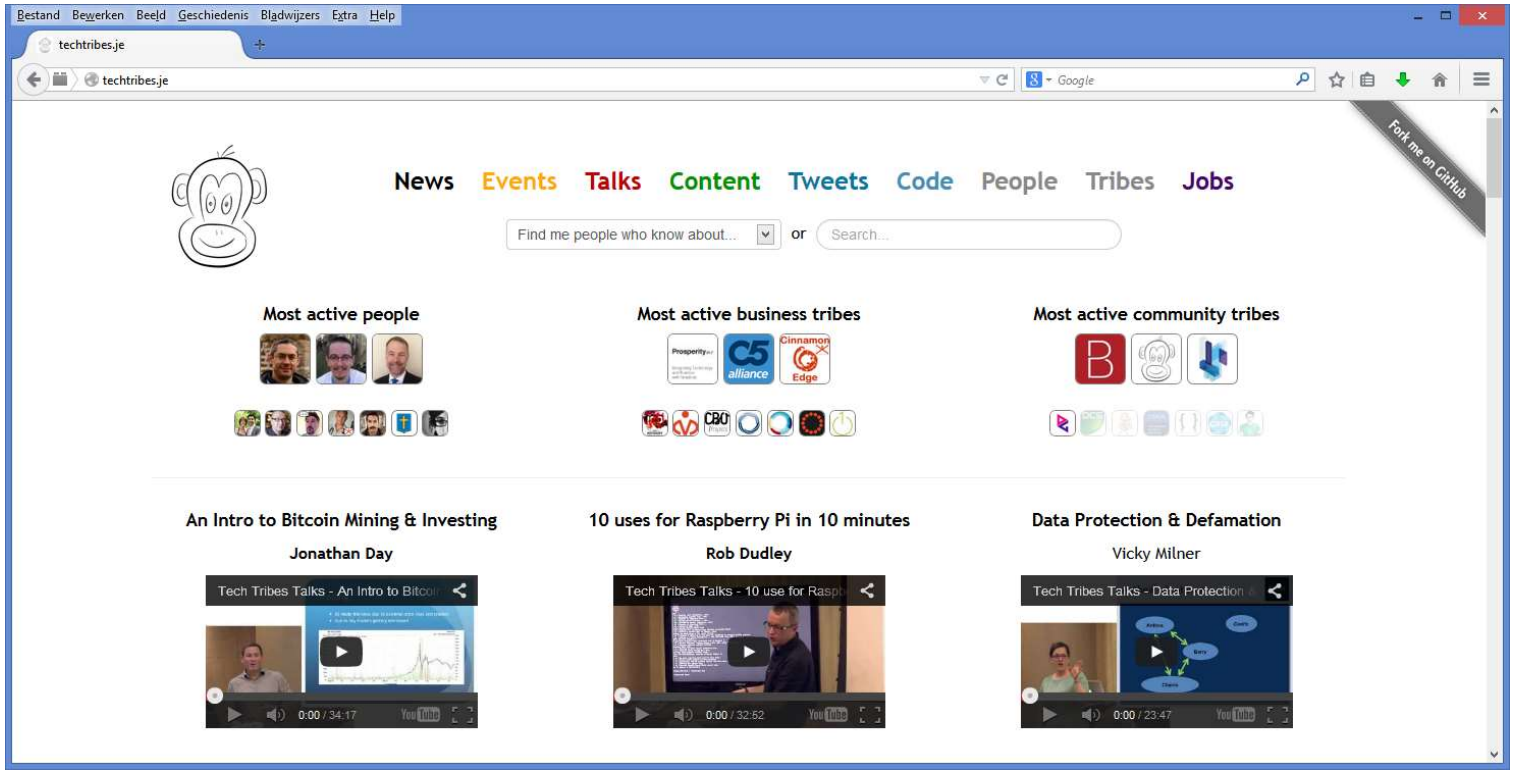
Context diagram

- Why?
  - It makes the context explicit so that there are no assumptions.
  - It shows what is being added (from a high-level) to an existing IT environment.
  - It's a high-level diagram that technical and non-technical people can use as a starting point for discussions.
  - It provides a starting point for identifying who you potentially need to go and talk to as far as understanding inter-system interfaces is concerned.

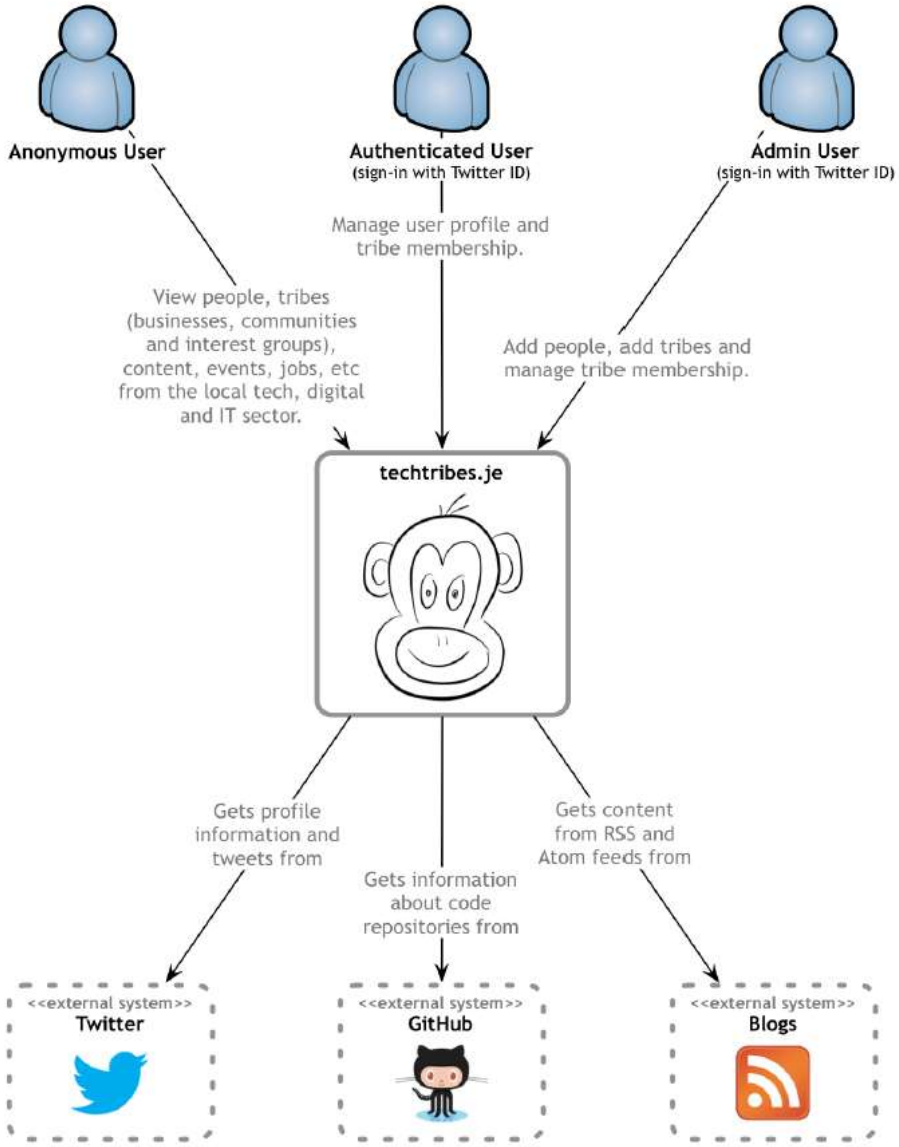
Should only take a couple of minutes to draw, so there really is no excuse not to do it

Example: Techtribes.je

Website that provides a way to find people, tribes (businesses, communities, interest groups, etc) and content related to the tech, IT and digital sector in Jersey and Guernsey. At the most basic level, it's a content aggregator for local tweets, news, blog posts, events, talks, jobs and more.

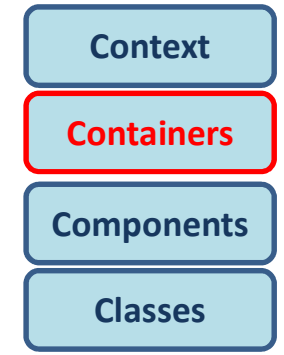


Techtribes.je context diagram

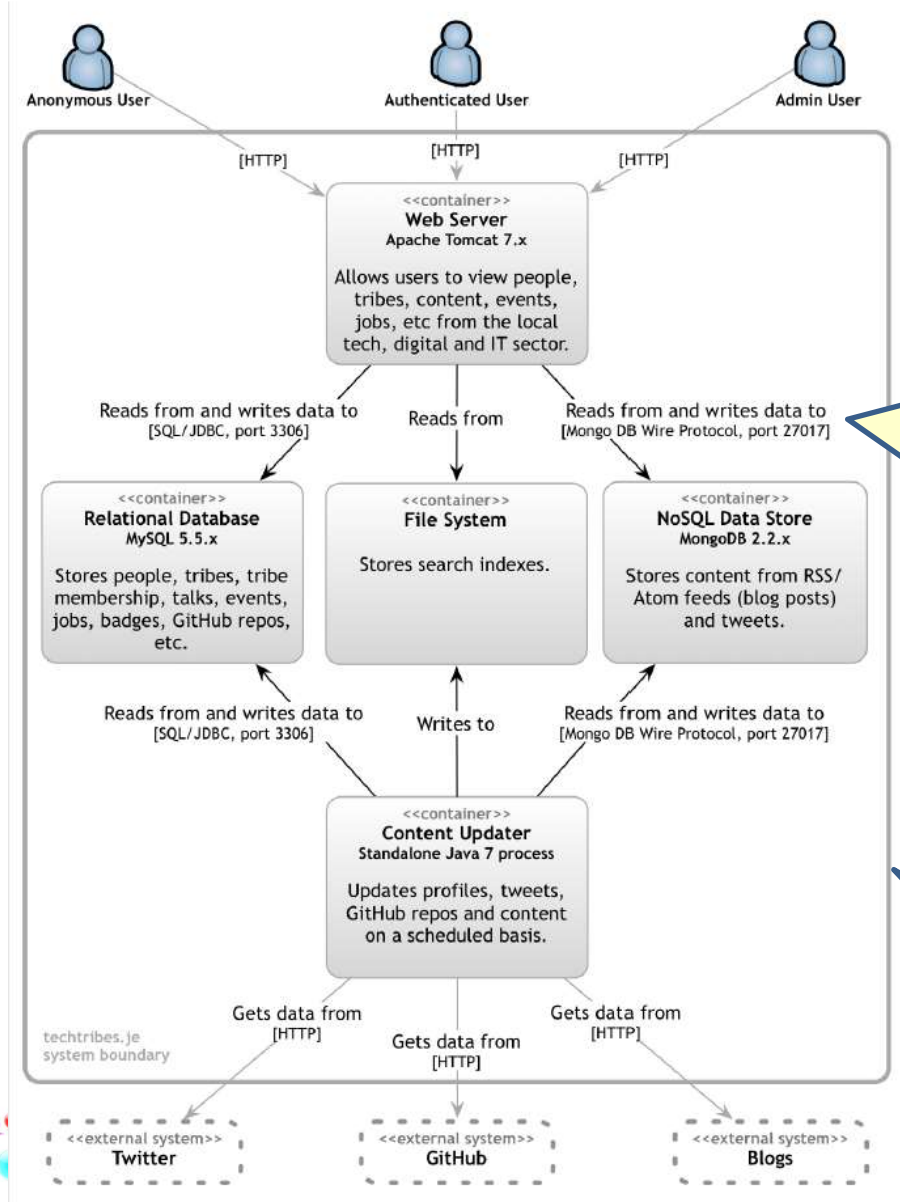


Container diagram

- Shows the high-level shape of the software architecture and how responsibilities are distributed across it. It also shows the major technology choices and how the containers communicate with one another.
- Containers are the *logical* executables or processes that make up your software system
  - Web servers
  - Application servers
  - ESBs
  - Databases
  - Other storage systems
  - File systems
  - Windows services
  - Standalone/console applications
  - Web browsers
- For each container specify:
  - **Name**: “Web server”, “Database”, …)
  - **Technology**: (e.g. Apache Tomcat 7, Oracle 11g, …)
  - **Responsibilities**: A very high-level statement or list of the container’s responsibilities.
- Everything on a container diagram should potentially deployable separately



Techtribes.je container diagram



Describe the interactions :

- **Purpose ("reads/writes data from", "sends reports to").**
- **Communication method (Web Services, REST, Java RMI, JMS).**
- **Communication style (synchronous, asynchronous, batched, two-phase commit)**
- **Protocols and port numbers (HTTP, HTTPS, SOAP/HTTP, SMTP, FTP).**

Show the boundary of what you are building

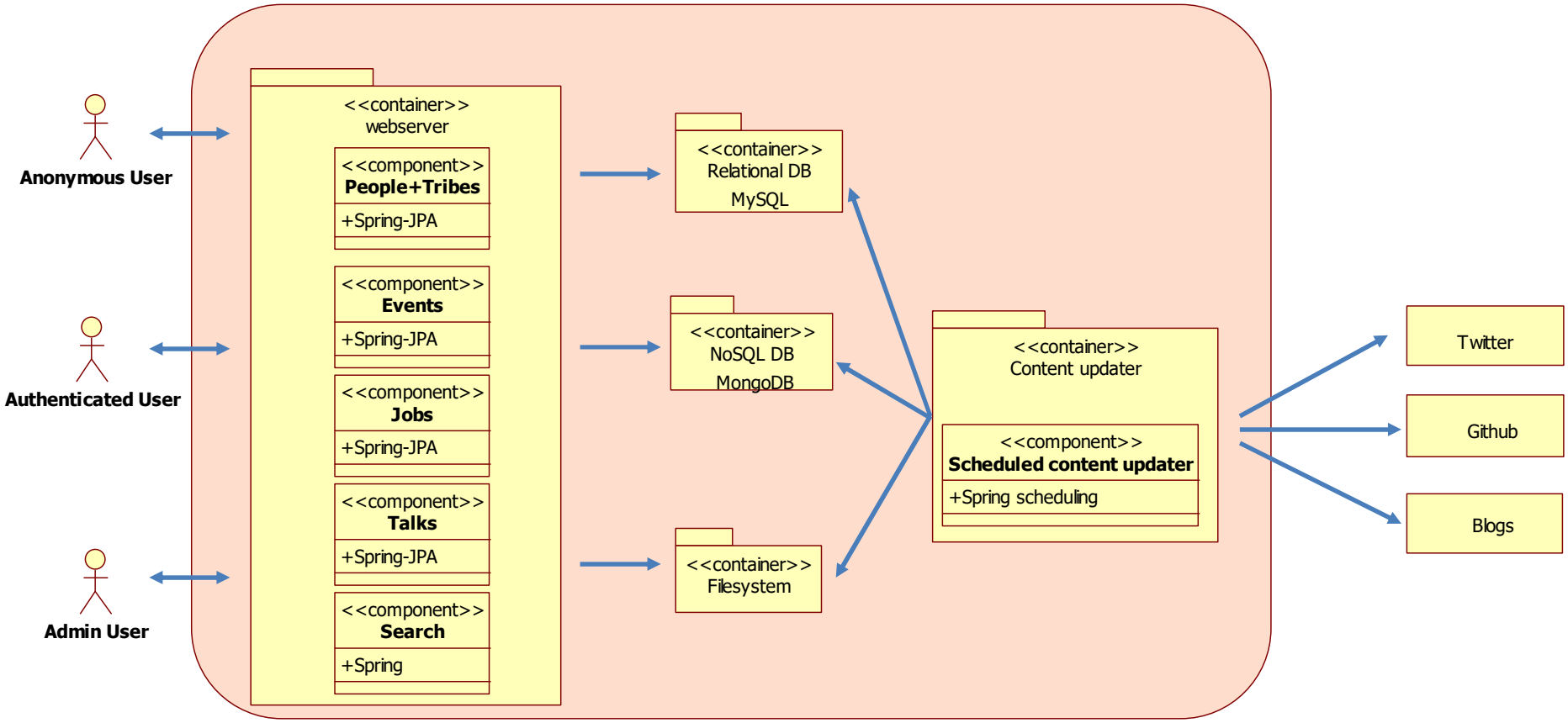
Container diagram

- Why?
  - It makes the high-level technology choices explicit.
  - It shows where there are relationships between containers and how they communicate.
  - It provides a framework in which to place components (i.e. so that all components have a home).
  - It provides the often missing link between a very high-level context diagram and (what is usually) a very cluttered component diagram showing all of the logical components that make up the entire software system.

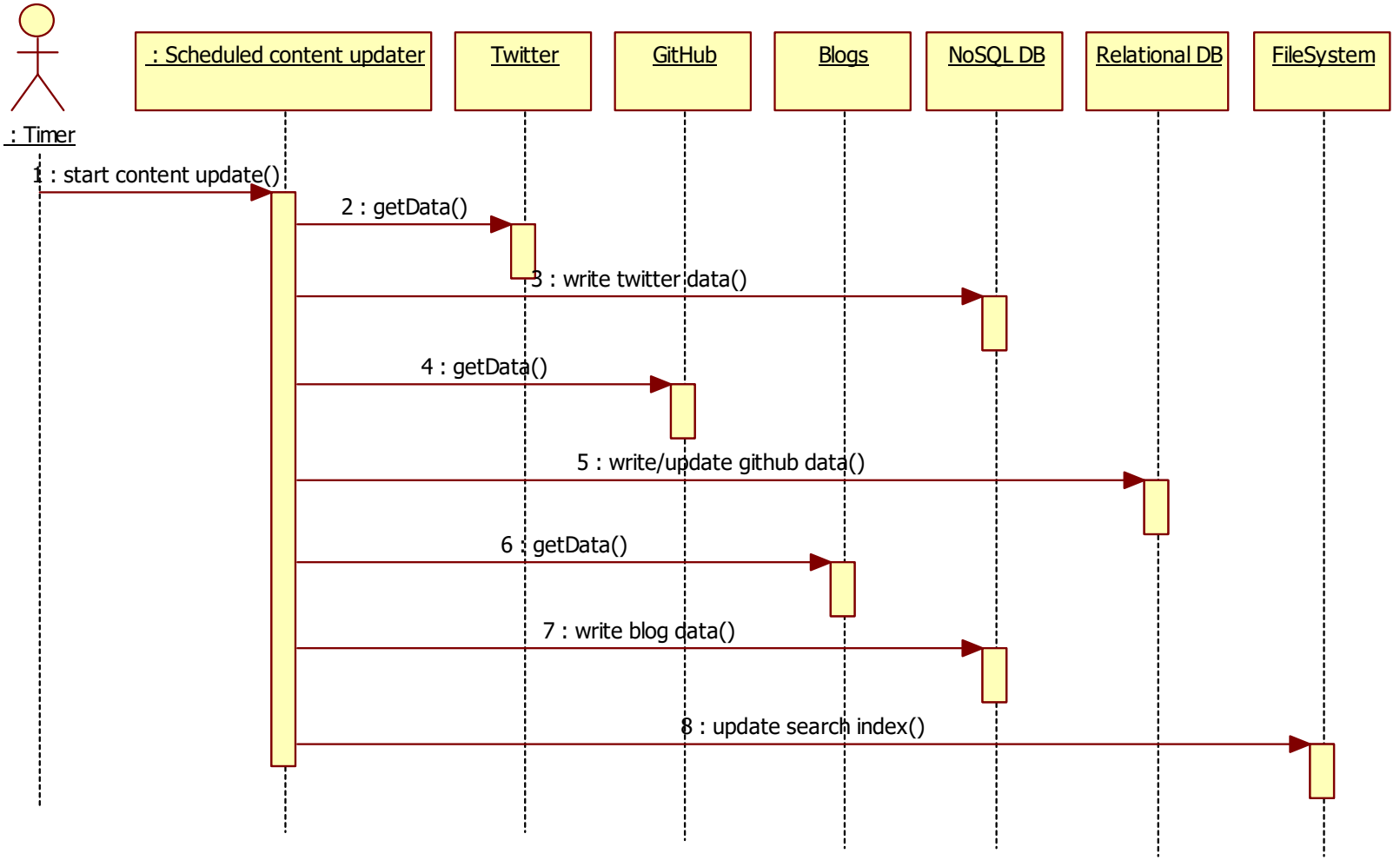
Component diagram

- Decompose each container further into a number of distinct components, services, subsystems, layers, workflows, etc.
- For each of the components drawn on the diagram, you could specify:
  - **Name**: The name of the component (“Risk calculator”, “Audit component”, etc).
  - **Technology**: The technology choice for the component (Plain Old Java Object, Enterprise JavaBean, etc).
  - **Responsibilities**: A very high-level statement of the component’s responsibilities (either important operation names or a brief sentence describing the responsibilities).

Component diagram



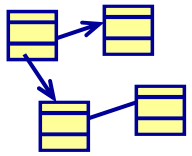
Behavior of the components



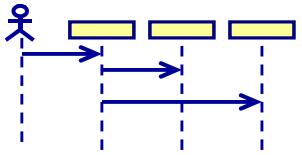
Component diagram

- Why?
  - It shows the high-level decomposition of your software system into components with distinct responsibilities.
  - It shows where there are relationships and dependencies between components.
  - It provides a framework for high-level software development estimates and how the delivery can be broken down.

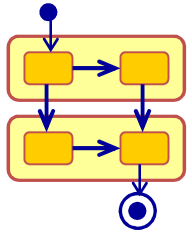
Detailed diagrams



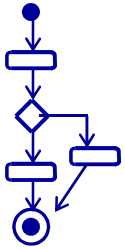
class diagram



sequence diagram



state machine diagram



activity diagram

Context

Containers

Components

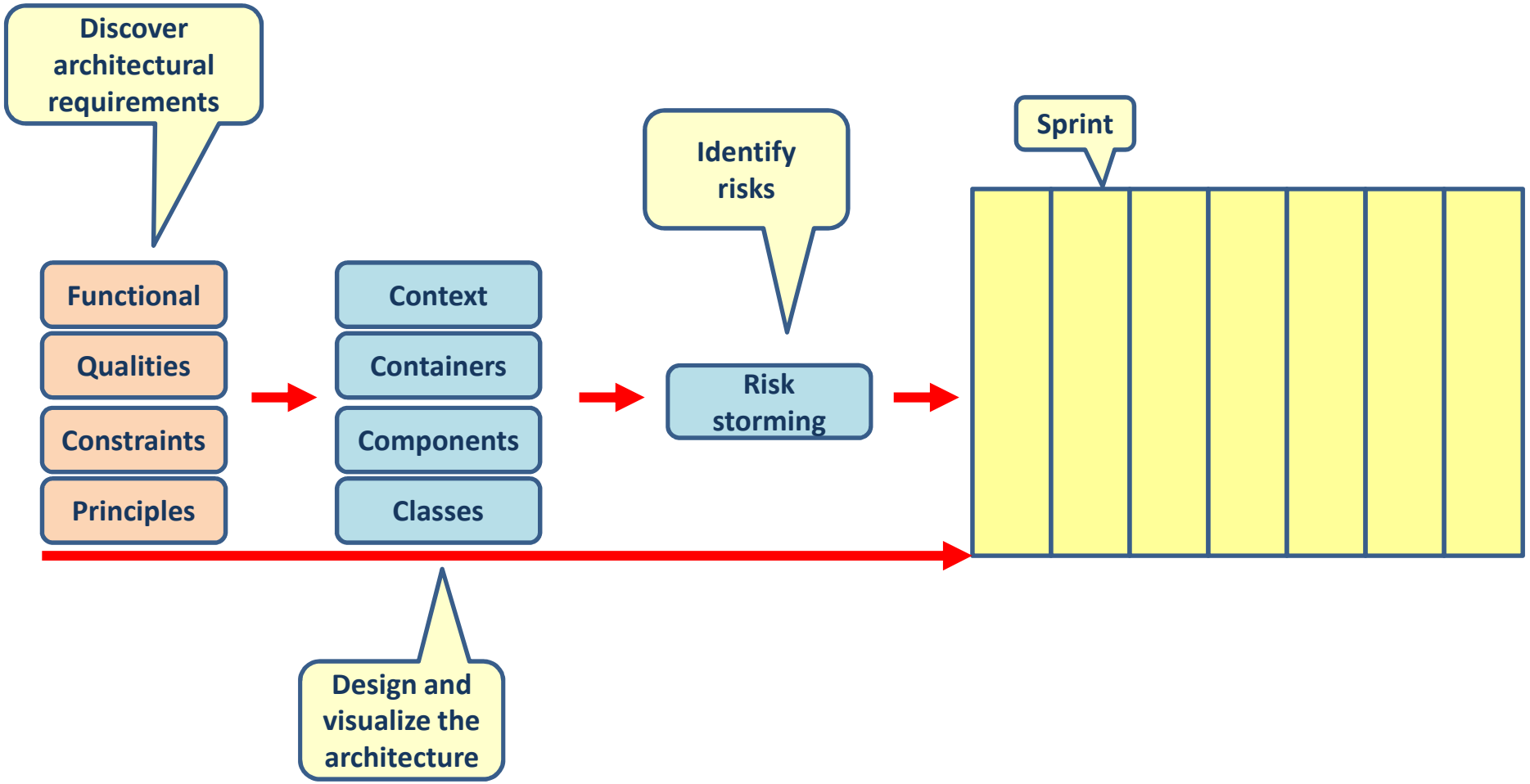
Classes

Agile diagrams

- Know the purpose of the diagram before you start
  - Only draw diagrams that you really need
- The diagram should fit on one page
  - And should still be readable
- Do not clutter the diagram with too much information
  - Remember what the purpose is
- What is allowed in the diagram?
  - Whatever fits the purpose
- You don’t need an UML tool.

IDENTIFY RISK

Identify risk



Quantify risk

- **Probability**
  - How likely is it that the risk will happen?
    - Do you think that the chance is remote or would you be willing to bet cash on it?
- **Impact**:
  - What is the negative impact if the risk does occur?
    - Is there general discomfort for the team or is it back to the drawing board?
    - Will it cause your software project to fail?

Probability/impact matrix

Risk storming

- Quick and fun technique that provides a collaborative way to identify and visualize risks.

Risk storming

Step 1. Draw the architecture diagrams

- on whiteboards or large sheets of flip chart

Step 2. Identify the risks individually

- The whole team stand in front of the architecture diagrams and *individually* write down the risks that they can identify, one per sticky note.
- Quantify the risk
- Use different colors of sticky note to represent the different risk priorities.

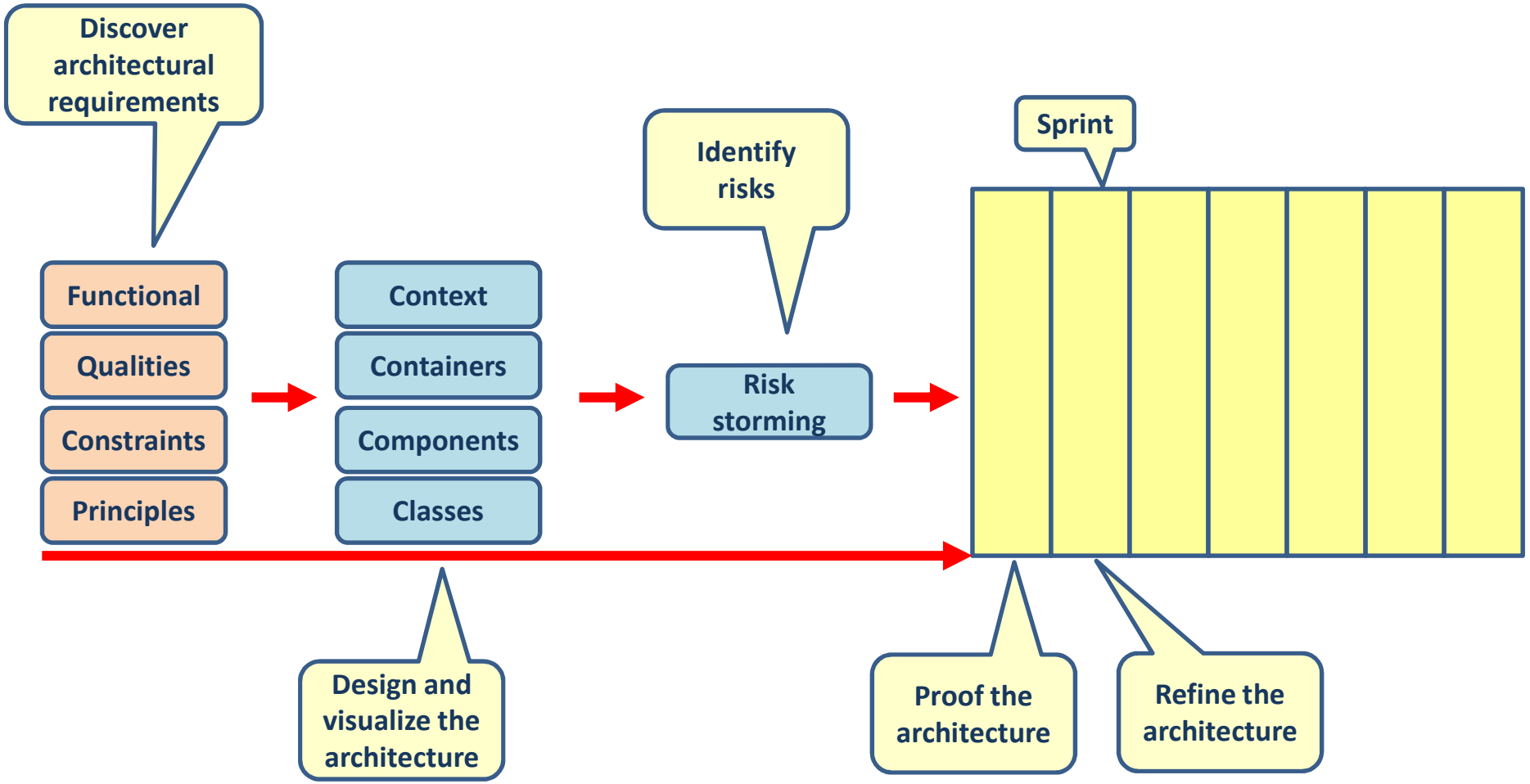
Risk storming

Step 3. Converge the risks on the diagrams

- Everybody places their sticky notes onto the architecture diagrams, sticking them in close proximity to the area where the risk has been identified.

- **Step 4. Prioritize the risks**
  - Take each sticky note (or cluster of sticky notes) and agree on how you will collectively quantify the risk that has been identified.

Just enough architecture



Main point

- Software architecture is best communicated in visual diagrams starting with the abstract context diagram all the way down to the detailed class diagrams
- The orderly sequential unfoldment of creation is a reflection of pure consciousness. The Vedas are expressions of the structure of this pure consciousness

Connecting the parts of knowledge with the wholeness of knowledge

1. Software architecture defines all important aspects of a system.
2. The architecture decisions are based on the functional requirements, the qualities, the business constraints and the architecture principles

---

3. **Transcendental consciousness** is the natural experience pure consciousness, the home of all the laws of nature.
4. **Wholeness moving within itself:** In unity consciousness, one appreciates and enjoys the underlying blissful nature of life even in all the abstract expressions of pure consciousness.

SUMMARY

Agile architecture

Functional
- Use cases
- User stories
- Scenarios
- Processes
- Business rules
- Acceptance criteria
- …

Qualities
- Performance
- Scalability
- Reusability
- Security
- Availability
- Modifiability
- …

Constraints
- Budget
- Deadlines
- Enterprise arch.
- Experience
- …

Principles
- Keep it Simple
- Separation of concern
- Loose coupling
- Information hiding
- Modularity
- Flexibility
- …

Architectural requirements

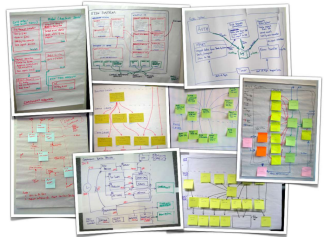
Context diagram: Shows the big picture.

Container diagram: Shows the high-level software architecture using containers and technology choices.

Component diagram: Decompose each container further into a number of distinct components.

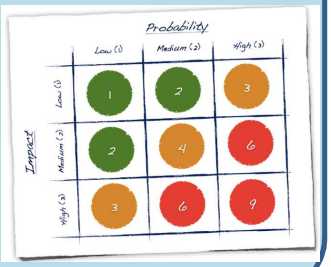
Detailed diagrams: Class diagram, sequence diagram, deployment diagram.

Design and visualize the architecture

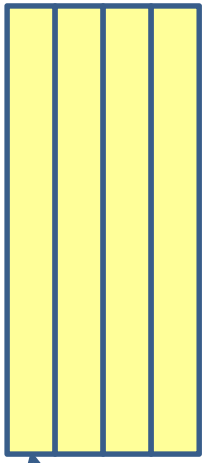


Identify risks

Risk storming



Refine the architecture in the Sprints



Proof the architecture.
Do the most risky things first.

Architecture (design) principles

- Keep it simple
- Keep it flexible
- Loose coupling
- Separation of concern
- Information hiding
- Principle of modularity
- High cohesion, low coupling
- Open-closed principle

Most important qualities

- Performance
- Scalability
- Availability
- Reliability
- Maintainability
- Security
- Interoperability
- Usability