# Agilität oder Architektur oder Beides?

Andrena ObjektForum 22. Juni. 2015

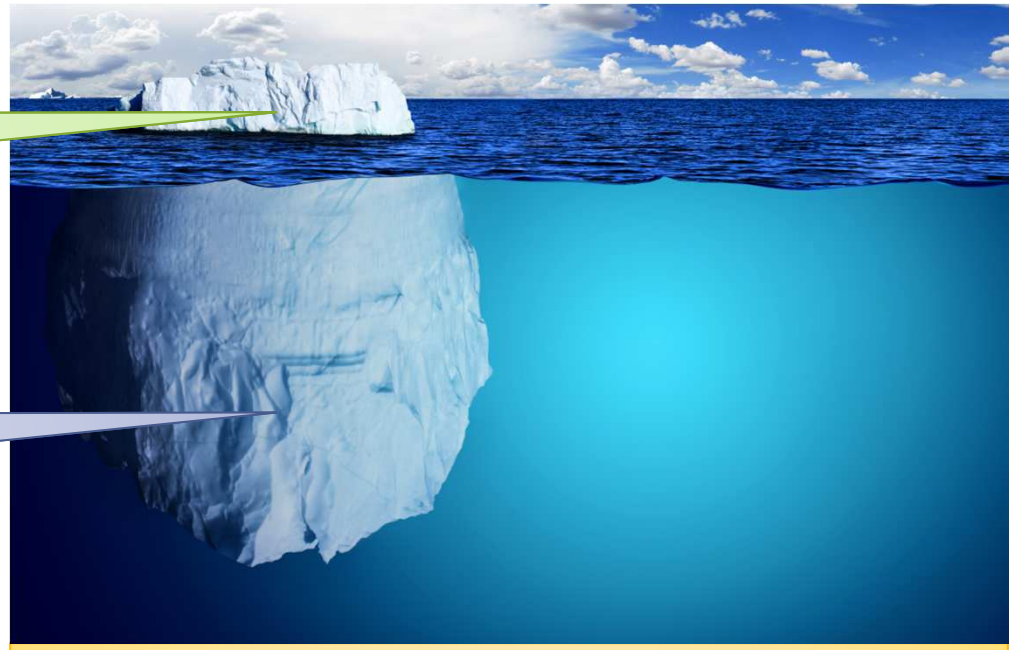
Prof. Dr. Michael Stal

Vortrag: Deutsch
Folien: Englisch

# Our Challenge

Ways to create Good Architecture

Ways to create Bad Architecture



*If you think good architecture is expensive,*
*try bad architecture*
Joseph Yoder, Brian Foote

# In a Perfect World

Knowledge of all Requirements a priori:

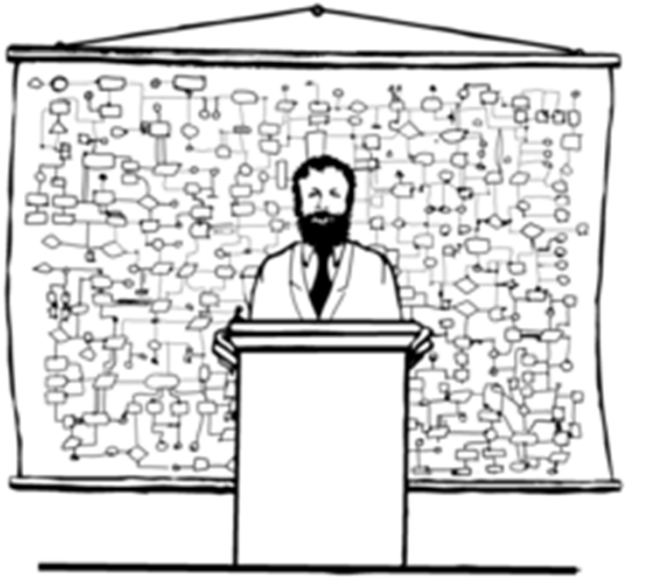
- Completeness
- Comprehensive and simple but not simplistic specification
- Error-free mapping to architecturally significant requirements


*"We have been having a hard time guessing the business requirements. I'm hoping our new analyst can help."*

# … and …

Context:

- No errors when mapping requirements to architecture
- No errors when mapping architecture to implementation
- No design erosion or architecture drift in further architecture



"Now that you have an overview of the system, we're ready for a little more detail"

# … and …

Architecting and Designing:

- Full knowledge about tools and technologies being used
- No errors in tools and technologies
- “Error-free” communication among stakeholders

"Nurse, get on the internet, go to SURGERY.COM, scroll down and click on the 'Are you totally lost?' icon."

# … and

no later  emergence of additional risks or (systemic) changes

… all of which requires

a pure Waterfall Model

In-depth and in-breadth knowledge of presence and past

=> get a time machine

Capability of forecasting the future

=> get precogs

… which, as we all know, is not possible – at least not today

Let's go Agile

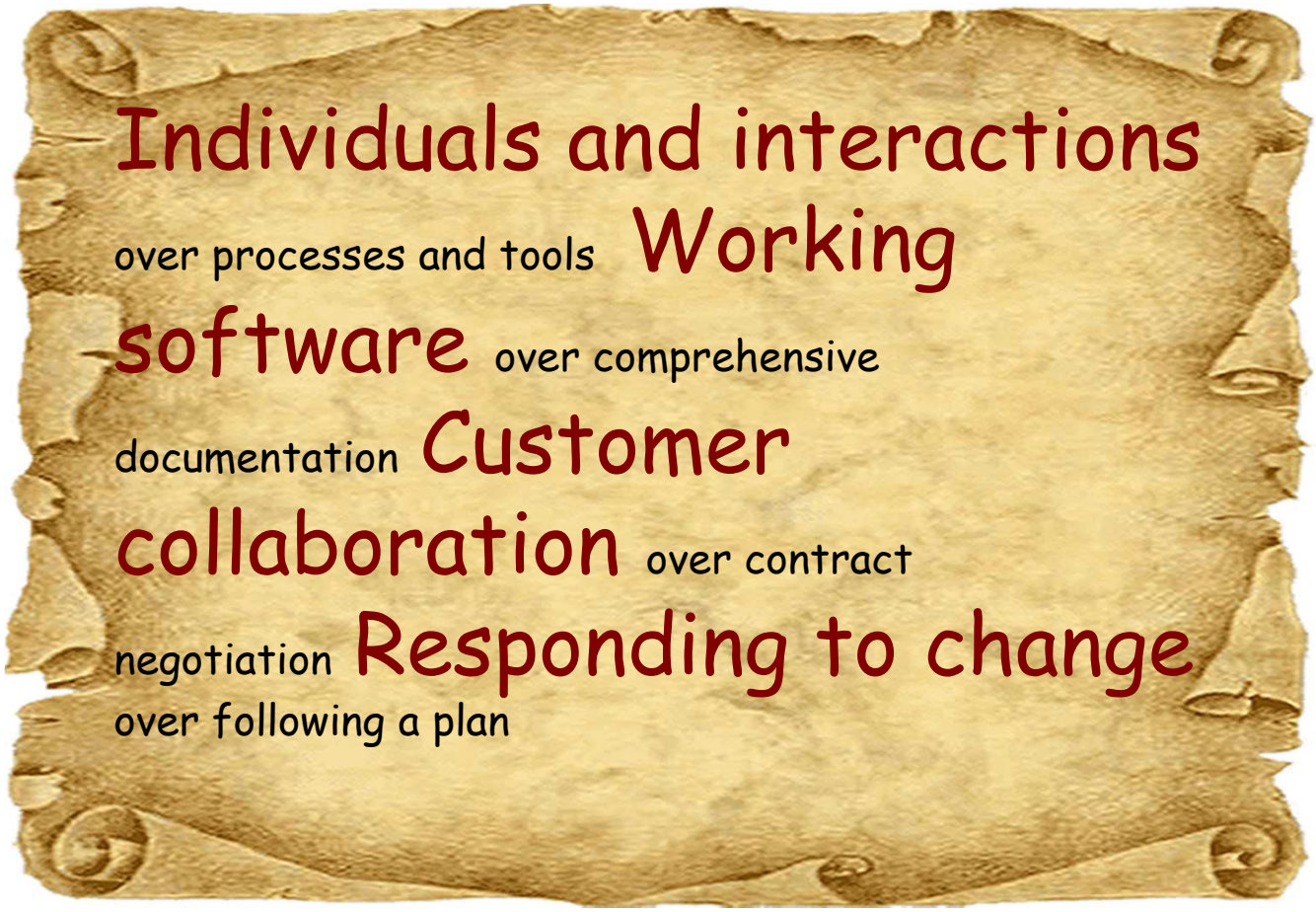
# Panta rhei

**There is nothing permanent except change**

**[Heraclitus, 535–475 BC]**

Agile Development introduces Evolutionary Design that embraces Change & Risks

# Agility – “Agile Manifesto”



# However, extreme approaches do not work

Purely Change-Driven Architecture Design

Fragile architecture!

BDUF Architecture (Analysis Paralysis)

Initially: Overengineered architecture!

Later: Sick Architecture due to Design Erosion =>Change resistant Design

# Naïve agile development process

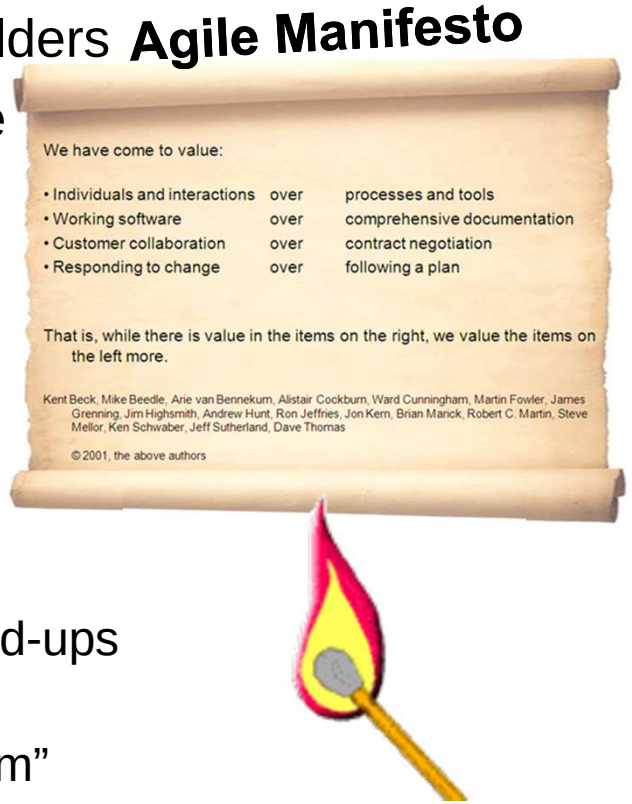
Less or no documentation – code is not documentation for non-technical stakeholders

Shared responsibility instead of expertise

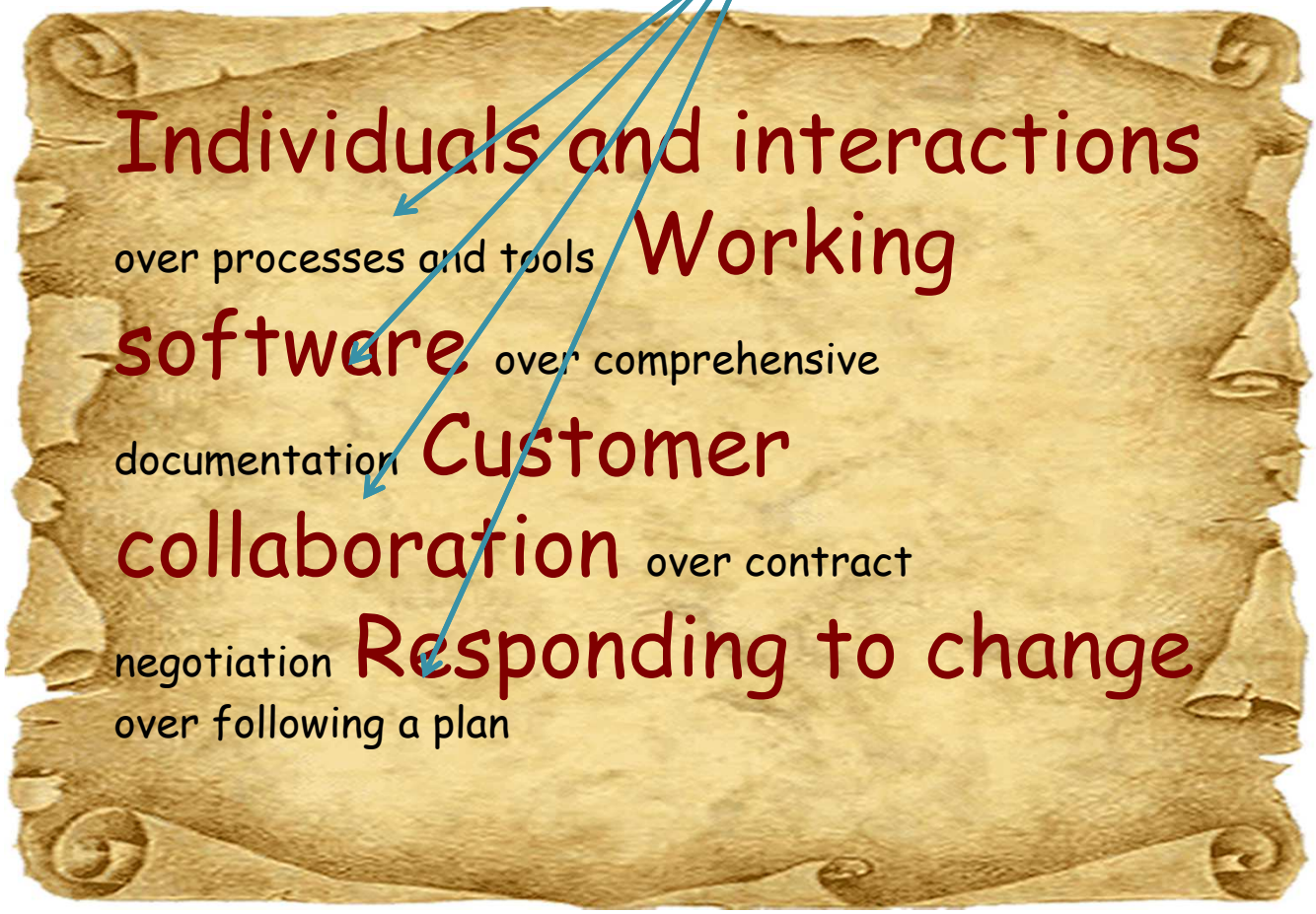
“Long” iterations (e.g. >= 6 weeks)

Additional Scrum traps:

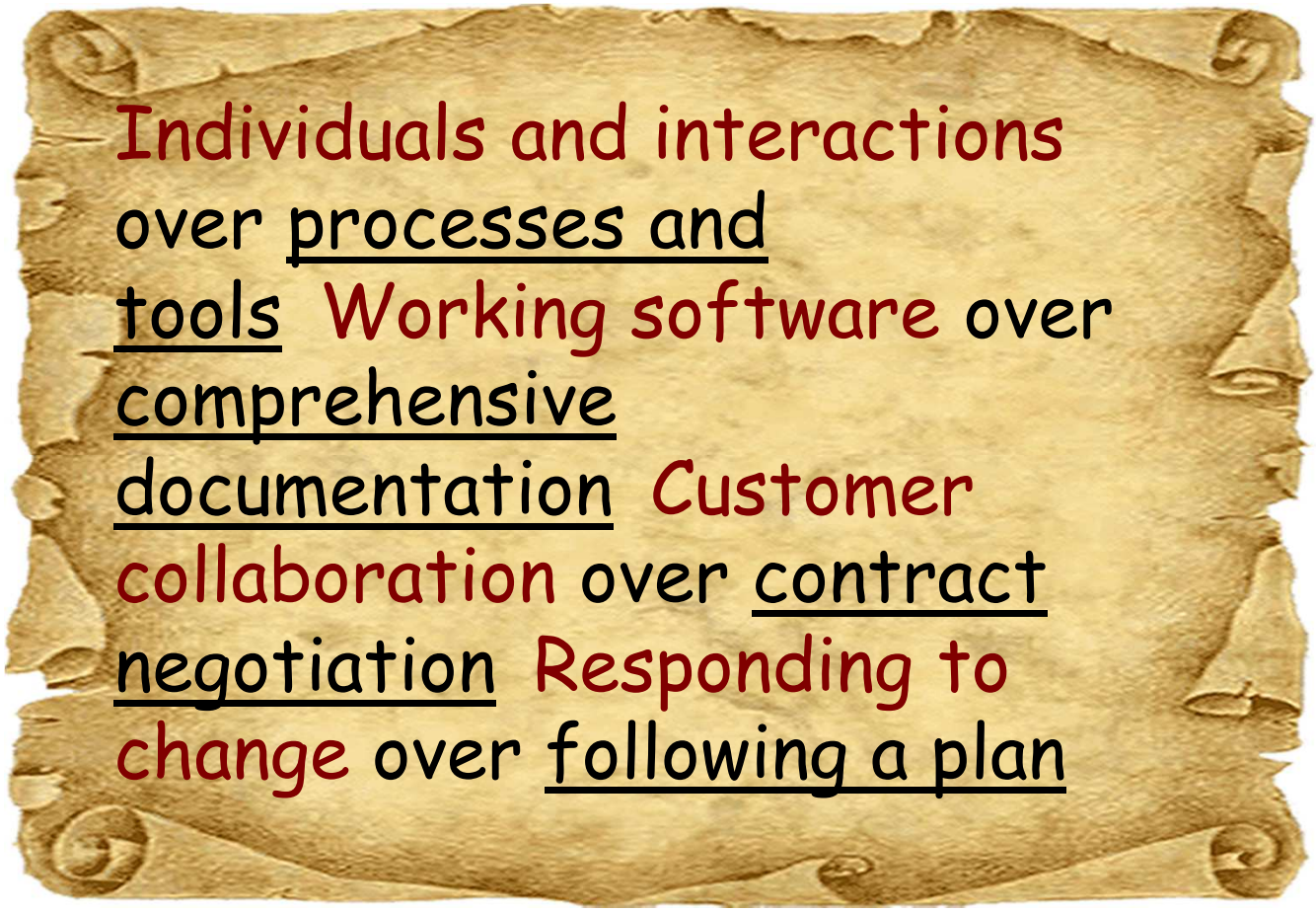
- Hybrid approach such as “V-Scrum”
- Unassigned Roles (e.g., Product Owner)
- Oversized Scrum Teams
- Misuse of Scrum Practices, e.g., daily Stand-ups organized like traditional Meetings
- Neglecting Scalability, e.g., “Scrum of Scrum”
- Lack of Tools for TDD, Continuity, e.g., Continuous {Build, Integration, Release, Delivery}



# Mind “das Kleingedruckte”



# How to read the Agile Manifesto



# Lean Agility - Pragmatic and Systematic Agile Process

The right sides of the agile manifesto are important

Agile Process must be well prepared like all other processes. Anarchy doesn't rule!

When combined with other methods, regular process assessment is a must

Daily Stand-up Meeting in Scrum

| | | |
|---|---|---|
| Competence Ramp-up | Continuous Process Improvement | Good Communication & Documentation |
| | Hire an Expert for Agile & Lean | Productive Tool Chain |
| ... | Agile Planning | Experts for Important Responsibilities |

# Scaling Up Agile Methods

Scalability in Scrum

- Hierarchy of Scrum teams (Scrum of Scrum)
- Specialized teams (such as architects, lead architect, middleware team, UI Team, Subsystem team)
- Availability of all Roles!
- Responsibility of product owners for multiple teams
- Synchronization with System Engineering, System Integration

# Organization as a Challenge

Conway's Law addresses a core challenge of Organizations

- "Organization affects Communication" implies: at least Project Organization should be based on communication needs

Avoid Design by Committee, introduce a Lead Architect

Establish an Agile & Lean development process, but scale it up for large projects

Agile Architecting implies Piece-Meal Growth

You can't restrict Agility to Architecture & Design

# Lead Designer

Best designs had a Lead Designer (=> cathedrals, Unix)

- i.e., an experienced and competent Lead Architect with the last word in architecture decisions and the courage to decide
- who needs to be empowered to lead efficiently & effectively
- who leads an architects group with 3-6 fellow architects

Also applicable for other roles!

***Basílica i Temple Expiatori de la Sagrada Família***
designed by Antoni Gaudi - Source: wikipedia

# Responsibilities of Agile Architects in Architecture & Design

- Architecture Baseline
- Scopes & Boundaries
- Qualities
- Architecting Process

- Habitability & Usability
- Commonality & Variability
- Systematic Reuse
- Architecture Enforcement

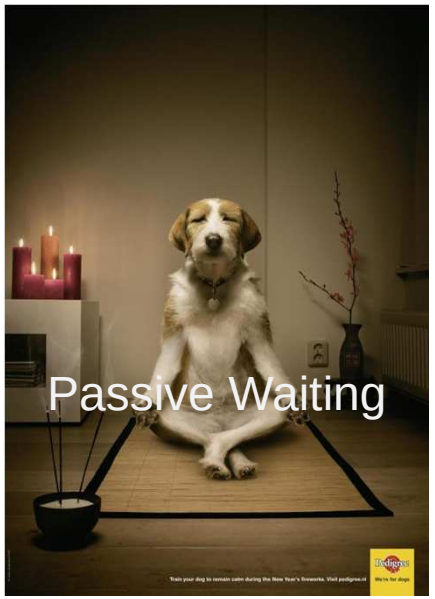
# Dealing with Uncertainty

Rule of thumb: to start architecting it's sufficient to know and understand a subset of requirements:

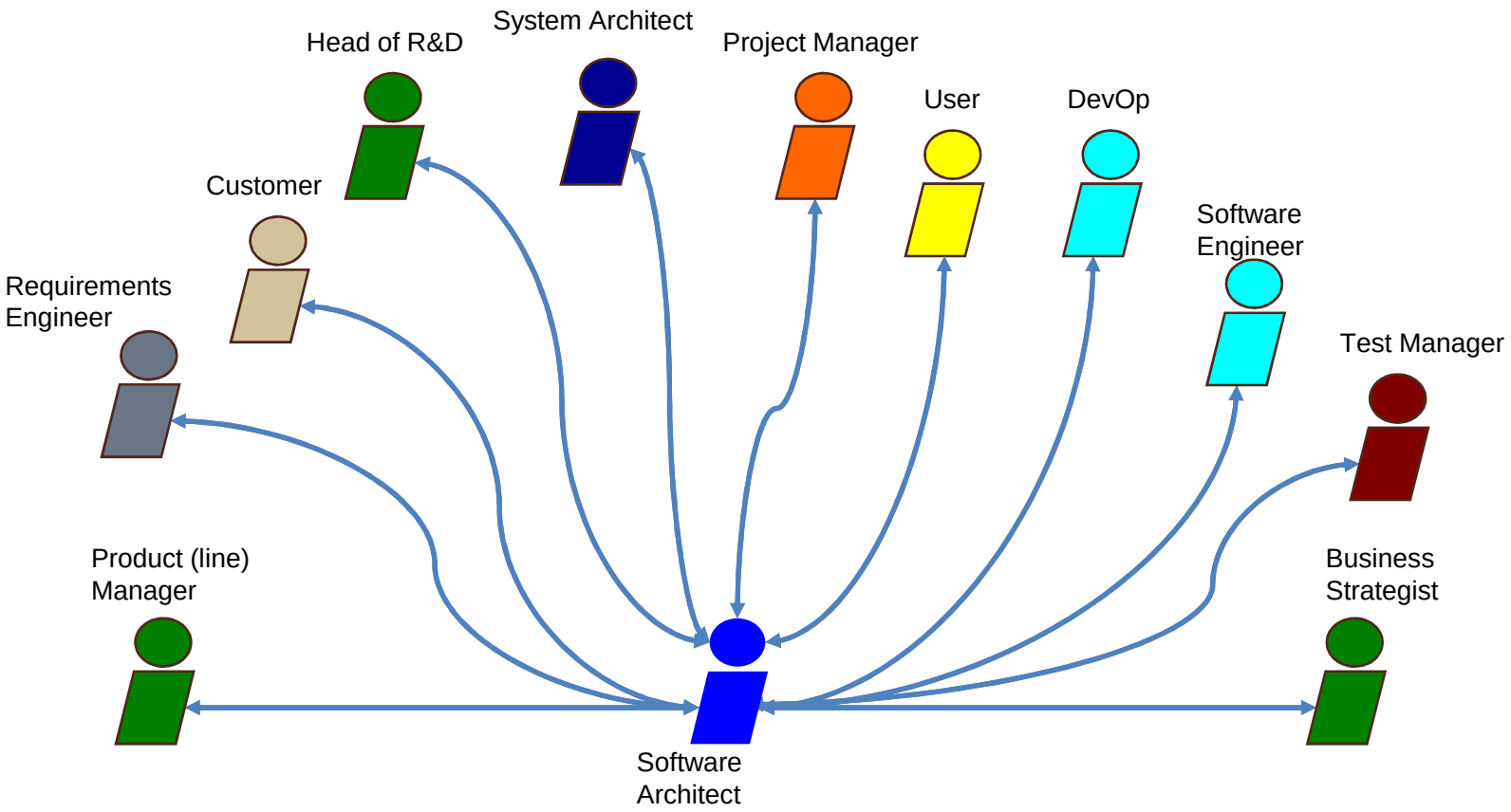
- most important use cases (happy day scenarios)
- most important quality attribute scenarios

If these are not available,

- communicate with responsible persons
- escalate to management
- start with feasibility prototypes or competence ramp-up



# Agile Development needs Agile Communication




Interaction Network of Software Architects

# Agile Communication - Meetings

Meetings interrupt work of project members

The more meetings the more interrupts

Thus, organize or join

- only the meetings that are necessary with
- only the participants required
- with a clearly defined and commonly accepted goal and
- with a fixed agenda (i.e., fixed before the meeting & slightly changed in the meeting) and a fixed time-box
- with meeting minutes


Solution Step: Locate source of overflow

# Agile Communication - eMail

Information should be distributed face to face or ear to ear

Only if that's not possible, use mail

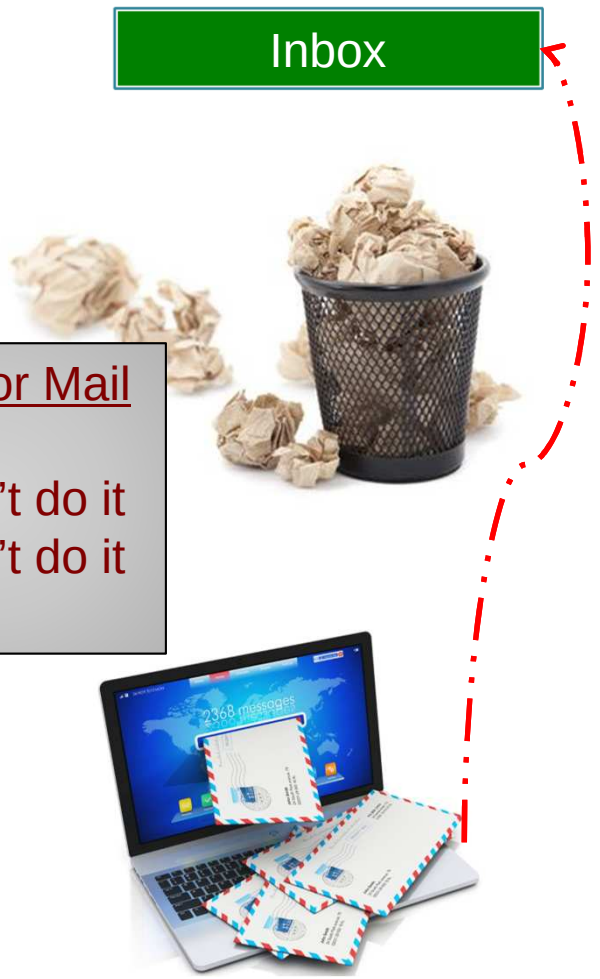
Useful for information that does not require many interactions

Large mails are terrifying

Trap: implicit assumptions or decision enforcement dependent c (lack of) mail response

No superfluous mail (small talk)

Mail only sent to recipients who need the information

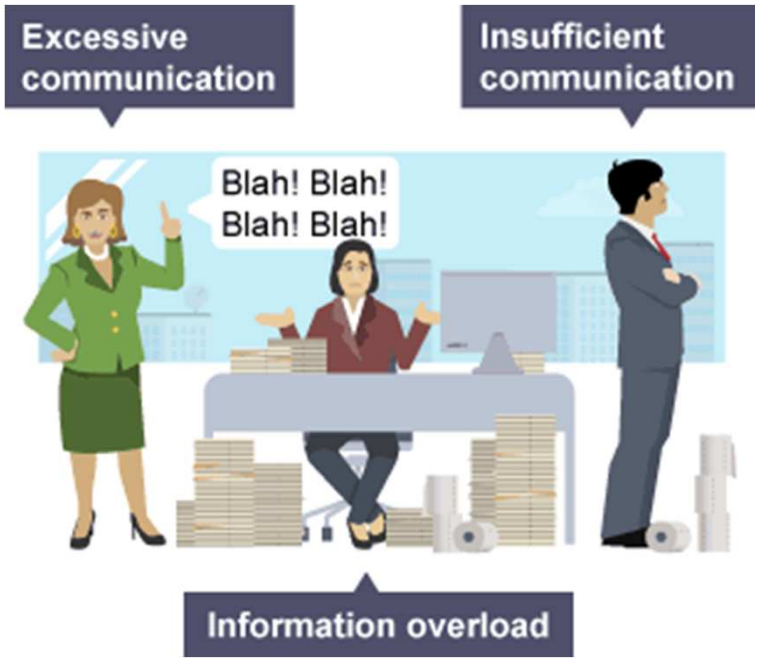


# Agile Communication – Peer-to-Peer

Communication efforts of architects in large projects might be close to 50% of work time

To increase your productivity

- shorten communication paths
- ensure communication is effective, efficient and sufficient
- minimize number of communication partners
- use always an appropriate communication infrastructure and communication method




Source: BBC

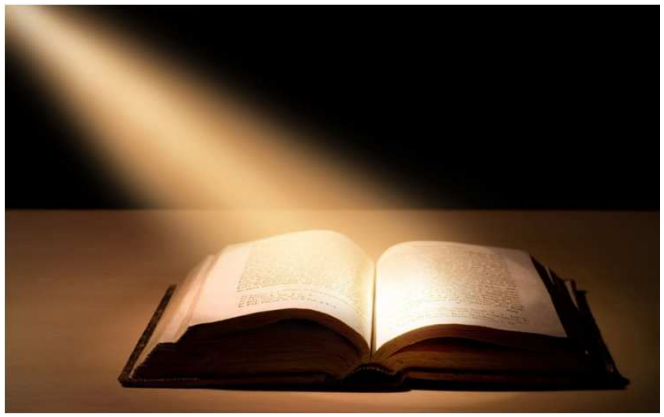
# Documents are for Communication, not for the Drawer

subject to versioning, change management, and reviews

contain the What, Why and How

with …

- one responsible main author
- clear and comprehensive vertical and horizontal structure
- focus on readability and understandability
- good habitability of information provided (KiSS)
- regular but not too frequent updates (e.g., after each increment)
- usage of domain language(s)
- balance between repetition and references



Good Content
– Enlightenment guaranteed

# For all Communication: Mind the Target Audience

Considering purpose & target audience

- Stakeholder-specific communication adapts content and communication style to target audience/culture
- Presenters need to understand mindset, expectations & culture of target audience
- Hints: Keep the communication time-boxed; close with positive aspects!
- To influence is fine, to manipulate isn't

Get feedback from colleagues in a dry run

Source: wikipedia

# Decision Tracking - Project Diary

Introduce a project diary* for everything important and not documented elsewhere

- Decisions and their rationale
- Discussions and their results
- Minutes

Helps remember decisions and discussions

With a diary you can

- prevent unproductive and repeating discussions
- understand the whole project much better
- track project history and decisions


* You may use a Wiki for this purpose

# Agile Requirements Engineering

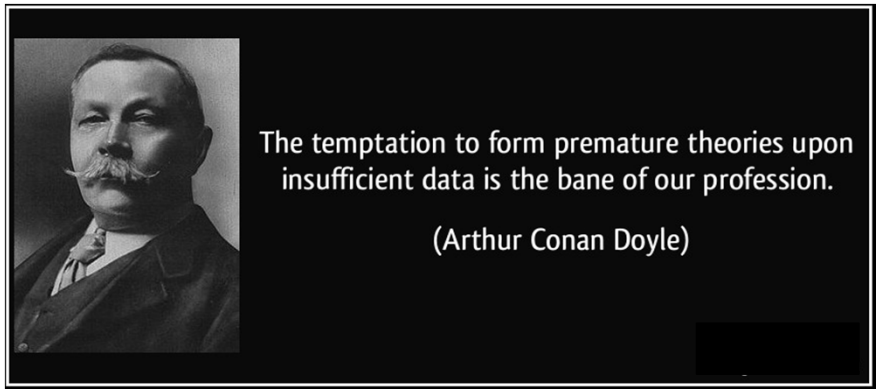
Cem Kaner: “A program which perfectly meets a **lousy specification** is a **lousy** program”

# Quality Check

- Check whether the specification is correct, complete, consistent, observable, verifiable, current, mandatory, unambiguous, feasible
- No implicit requirements!
- What the users & customers need rather than what they want!

# Prioritization

- Assign unique priorities to requirements
- Let priorities guide all architecture & design activities
- Address requirements incrementally and iteratively with high priorities first:
  - Conduct Strategic before tactical design
  - Operational before developmental qualities
- If management can't provide prioritization
  - propose one and ask for feedback!




# Twin Peaks

In high uncertainty* pure sequential processes do not work, because

- Requirements engineers do not know if their specification is feasible
- Architects do not know whether they can meet the specification

**Twin Peaks** approach:

- Requirements engineering and architecting happen in parallel and with tight interaction
- Feasibility prototypes to clarify and discuss open issues early

Twin Peaks helps mitigate risks and reduce waste

Change of Perspective is essential

* such as unknown domain, insufficient technology knowledge, unavailability of a reference architecture

# Risk Management – Findings Risks

Template for Describing Risks– created by Christine Hofmeister

**Description of risk**

e.g., dependence on persistence layer

**Influential factors that lead to this risk**

e.g., requirement to decouple business from persistence layer, not enough management skills in team

**Solution approach**

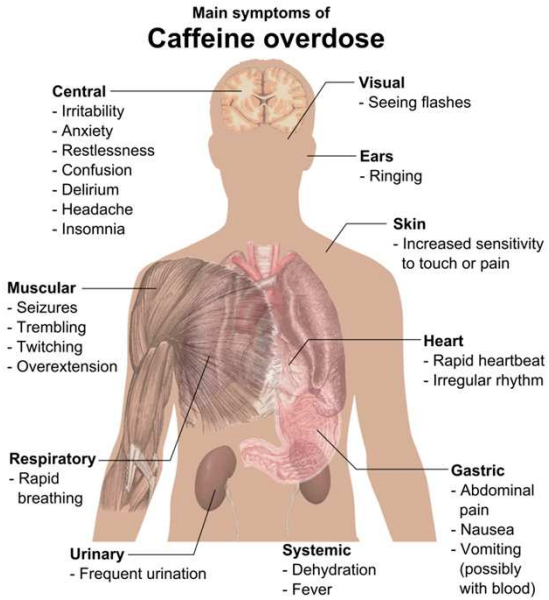
e.g., introduce data access layer

**Possible strategies**

e.g., give subproject to external company, use open source solution, use platform-specific solution

**Related topics and strategies**

e.g., decoupling business logic from other backend layers



# Risk Management: Risk-based Testing Strategy


Architects and Testers jointly define a Risk-based Testing Strategy

| Part | Risk | Area affected | Probability | Impact & Damage | Priority | Testable and how | Automated testing? | Time | Budget |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| Use case A | high | Packet tracking | 0,01% | Low impact 1000€ loss per minute | 2 of 10 | Yes White box testing | Yes | 10 Days | 10€ |
| | | | | | | | | | |
| Usability | | | | | | y Lab | | | |
| … | … | | | | | | | | |

Use a Risk-based Testing Strategy

# Design for Testability

Provide functionality to retrieve run-time information, e.g.,

test interfaces, test components, logs, traces, …

Important Aspects:

Observability: capability of identifying the cause of an effect

Controllability: capability of setting the system to a specific state

Mind the Heisenbug: test functionality may affect system properties and even cause bugs

=> identify degree of observability &

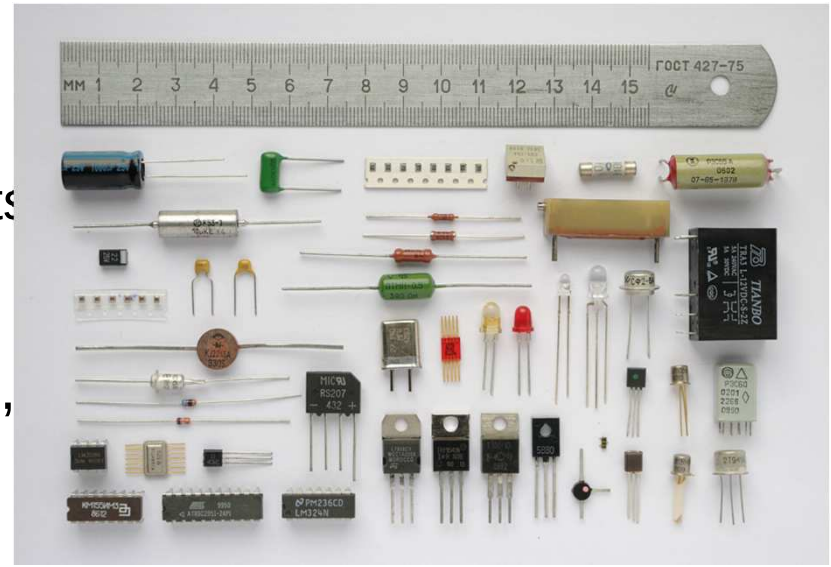

# Technology Check

- Technology Management is necessary to reduce uncertainty and risks
- Quantitative Assessment is necessary for ensuring Quality Attributes:
  - Feasibility Prototypes to verify technology decisions
  - Simulators to model environment
  - Benchmarks to trigger events and to check & measure system reaction

# Prerequisites for Systematic Agile Architecting

- Understanding of Problem Domain
- High Quality Requirement specification
  - at least 30% of all requirements
  - almost all high priority requirements
- Knowledge of Business Goals, Strategy, Constraints, Risks
- Risk-Based Testing Strategy
- Agile/Lean Process Model
- Technology & Tool Competence in Solution Domain
- Adequate Staffing

Assumption: all of this is in place



# Agile Architecture = Walking Skeleton Metaphor

Skeleton initially represents architecture baseline

and is extended slice by slice with functionalities or qualities:

User Scenario,

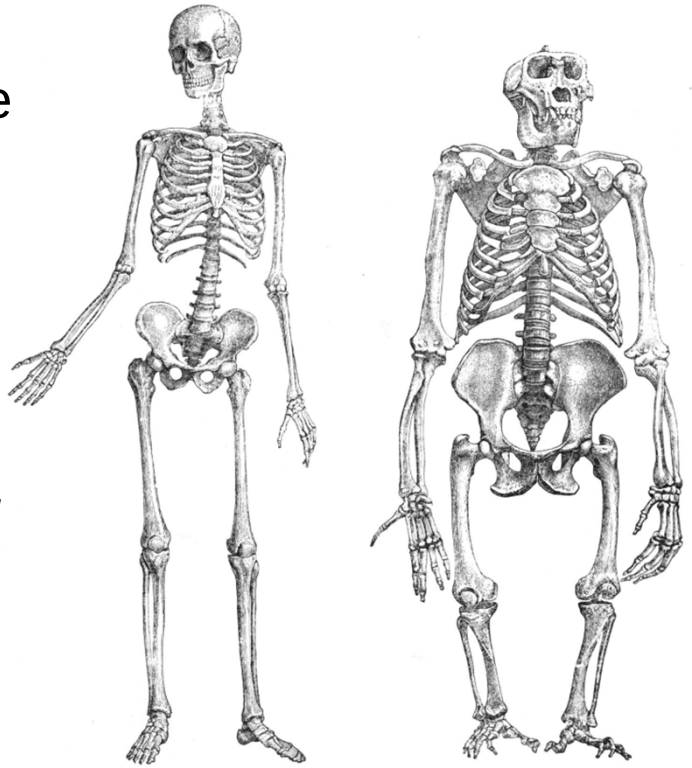
*adding all muscles, nerves, …, needed to move a finger*

Quality Attribute Scenario,

*(response time of) finger movement*

After integration of new slices Regression-Testing required

which is known as piecemeal growth



Left: Walking Skeleton for New System
Right: Walking Skeleton for Legacy Code

Source: wikipedia

# Architecture Design is Systemic Design

## Strategic Design

- Design of Entities with Global Impact
- limited to Higher Abstraction Layers

## Tactical Design

- Refinement of Strategic Design
- constrained to Entities with Local Impact

Architecting = Strategic Design + Selection of Global Infrastructures for Tactical Design

# Strategic and Tactical Requirements

Consequently, requirements can be separated in

- Functional Requirements: all requirements that define the functionality of a system (family).
- Operational Requirements
  - define Quality of Service (quality of the system operation)
  - Mostly cross-cutting
  - Examples: Performance, Security, Dependability, …
- Developmental Requirements
  - define Quality of Architecture (e.g., structural properties of the system architecture)
  - Mostly local
  - Examples: Modifiability, Testability, …

# Systematic Evolution of Architecture

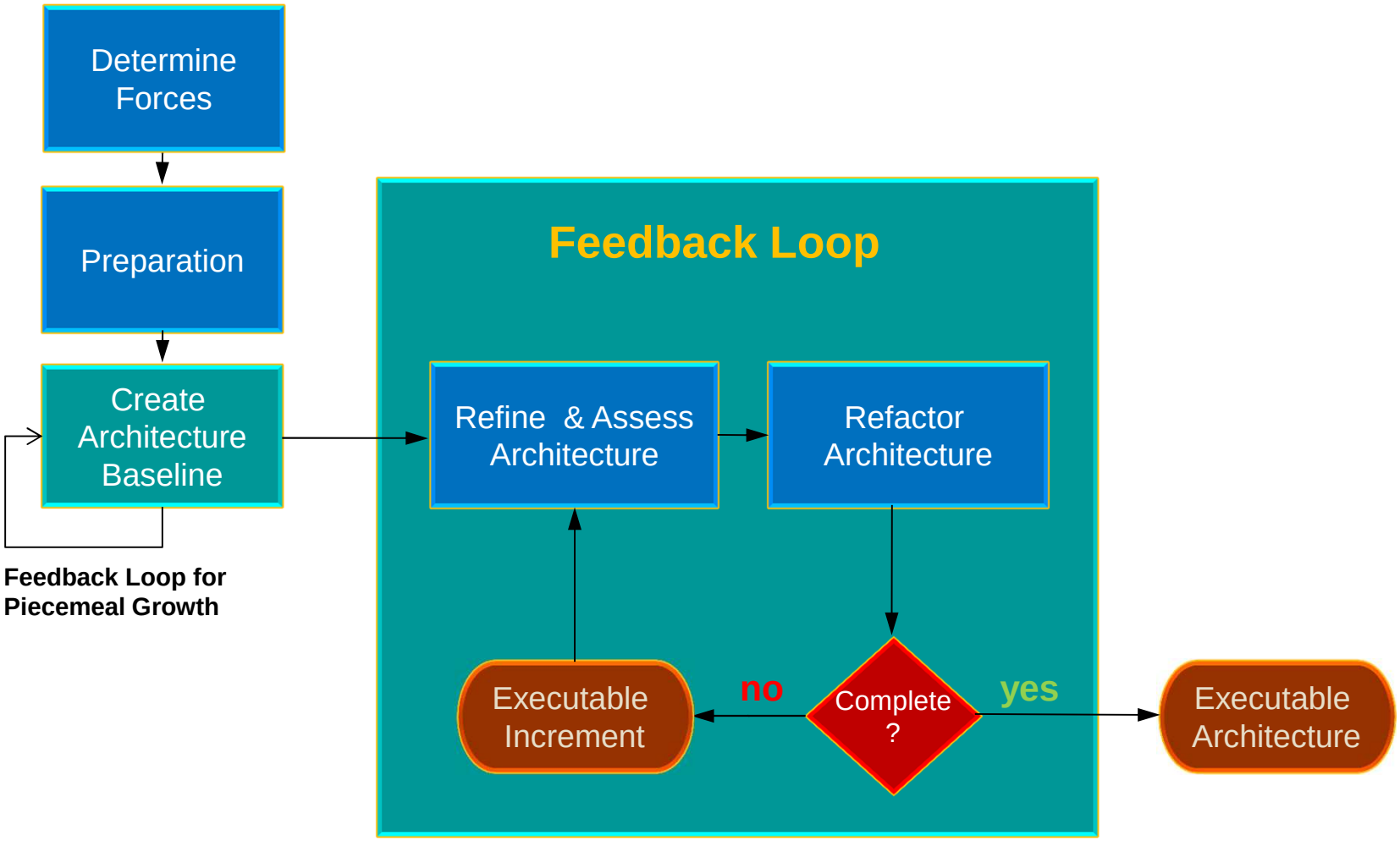
Steps:

1. Derive functional architecture from both functional qualities (functional scenarios use cases, user stories, …) and the problem domain (model).
2. Introduce the strategic qualities (scenarios) desired for the target system.
3. Ensure that the target system can be changed, maintained, modified, … according to the tactical qualities (scenarios)


Observations:

- Before dealing with strategic qualities such as performance, the functional scenarios are required to which the qualities are applied. Thus, step 2 must follow step 1.
- Tactical qualities such as extensibility can only be introduced when the architectural constituents are available that are subject to extensibility (i.e., either functionality or strategic qualities might vary). Thus, step 3 must follow step 2.
- It is necessary to continuously evolve the architecture, i.e. its layers. Thus, layers are neither carved in stone nor strict, but fluent.
- The earlier a requirement is addressed, the more impact it will have on a system. Thus, architecture design should start with high priority requirements and continue down to low priority requirements. **Keep track of tradeoff points and sensitivity points!**

# Agile Architecture Process from 10000 feet

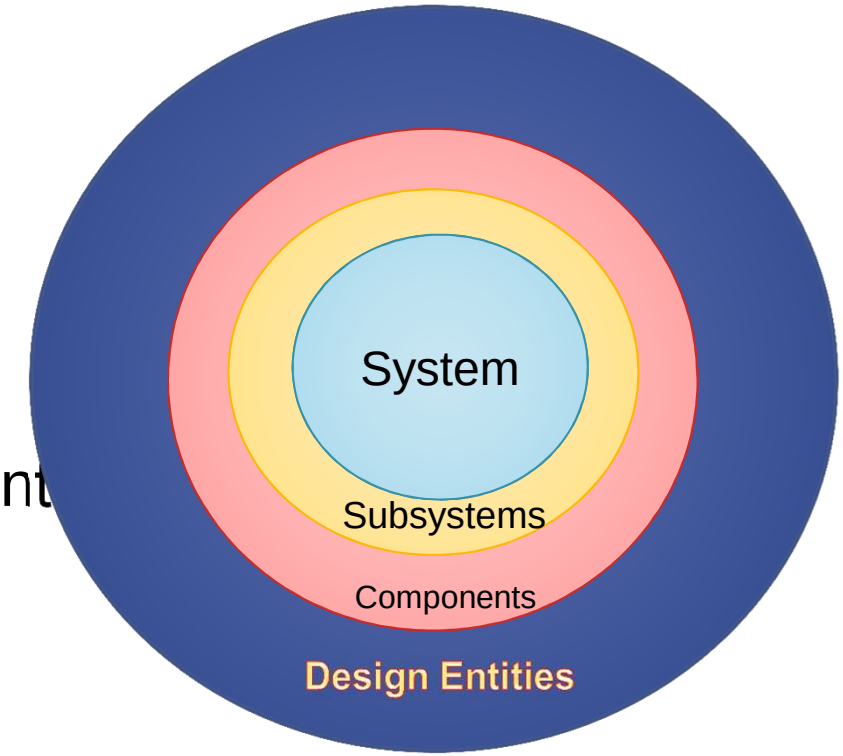


# Limiting Abstraction Depth

Rule of Thumb: use around 3 or 4 levels

- System
- Subsystems
- Components

Semantics of terms like Subsystem or Component need to be defined by development organization!



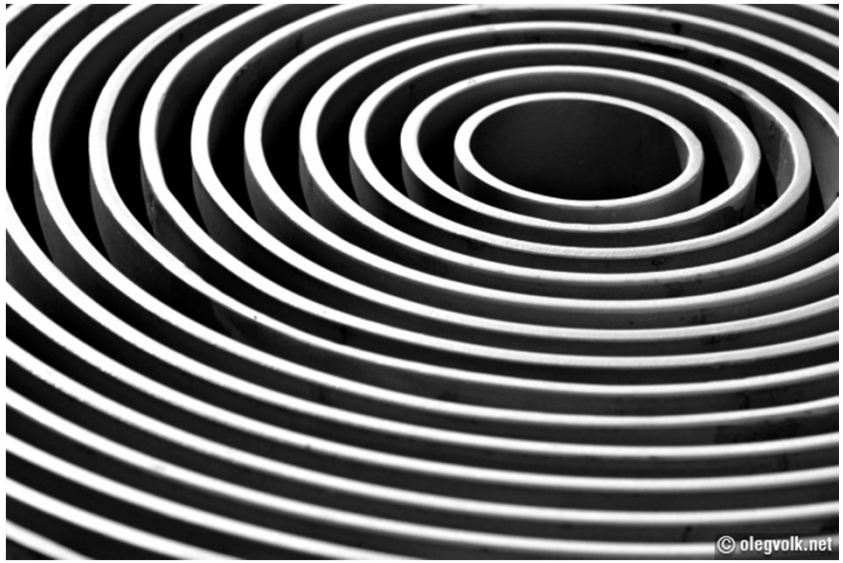
# Reviewing the Architecture regularly

Architecting and Design should apply “Piece-Meal” Growth

Assessments/Reviews are substantial ingredients of Architecting

Feedback Loop: Look back, identify problems, address them early, and then continue

All design work must be solely driven by business goals, constraints, requirements




Source: olegvolk.net

To avoid accidental complexity, design happens in feedback loops
– perform task
– check the outcome
– mitigate risks
– repeat

# Architecture Assessments

Architecture Assessments help identify and mitigate risks

Better have short reviews after each increment than a huge review at the end

- because some issues might not be resolvable in later phases

Combine:

- Regular short Reviews after each increment with
- Extended Reviews at important sync points (after implementation, after/before system integration, …)


Better too early than too late
(Principal of Least Surprise)

# In an Agile Context Internal Quality is essential

Quality indicators & Principles

- Economy
- Visibility
- Spacing
- Symmetry
- Emergence

Metrics & Design Smells

- Dependency Cycles
- Cyclomatic Complexity
- Coupling
- Cohesion
- …

This is substantial for good habitability of code base in an Agile Environment

# Improve the Architecture and prepare it for Evolution

| | Refactoring | Reengineering | Rewriting |
|---|---|---|---|
| **Scope** | ▪ Many local effects | ▪ Systemic effect | ▪ Systemic or local effect |
| **Process** | ▪ Structure transforming<br>▪ Behavior / semantics preserving | ▪ Disassembly / reassembly | ▪ Replacement |
| **Results** | ▪ Improved structure<br>▪ Identical behavior | ▪ New system | ▪ New system or new component |
| **Improved qualities** | ▪ Developmental | ▪ Functional<br>▪ Operational<br>▪ Developmental | ▪ Functional<br>▪ Operational<br>▪ Developmental |
| **Drivers** | ▪ Complicated design / code evolution<br>▪ When fixing bugs<br>▪ When design and code smell bad | ▪ Refactoring is insufficient<br>▪ Bug fixes cause rippling effect<br>▪ New functional and operational requirements<br>▪ Changed business case | ▪ Refactoring and reengineering are insufficient or inappropriate<br>▪ Unstable code and design<br>▪ New functional and operational requirements<br>▪ Changed business case |
| **When** | ▪ Part of daily work<br>▪ At the end of each iteration<br>▪ Dedicated refactoring iterations in response to reviews<br>▪ It is the 3rd step of TDD | ▪ Requires a dedicated project | ▪ Requires dedicated effort or a dedicated project, depending on scope |

# Architecture Sustainability

- Symptoms for Lack of Sustainability:
  - Architecture Erosion
  - Architecture Drift
  - Technical Debt
- Complexity needs Governance
- Avoid Task Forces
- Apply Software Architecture Management to align architecture and business
- Discover Problems early by organizing Assessments (Reviews, Metrics, Introspection)

# Ensuring Sustainability

## Sustainability Toolbox

- Architecture Governance including Guidelines and Recommendations
- Regular Architecture Assessments (Internal and External Qualities)
- Re-{Factoring, Engineering, Rewriting}
- Risk-based Testing
- Avoidance of Accidental Complexity (via KiSS principle)
- Emphasis on Quality over Featuritis
- Active Architecture Enforcement
- Leveraging Feedback from other Stakeholders/Experts (Push & Pull)


Source: digitaltrends.com, Police in Dubai

Sustainable Architecture needs Monitoring, Preventive Maintenance, Instant Responsiveness

# Conclusions

# Observations and Experiences

Agility

- is an evolutionary design approach: iterative + incremental + agile practices
- embraces change and risk mitigation
- addresses all disciplines and phases
- leverages architecture as its backbone: when everything changes the architecture baseline should be stable
- Piecemeal growth and Agility perfectly complement each other