SIEMENS

Good is not good enough

- Evaluating and Improving Software Architecture

Keynote COMPARCH/WICSA 2011

Michael Stal

Last chance to escape!

Warning: This presentation is about experiences in large industrial software development projects. It only contains very few scientific insights

Or as an unknown scientist called Einstein once said: In theory, theory and practice are the same. In practice, they are not

But I will provide the first ~~Perpetuum Mobile~~ Silver Bullet for optimizing systems:

SIEMENS

The Silver Bullet

There actually is a silver bullet for boosting software engineering productivity. You just need to shoot the right person

SIEMENS

About myself

- **Principal Engineer, Siemens R&D, Munich (GER)**
  - Coaching & mentoring large projects (currently: Healthcare, Rail IT)
  - Education of Certified (Senior) Software Architects
  - Research in Distributed Systems, Architecture, Product Lines
  - Books (e.g., Pattern-Oriented Software Architecture aka POSA Series)
- **Professor at Rijksuniversiteit, Groningen (NL)**
  - Promoter for Ph.D. students
  - Teaching lectures on Software Architecture
- **Contact:**
  - E-mail: michael.stal@siemens.com
  - Twitter: @MichaelStal
  - Blog: http://stal.blogspot.com

Photo: After the Ph.D. exam and several glasses of wine in Groningen celebrating with my promoter Jan Bosch & my reviewers

SIEMENS

Why we should care

If you think good architecture is expensive, try bad architecture

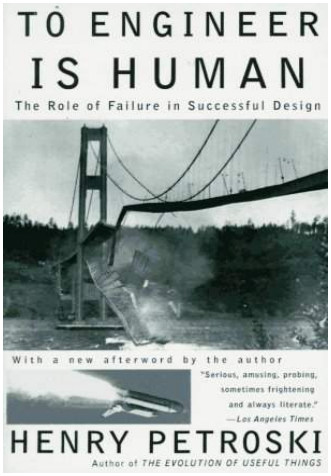
[Brian Foote and Joseph Yoder]

SIEMENS

Learning from failure

Failure and understanding failure is a key factor for successful design!

[Henry Petroski]



SIEMENS

Why should companies care?

- At Siemens, **approximately 60% of current sales** achieved with products, systems, and services **are dependent on software**
- Due to the current trends towards networking and distributed intelligence, **this figure will increase even further**
- As a result, the ability to develop new software efficiently while maintaining market-oriented quality will be increasingly important

SIEMENS

Software Failures cause huge Losses

Some of these failures are related to software architects and software architecture!

Requirements engineering	Ambiguity, lack of precision and understanding of long-term (platform) requirements
Software architecture and development	No, incomplete, or inappropriate architecture description Predominant preference for features and developmental properties such as extensibility, adaptability rather than for operational properties such as stability and performance Architect has insufficient technological breadth and depth – applies the known approach
Testing and quality	Architect views testing, reviews, refactoring, quality measurement as a means of undesirable control rather than as a design approach and quality safety net
Software processes	Architecture specification started before identifying and specifying key business and technical requirements qualitatively and quantitatively Architect does not implement / or implements on the critical path
Business and strategy	No clear (understanding of) business cases
Leadership	Communication with stakeholders and development team often missing or inappropriate Architect does not mentor development team Lack of courage for taking architecture decisions

SIEMENS

Panta rhei - Evolutionary Design embraces Change

There is nothing permanent except change

[Heraclitus, 535–475 BC]

SIEMENS

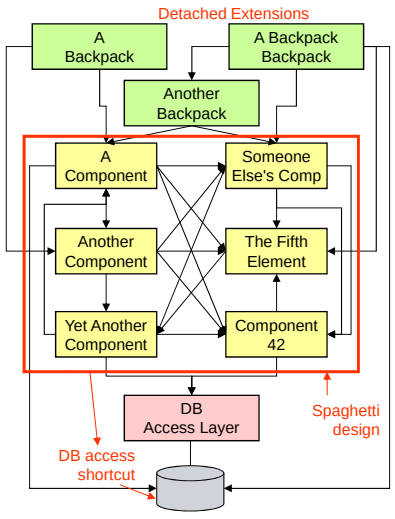
Design erosion is the root of evil

In the lifecycle of a software system changes are the rule and not the exception

Unsystematic approaches ("workarounds") cure the symptom but not the problem

After applying several workarounds, software systems often suffer from design erosion

Such systems are doomed to fail (negative impact on operational & developmental properties)



SIEMENS

How do we know we must improve?

Lack of Internal or External Quality

- Quality Attributes (use methods like ATAM), and,
- Structural quality indicators which include
  - Economy
  - Visibility
  - Spacing
  - Symmetry
  - Emergence
- Consequently, the goal of architecture improvement is to achieve or meet such qualities

SIEMENS

There is a strange Smell

If it stinks, there must be something we need to clean up

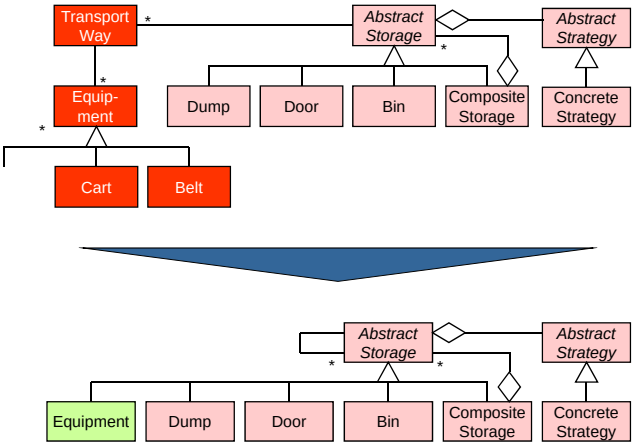
Architecture smells

- Duplicate design artifacts
- Hammer&Nail syndrome
- Unclear roles of entities
- Inexpressive or complex architecture
- Everything centralized
- Home-grown solutions instead of best practices
- Over-generic design
- Asymmetric structure or behavior
- Dependency cycles
- Design violations (such as relaxed instead of strict layering)
- Inadequate partitioning of functionality
- Unnecessary dependencies

SIEMENS

Example of Architecture Problem

A true story: In this example architects introduced Transport Way as an additional abstraction. But can't we consider transport ways as just as another kind of storage? As a consequence the unnecessary abstraction was removed, leading to a simpler and cleaner design.



SIEMENS

Possible Refactoring Pattern

Context

Eliminating unnecessary design abstractions

Problem

Minimalism is an important goal of software architecture, because minimalism increases simplicity and expressiveness

If the software architecture comprises abstractions that could also be considered abstractions derived from other abstractions, then better remove these abstractions

General solution idea

Determine whether abstractions / design artifacts exist that could also be derived from other abstractions

If this is the case, remove superfluous abstractions and derive from existing abstractions instead

Caveat

Don't generalize too much (such as introducing one single hierarchy level: "All classes are directly derived from Object")

SIEMENS

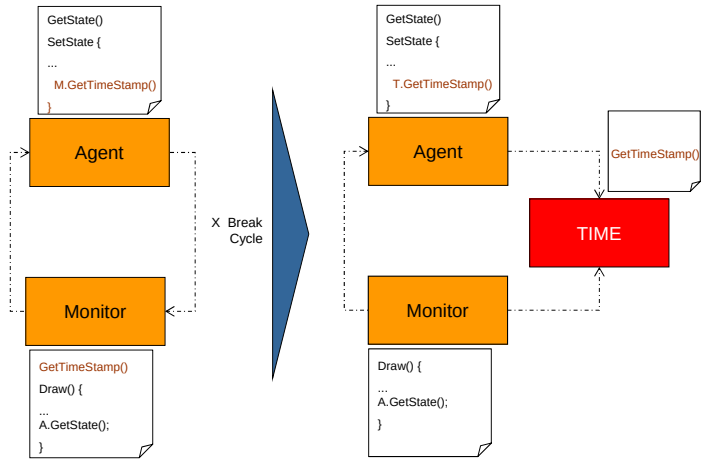
Yet another example

Monitors need to access agents to control the network

Agents change their internal state and create events

To have a common clock architects decided to enhance the monitor

… which leads to a cycle



SIEMENS

Yet Another Refactoring Pattern

Context

Cyclic dependencies between subsystems

Problem

Your system reveals at least one dependency cycle between subsystems

Subsystem A may either depend directly or indirectly on subsystem B (e.g., A depends on C which depends on B) which is why we always need to consider the transitive hull

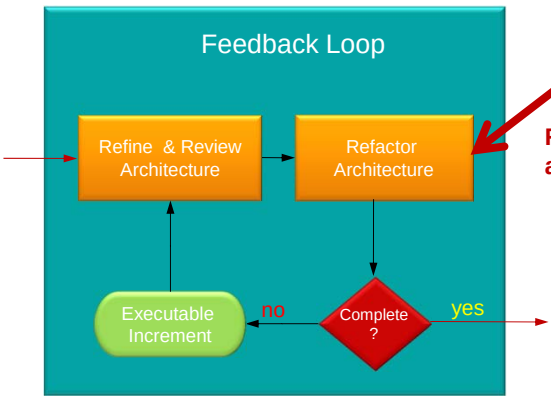
Dependency cycles make systems less maintainable, changeable, reusable, testable, understandable

General solution idea

Get rid of the dependency cycle by removing or inverting dependencies

SIEMENS

Refactoring is part of the architecture design process



Refactoring is integrated into the architecture design process:

- **It improves the structure**
- **It supports a risk-, requirements- and test-driven approach**

SIEMENS

We need to collect Architecture Refactorings

Some Examples

1. Rename Entities
2. Remove Duplicates
3. Introduce Abstraction Hierarchies
4. Remove Unncessary Abstractions
5. Substitute Mediation with Adaptation
6. Break Dependency Cycles
7. Inject Dependencies
8. Insert Transparency Layer
9. Reduce Dependencies with Facades
10. Merge Subsystems
11. Split Subsystems
12. Enforce Strict Layering
13. Move Entities
14. Add Strategies
15. Enforce Symmetry
16. Extract Interface

17. Enforce Contract
18. Provide Extension Interfaces
19. Substitute Inheritance with Delegation
20. Provide Interoperability Layers
21. Aspectify
22. Integrate DSLs
23. Add Uniform Support to Runtime Aspects
24. Add Configuration Subsystem
25. Introduce the Open/Close Principle
26. Optimize with Caching
27. Replace Singleton
28. Separate Synchronous and Asynchronous Processing
29. Replace Remote Methods with Messages
30. Add Object Manager
31. Change Unidirectional Association to Bidirectional

SIEMENS

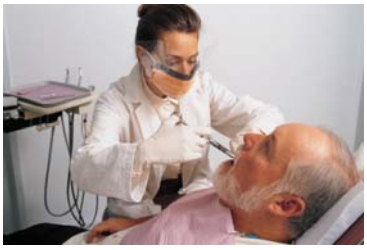
After refactoring check for correctness

To check the correctness of refactorings, we should use a test-driven approach.

Available options:

- Formal approach: Prove semantics and correctness of program transformation
- Implementation approach: Leverage unit and regression tests to verify that the resulting implementation still meets the specification
- Architecture analysis: Check the resulting software architecture for its equivalence with the initial architecture (consider requirements)

Use at least the latter two methods to ensure quality



SIEMENS

Obstacles to Refactoring

Organization / management

- Featuritis: Considering improvement by refactoring as less important than features
- "Organization drives architecture" problem

Process support

- No refactoring activities defined in process
- Refactorings not checked for correctness, test manager not involved

Technologies and tools

- Unavailability of tools: refactoring must be done manually
- Unavailability of refactoring catalog

Applicability

- Refactoring used instead of reengineering
- Wrong order of refactorings

SIEMENS

Reengineering – when and how to use it

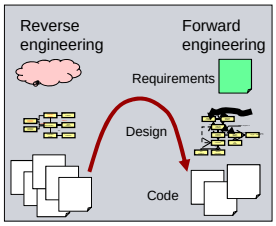
Use Reengineering when

- The system's documentation is missing or obsolete
- The team has only limited understanding of the system, its architecture, and implementation
- A bug fix in one place causes bugs in other places
- New system-level requirements and functions cannot be addressed or integrated appropriately

Process

Phase I: Reverse engineering
- Analysis / recovery: determine existing architecture (consider using CQM)
- SWOT analysis
- Decisions: what to keep, what to change or throw away

Phase II: Forward engineering



SIEMENS

Rewriting in a Nutshell

Rewriting is a radical and fresh restart: existing design and code is trashed and replaced by a whole new design and implementation. Depending on focus:

Improves structure regarding:
- Simplicity, visibility, spacing, symmetry, emergence
- Maintainability, readability, extensibility
- Bug fixing

Provides new functionality

Improves its operational qualities

Improves design and code stability

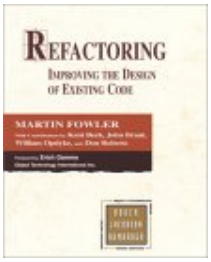
As a consequence, rewriting addresses all types of software quality: functional, operational, and the various developmental qualities

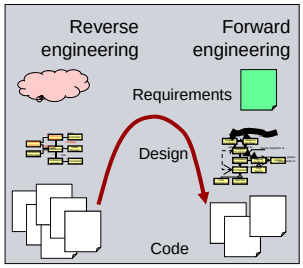
SIEMENS

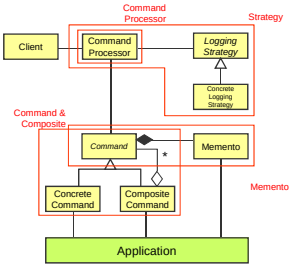
Refactoring, reengineering, and rewriting comparison (1)

Refactoring, reengineering, and rewriting are complementary approaches to sustain architecture and code quality

Start with refactoring – it is cheap and (mostly) under the radar
Consider reengineering when refactoring does not help – but it is expensive
Consider rewriting when reengineering does not help – but it is expensive and often risky







SIEMENS

Refactoring, reengineering, and rewriting comparison (2)

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

SIEMENS

Software Architect's Dilemma

Life must be understood backwards; but ... it must be lived forward

[Søren Aabye Kierkegaard, Danish philosopher and theologian, 1813-1855]

SIEMENS

Reviews help finding the Bad Smells

Quantitative Architecture Reviews

- Code quality assessment
- Simulations
- Prototypes

Qualitative Architecture Reviews

- Scenario-based approaches
- Experience-based approaches

An Architecture Assessment or Review should not be considered an afterthought.

It is a means to check a system regularly and find problems early

SIEMENS

Quantitative review

Benefits

- Yield "hard" results
- Quantifiable, objective means for selecting alternatives
- Experiments by altering the parameters relatively easy

Liabilities

- Focus on only a couple of concerns or system parts
- Works only if data is interpreted correctly
- Effect on quality attributes other than the focus is unknown
- Probably costly

Similar to test automation, the initial cost might be high, but is typically justified by early detection of conceptual faults.

SIEMENS

Qualitative review

Benefits

- Involves all relevant stakeholders
- Overview of the whole system
- Improve understanding for all participants
- Relatively cheap to execute
- Can be conducted as soon as high level architecture design is available

Liabilities

- Relies mainly on documents and statements from personally involved stakeholders
- Experienced reviewers required
- No "hard facts" (unless supported by quantitative assessments)

Visualization Tools help keeping the system in good Shape

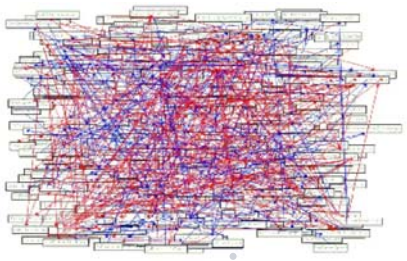
SIEMENS

In many projects the responsibility for internal code and design quality is not well defined

The software architect has to ensure that the required CQM activities are established

The software architect should be the protector of the quality of the software system!

Use Visualization Tools at least in larger code bases



By the way: this is a real system

Page 29 Keynote WICSA/COMPARCH 2011

A (Personal) Note on Metrics

SIEMENS

Metrics represent a popular tool to measure software quality:

Metrics can be clearly measured

Lots of metrics exist

Lots of literature on metrics

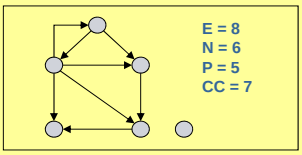
However metrics rarely help to measure the quality of a piece of design/code!

What is the meaning of LOC-based metrics?

What is the meaning of a low cyclomatic complexity value in the presence of a complex subject—or vice versa?

At best, a metric can indicate a quality problem, but it cannot ultimately confirm it

The cyclomatic complexity (by McCabe) of a software module is calculated from a connected graph of the module (that shows the topology of control flow within the program):
Cyclomatic complexity (CC) = E - N + p where:
E = the number of edges of the graph
N = the number of nodes of the graph
p = the number of connected components



A common interpretation of the CC metric is
01-10 – a simple program, without much risk
11-20 – more complex, moderate risk
21-50 – complex, high risk program
> 50 – untestable program, very high risk

According to the above interpretation, an Observer arrangement with a subject and 50 observer types has a CC of 50. Is this a complex, high risk structure? Not really, an Observer structure is clearly defined, very stable, testable, and has a manageable complexity!

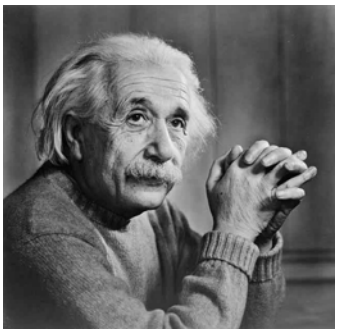
Page 30 Keynote WICSA/COMPARCH 2011

SIEMENS

"Preventive Maintenance"

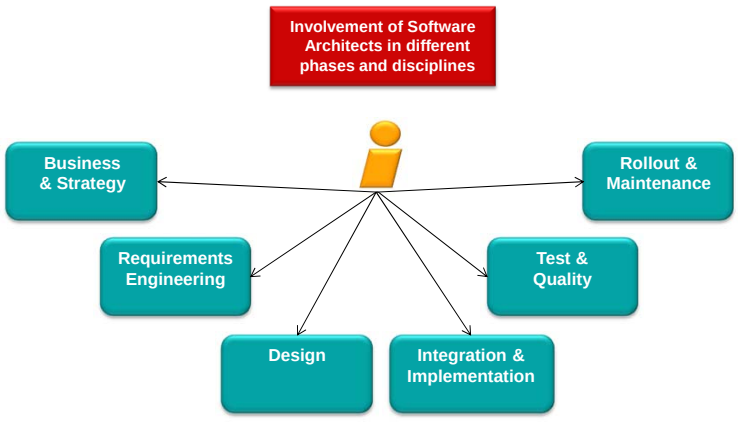
Experts solve problems, geniuses avoid them

[Albert Einstein]



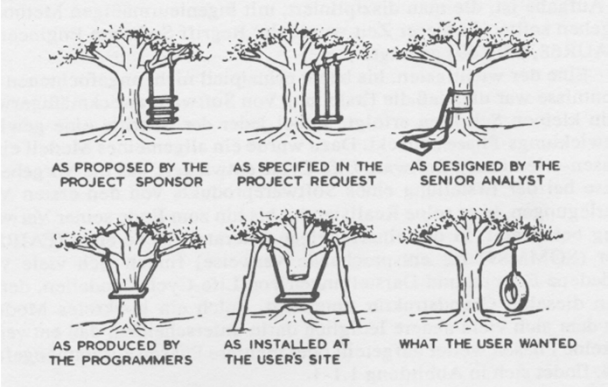
SIEMENS

Architecture Quality is also influenced by other aspects



Architects & Requirements – Problem 1: There are always different views on a requirement

SIEMENS



Architects & Requirements – Problem 2: Do we have the *right* requirements?

SIEMENS

A program which perfectly meets a lousy specification is a lousy program

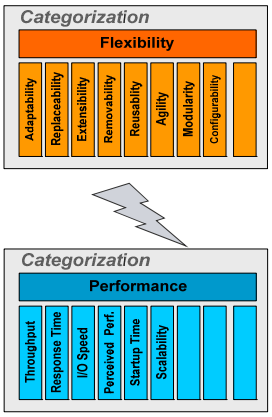
[Cem Kaner, Software Engineering Professor and Consumer Advocate]

SIEMENS

=> Requirements must have high Quality

Quality of Requirements determines Quality of Software Architecture

Cohesive
Complete
Consistent
Correct
Current
Externally Observable
Feasible
Unambiguous
Mandatory
Verifiable



SIEMENS

No Risk – No Fun?

The most likely way for the world to be destroyed, most experts agree, is by accident. That's where we come in; we're computer professionals. We cause accidents

[Nathaniel Borenstein, US Programmer]

SIEMENS

Mind all Risks and conduct a Risk Analysis early

Approach for risk analysis according to Christine Hofmeister ("Applied Software Architecture"):

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 technology 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

SIEMENS

Real engineers in action

SIEMENS

The Art of Architecture

There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult

[C.A.R Hoare]

SIEMENS

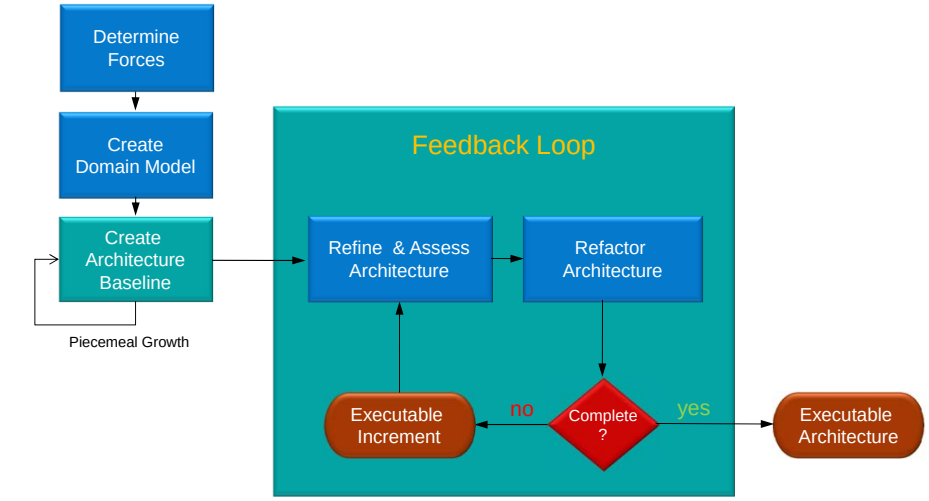
Architecture versus Design

- Design is a continuous activity of making decisions
  - beginning with decisions that have broad system wide scope (**Strategic Design**), and
  - moving to decisions that have very narrow scope (**Tactical Design**)
- A decision is architectural if it has one or more of the following properties:
  - it has system wide impact
  - it affects the achievement of a quality attribute important to the system

Architecture is about everything costly to change [Grady Booch]

Piecemeal Growth - An Iterative-Incremental Architecture Design Process from 10000 feet

SIEMENS

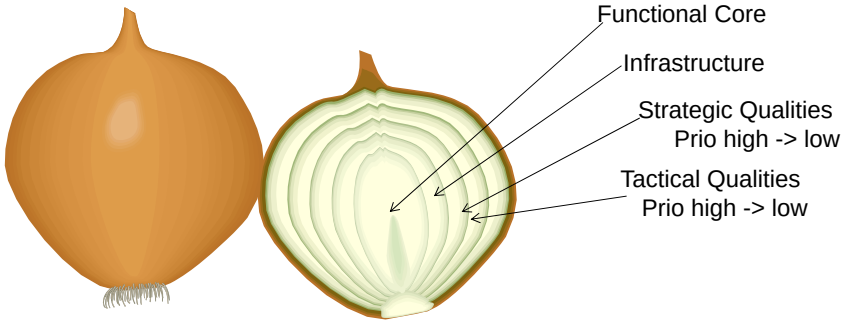


The Onion Model

SIEMENS

Architecture design follows an onion metaphor:

start with the inner core,
incrementally continue with outer layers



How deep should we go in the Baseline Architecture

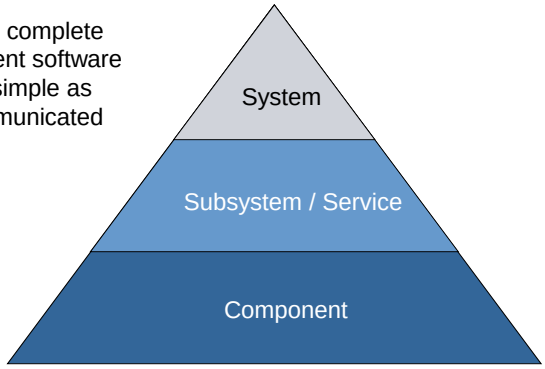
SIEMENS

Pyramid Model

The baseline architecture must be as complete as necessary to govern the subsequent software development, but it must also be as simple as possible, otherwise it cannot be communicated

Three levels of detail to limit depth

Focus on architecturally significant requirements and corresponding architecture views to limit breadth



SIEMENS

The Not Invented Here Syndrome

Human beings, who are almost unique in having the ability to learn from the experience of others, are also remarkable for their apparent disinclination to do so.

[Douglas Adams. 1952-2001.
Last Chance to See]

SIEMENS

Think Green and Re-Use

Re-use design concepts:

- In most cases efficiency and effectiveness are more important than originality
- Proven expertise instead of inventing new solutions – Mind the NIH Syndrome!

Different Levels of re-use:

- Idioms and Design Patterns
- Domain specific Patterns – Analysis Patterns
- Patterns for using a Technology
- Architectural blueprints – reference architectures

SIEMENS

Also keep in mind the issue of Architecture Governance

Without Architecture Governance the System is subject to uncontrolled Change and Extension

Introduce countermeasures such as:

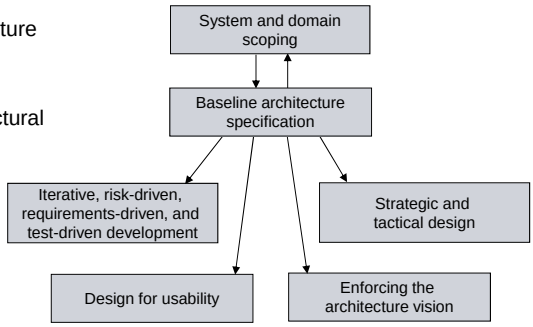
- Architecture Guidelines and Policies as well as their Enforcement
- Means to ensure Requirements Traceability
- No Checking-in without other Persons reviewing Code and Documents
- Test-Driven-Design
- Risk-Based Analysis & Test

SIEMENS

An Architect's Toolbox as a common Mindset

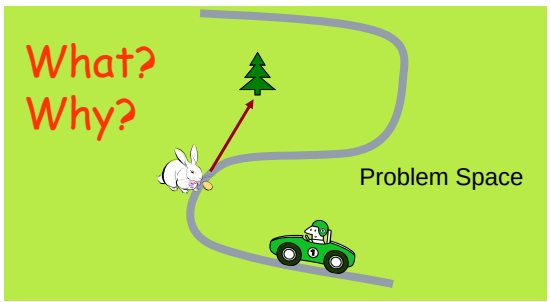
The mindset, activities, practices, methods, and technologies for defining and realizing software architectures form a best practice framework for senior software architects to

- Specify and implement a software architecture systematically and in a timely fashion
- Check and ensure the appropriate architectural quality
- Respond to changes of all kinds, such as changing requirements and priorities
- Deal with problems that arise during the definition and realization of the software architecture



SIEMENS

Give the client not what he wants, ….



- **Customer**: "Whenever, I am driving the road, this rabbit will cross the street. Trying not to run down the animal in the curve, I always hit the tree. Please, sell me a car with big bumpers"

SIEMENS

…, but what he needs

Solution Space

- **Car seller**: "You don't need a car with big bumpers, but ABS to stabilize your car in the curve"

Keynote WICSA/COMPARCH 2011

SIEMENS

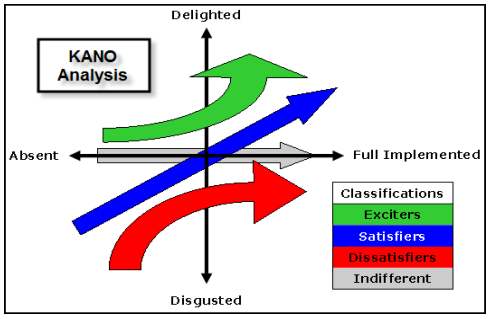
Knowing the expectations is essential

At least at project begin,

- Architects don't understand requirements very well
- Customers tell what they want, not what they need
- Architects may even not know the implicit requirements

Hence,

- Keep in touch with Customers
- Apply a KANO Analysis
- Understand your Business Goals
- Develop Design and Requirements in parallel



Page 50 Keynote WICSA/COMPARCH 2011

SIEMENS

Testing as a never ending story

Testing is an infinite process of comparing the invisible to the ambiguous in order to avoid the unthinkable happening to the anonymous

[James Bach, Test Guru]

SIEMENS

Communication is essential

Software Development is a collaborative game

[Alistair Cockburn]

Architects need to tightly interact with all other roles in software development

SIEMENS

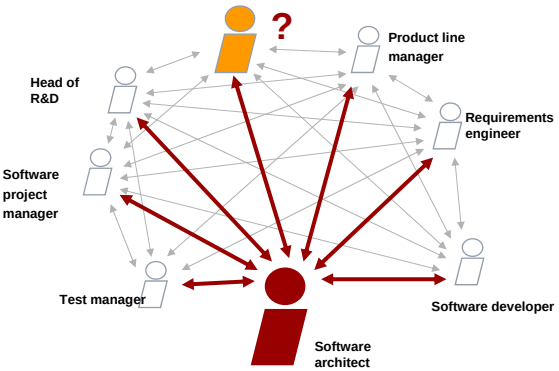
Leadership, and communication and interaction with other roles in software development, are probably the most time-intensive and most important responsibilities Software Architect

The roles with whom the architect interacts,

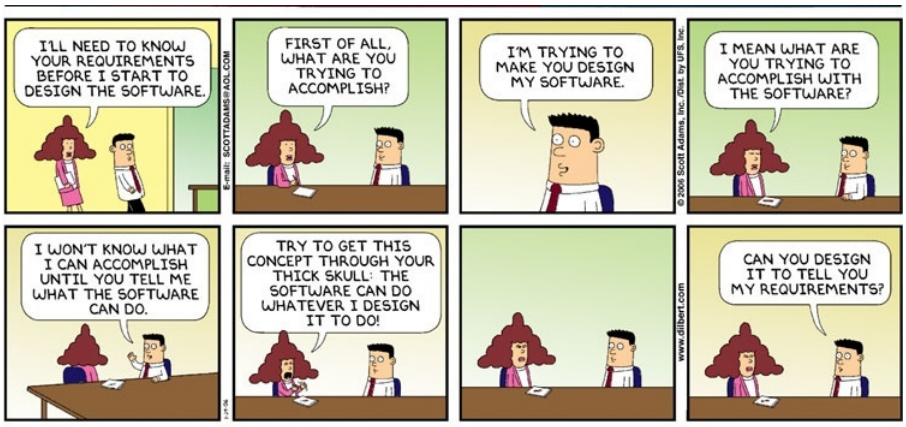
the topics about which they interact with these roles,

and the intensity of the interaction

depend on the concrete development workflow and activity performed in a software project



Communication with stakeholders can be surprisingly challenging



© Scott Adams, Inc./Dist. by UFS, Inc.

SIEMENS

Conclusions

- Architecture Improvement should be considered in the whole lifecycle, not at the end
- The later a problem is detected, the more expensive or impossible to get rid of it
- Conduct Assessment, Refactoring, Testing activities regularly
- Appropriate design methods help avoid problems
- KiSS Principle helps prevent accidental complexity
- High Software Architecture Quality impossible without involving other stakeholders
- An underestimated tool is effective Communication
- Research opportunities in terms of refactoring, visualization, modeling

It is your responsibility as architects to build sustainable systems!

SIEMENS

A departing thought

Each problem that I solved became a rule which served afterwards to solve other problems.

[René Descartes, 1596–1650, in "Discours de la Methode"]

SIEMENS