Introduction to Software Architectures

Mikael Svahnberg[1]

2017-04-03

[1] Mikael.Svahnberg@bth.se

SE Challenges

- Reduce Development Cost – Deliver on time, within budget
- Increase System Quality – . . . with the *right* quality
- Decrease Maintenance Costs
- Reduce Time-to-Market

Decisions

- Balancing all stakeholders result in a number of *Business and Technical Decisions*
- Software architecting is about identifying which decisions are necessary, and finding solutions that satisfy all stakeholders.

Decisions *are* the Architecture

I would go as far as to say that these decisions *are* the architecture! The rest is just an instantiation of the architecture.

How can the Architecture Help?

The architecture is a tool for

- Understanding
- Planning
- Communicating
- Predicting and Evaluating Quality
- Identifying Reuse

Influences on Architecture

- Customer Requirements, of course
- Developing Organisation
  - e.g., business goals
  - Organisational structure
  - Available expertise (the architect's experience)
- Technical Environment

The Architecture Business Cycle

The architecture is influenced by stakeholders, developing organisation, technical environment, architect's experience, etc.
Likewise, the architecture influences all of the above.
A software architecture

- Manifests early design decisions
- Constrains an implementation
- Dictates organisational struture
- Impacts change management

A developed system

- Adds to previous experience
- May affect business goals
- Refines development organisation (Process improvement)
- Affects customer requirements
- Provides a reusable software base

A “Good” Software Architecture

- Is based on *conscious* decisions
- Is *evaluated* to ensure that it satisfies the specific goals for the system
- Pays attention to current and future *quality attributes*
- Is well *documented*, with traceability to the architecture decisions
- Features well defined *modules/(components), with well defined /interfaces* and well defined *responsibilities*
- Is restricted to a small set of interaction patterns that are *consistently* used.

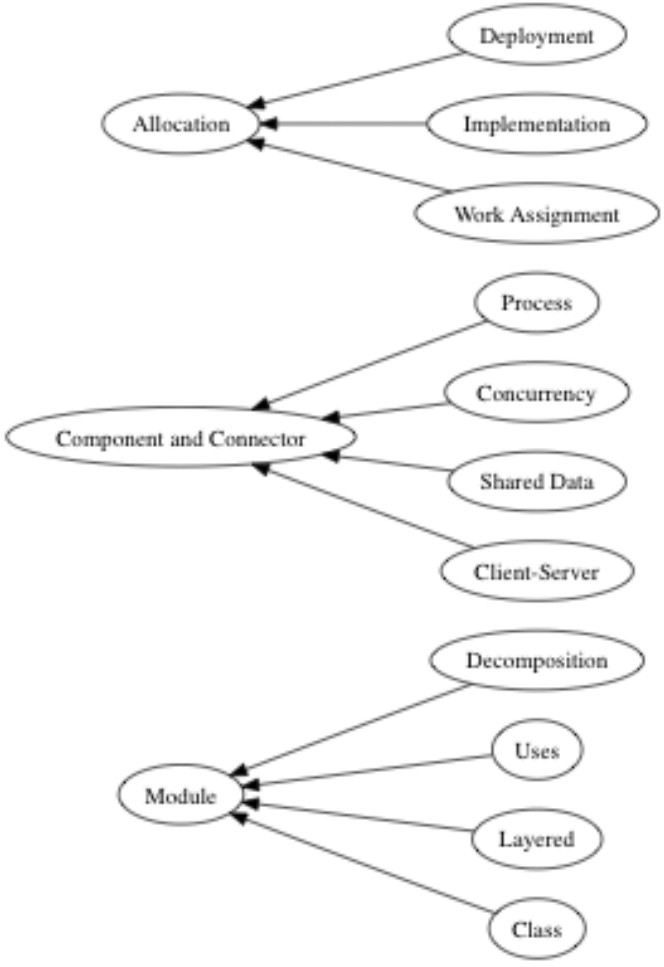
Software Architecture Defined

- First Definition: Boxes and Lines
  - What is the nature of the elements (boxes)?
  - What are the responsibilities of the elements?
  - What is the siginificance of the connections (lines)?
  - What is the significance of the layout?
- Second Definition: Add semantics (provide legend)
  - What is the significance of the layout?
  - What are the interfaces of the elements?
  - How does the architecture operate at runtime?
  - How do we build it?
- Third Definition:

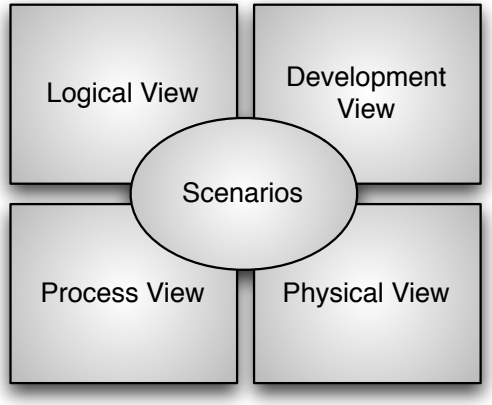
  The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Structure and Views

Bass et al. (2012)



Kruchten (1994):



Hofmeister et al (2000):

- Conceptual View
- Module View
- Execution View
- Code View
- Global Analyis (as scenarios)

Which View to Start with?

None. Remember: `Architecture === Decisions`

1. What requirements will have an architectural impact?
2. What strategies do you have to address these requirements?
3. *Then* you decide which viewpoint to address them in.

Architecture Styles and Patterns

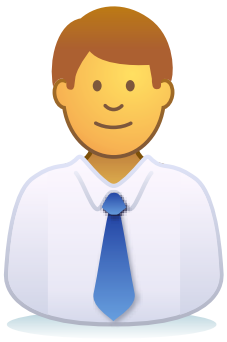
- “Design Patterns for Software Architectures”
- Common styles:
  - Layered
  - Pipes and Filters
  - Model View Controller
  - Centralised vs Distributed
  - Microkernel
  - Blackboard
  - Broker

Examples of Systems

List and discuss examples of systems that uses each style.

Quality Attribute Scenarios

- Quality Attribute Scenarios:



Source:
Users

Stimulus:
Initiate
Transactions

Environment:
Under normal
operations

Response:
Transactions
are processed

Response
Measure:
With an average
latency of two
seconds

Question

How can you – at an architecture level – solve this?

Architecture Tactics

- Book; Bass, Clements and Kazman, *Software Architecture in Practice*.
- A tactic is a generic solution for addressing a (common) quality attribute).
- Remember `Architecture === Decisions`:
  1. What requirements will have an architectural impact?
  2. What strategies do you have to address these requirements?
  3. *Then* you decide which viewpoint to address them in.

Example Architecture Tactics: Availability

Purposes of Architecture Evaluation

- Early Architecture Evaluation [2]
  - Do we meet the quality requirements on the system?
  - Do all stakeholders share a common understanding of the system?
  - Are all requirements accounted for?
  - Are there any weak spots in ther architecture?
  - Can the system (and/or the architecture) be improved?
  - Does the development team have all the necessary resources?
  - Should we let this project continue?
- Late Architecture Evaluation
  - Hard metrics.
  - How did we do? What needs to be improved for the next release?

[2] M. Lindvall, R. T. Tvedt, and P. Costa. An empirically-based process for software architecture evaluation. *Empirical Software Engineering*, 8:83–108, 2003.

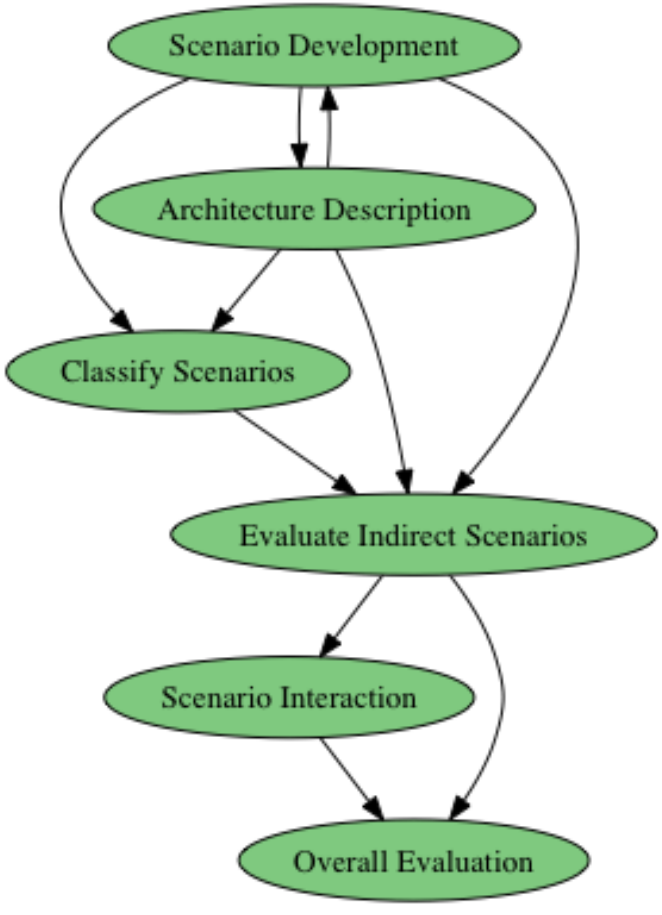
Early Architecture Evaluation Methods

- Experiences and Logical Reasoning
- Scenario-Based
  - Examples: SAAM, ATAM, Global Analysis, BTH-way
- Simulation-based
  - Build parts of the architecture / Build models of the architecture
  - Architecture description languages
  - Examples: AADL, Aesop, ArTek, C2, Darwin, LILEANNA, MetaH, Rapide, SADL, UniCon, Weaves, Wright, ACME, . . .
- Based on Mathematical models
  - Often domain-specific
  - Example: ABAS (ATAM)
- Other
  - Example: Software Inspections

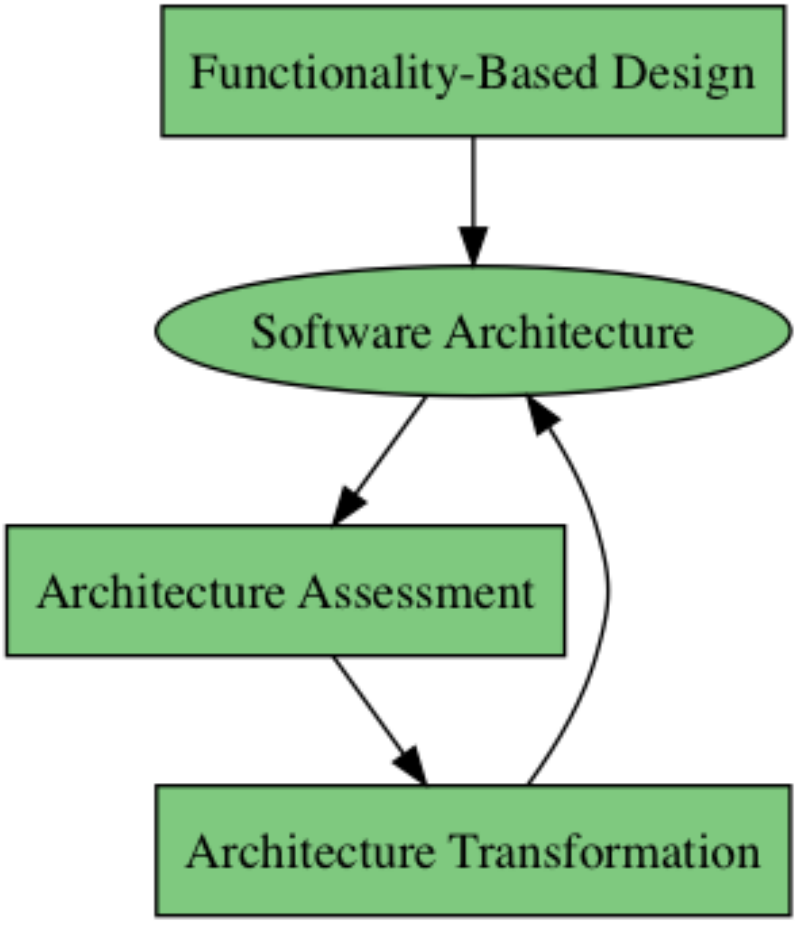
SAAM: Software Architecture Analysis Method

R. Kazman, L. Bass, M. Webb, and G. Abowd. Saam: A method for analyzing the properties of software architectures. In *Proceedings of the 16th international conference on Software engineering*, pages 81–90. IEEE Computer Society Press, 1994.

1. Develop Scenarios
2. Describe Candidate Architecture
3. Classify Scenarios
   - Directly supported vs. Indirectly supported
4. Perform Scenario Evaluations for indirect scenarios
   - Needed changes to support scenario, cost of changes.
   - C.f. Architecture Transformations
5. Reveal Scenario Interaction
   - Scenarios requiring changes in the same components $\rightarrow$ component overload?
6. Overall Evaluation
   - Prioritise scenarios



QASAR


Functionality-Based Design
Software Architecture
Architecture Assessment
Architecture Transformation

BTH 4-hour Architecture Evaluation

Step	Name	Time
1	Introduction	10 min
2	Identify Quality Requirements	30 min
3	Elicit Scenarios	50 min
4	Architecture presentation	2 x 15 min
5	Break	20 min
6	Scenario and Architecture Analysis	2 x 40 min
7	Conclusion	15 min

Evaluation Experiences

- Size of Evaluation Team [3]
- Clear Objective
- Present Recipients of Evaluation Document
- Moderate Pursuit of Issues
- Use Extreme Scenarios
- The Impact of the Project Manager
- Summarise Often

Also:

- Guidance – not Lecturing
- Avoid Grading Tension
- Make the Architecture Matter
- Encourage Peer Evaluation

[3] M. Svahnberg and F. Mårtensson. Six years of evaluating software architectures in student projects. *The Journal of Systems & Software*, 80(11):1893–1901, 2007.

Size of Evaluation Team

- 3-4 persons in the Evaluation Team
  - Fewer is harder
  - More may intimidate the evaluatees
- Task division:
  - One person documents
  - The others take turn in thinking / pursuiting issues

Clear Objective of Evaluation

Including: Present Recipients of Evaluation

- Open target == no end criterion
- Need clear objective to decide on most appropriate method and most appropriate participants
- Avoid objective guessing
  - For You are here to fail us and stop the project!/
- Make sure there are clear benefits for the project

Moderate Pursuit of Issues

- Conflict: You are there to find flaws, but if you do not know when to “let go” the project will become defensive and uncooperative.
- Knowing when to back down is not only a technical skill.
- Difficult to identify and investigate all ripple effects.
- $\rightarrow$ Leave warnings in the evaluation documentation.

Use Extreme Scenarios

- The absurd may jolt the project into defining limits.
- Typically, investigate one normal scenario and several extremes, where the boundaries of the requirements are probed.
- Be open to pursuit promising paths, e.g. with even more extreme scenarios even if you had not initially planned them.

The Impact of the Project Manager

- It is *absolutely vital* that the project manager understands the benefits of the evaluation.
- The project manager is the lest technical of the project members (?)
- Perceives pressure from mid-level management to produce
- Group issues: Do the other project members dare speak up against their project manager?

Summarise Often

- Keep the evaluation and the project “on track”
- Make sure that found issues are clearly presented and understood.