Architecture Evaluations and Transformations

L05 PA1308

Mikael Svahnberg[1]

[1]Mikael.Svahnberg@bth.se
School of Computing
Blekinge Institute of Technology

September 19, 2012

Purpose of Architecture Evaluations

- Early Architecture Evaluation[1]
  - 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?

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

Purpose of Architecture Evaluations

- Early Architecture Evaluation[1]
  - 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?

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

Purpose of Architecture Evaluations

- Early Architecture Evaluation[1]
  - 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?

[1] 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

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

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

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

Early Architecture Evaluation Methods

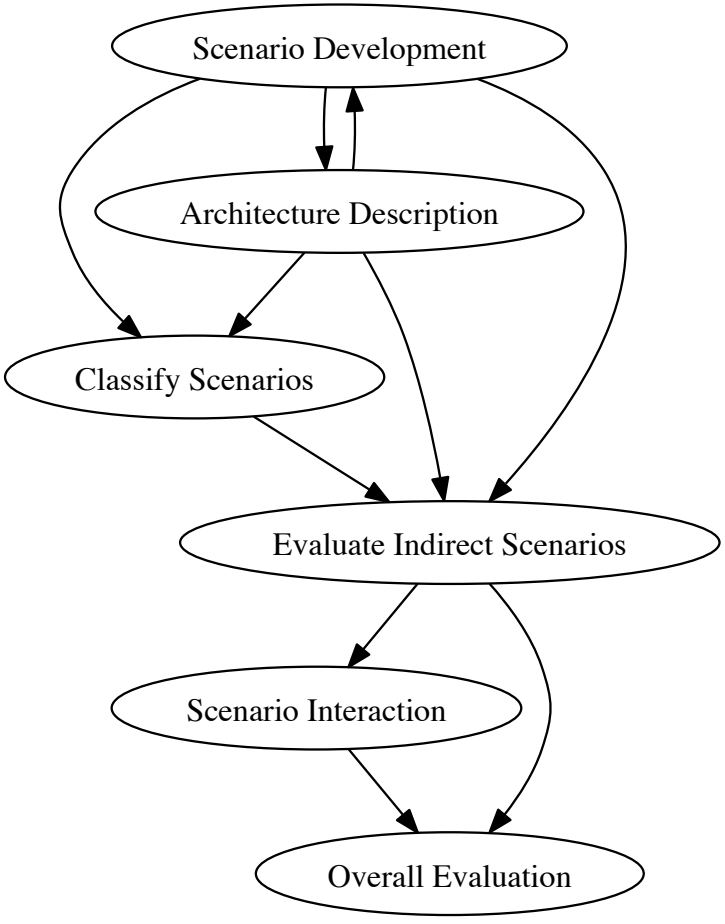
- Experiences and Logical Reasoning
- Scenario-Based
  - Examples: SAAM, ATAM, Global Analysis, BTH-way
- Simulation-based
  - Build parts of the architecture / Build models of the architecture
  - Architecture description languages
  - Examples: AADL, Aesop, ArTek, C2, Darwin, LILEANNA, MetaH, Rapide,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

SAAM: Software Architecture Analysis Method I

1. Develop Scenarios[2]
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

[2] 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.

SAAM: Software Architecture Analysis Method II



ATAM: Attribute Tradeoff Analysis Method

- Based on SAAM
- Focus on quality attributes
- Participants:
  - Evaluation Team: Team Leader, Evaluation Leader, Scenario Scribe, Proceedings Scribe, Timekeeper, Process Observer, Process Enforcer, Questioner
  - Project Decision Makers: Project manager, Customer?, Architect, Commissioner of Evaluation
  - Architecture Stakeholders: Developers, Testers, Integrators, Maintainers, Performance Engineers, Users, Builders of Related Systems, etc.
- Outputs:
  - A concise presentation of the architecture
  - Articulation of business goals
  - Quality requirements, expressed as scenarios
  - Mapping Architectural decisions to quality requirements
  - Set of identified sensitivity and tradeoff points
  - Set of risks and nonrisks
  - Set of risk themes

ATAM: Attribute Tradeoff Analysis Method

- Based on SAAM
- Focus on quality attributes
- Participants:
  - Evaluation Team: Team Leader, Evaluation Leader, Scenario Scribe, Proceedings Scribe, Timekeeper, Process Observer, Process Enforcer, Questioner
  - Project Decision Makers: Project manager, Customer?, Architect, Commissioner of Evaluation
  - Architecture Stakeholders: Developers, Testers, Integrators, Maintainers, Performance Engineers, Users, Builders of Related Systems, etc.
- Outputs:
  - A concise presentation of the architecture
  - Articulation of business goals
  - Quality requirements, expressed as scenarios
  - Mapping Architectural decisions to quality requirements
  - Set of identified sensitivity and tradeoff points
  - Set of risks and nonrisks
  - Set of risk themes

Phases of ATAM

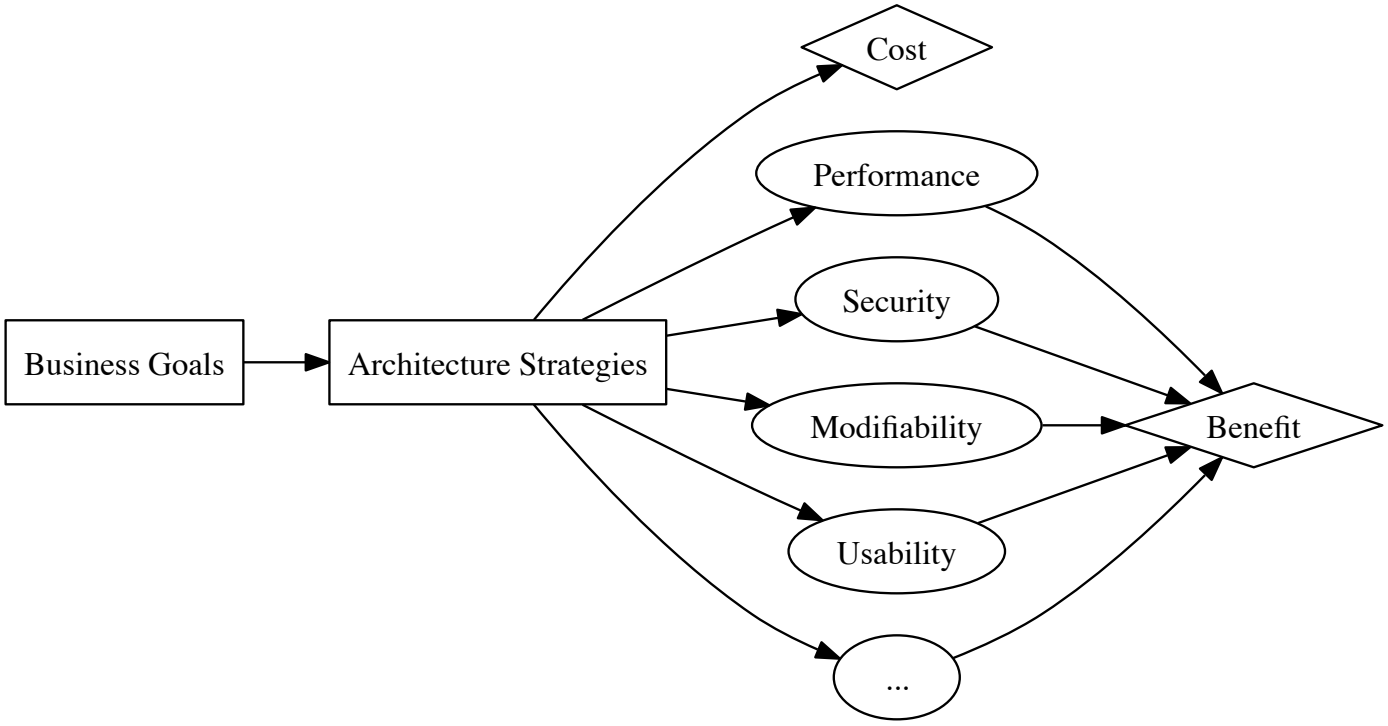
- Phase 0: Partnership and preparation (Duration: Informally, a couple of weeks)
- Phase 1: Evaluation (Evaluation team, project decision makers) (Duration: 1 day plus hiatus of 2-3 weeks)
  - Step 1: Present ATAM
  - Step 2: Present Business Drivers
  - Step 3: Present Architecture
  - Step 4: Identify Architectural Approaches
  - Step 5: Generate Quality Attribute Utility Tree
  - Step 6: Analyse Architectural Approaches
- Phase 2: Evaluation (Continued) (Evaluation team, project decision makers, stakeholders) (Duration: 2 days)
  - Step 7: Brainstorm and Prioritise scenarios
  - Step 8: Analyse Architectural Approaches
  - Step 9: Present Results
- Phase 3: Follow-Up (Evaluation team, evaluation client) (Duration: 1 week)

Phases of ATAM

- Phase 0: Partnership and preparation (Duration: Informally, a couple of weeks)
- Phase 1: Evaluation (Evaluation team, project decision makers) (Duration: 1 day plus hiatus of 2-3 weeks)
  - Step 1: Present ATAM
  - Step 2: Present Business Drivers
  - Step 3: Present Architecture
  - Step 4: Identify Architectural Approaches
  - Step 5: Generate Quality Attribute Utility Tree
  - Step 6: Analyse Architectural Approaches
- Phase 2: Evaluation (Continued) (Evaluation team, project decision makers, stakeholders) (Duration: 2 days)
  - Step 7: Brainstorm and Prioritise scenarios
  - Step 8: Analyse Architectural Approaches
  - Step 9: Present Results
- Phase 3: Follow-Up (Evaluation team, evaluation client) (Duration: 1 week)

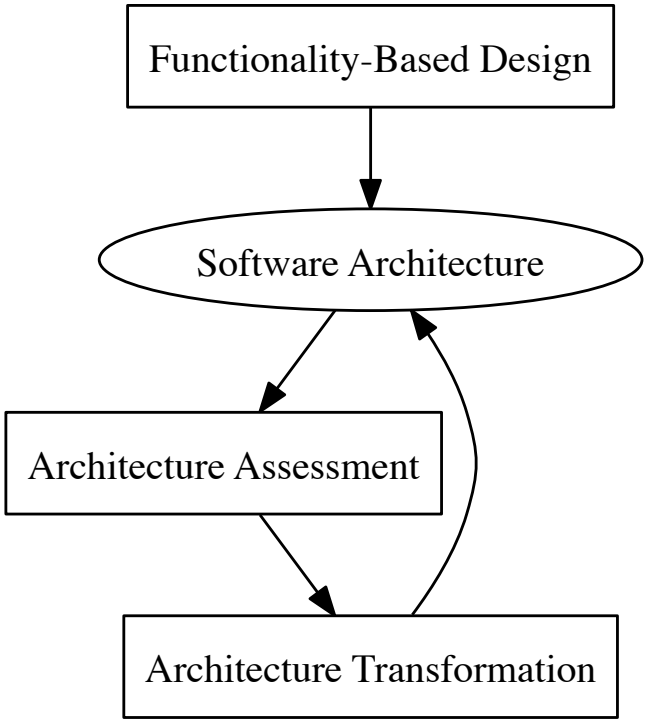
CBAM: Cost Benefit Analysis Method

- Takes off where ATAM ends.
- What is the *utility* $= (\frac{B_i}{C_i})$ of supporting a quality attribute better?



QASAR

J. Bosch. *Design & Use of Software Architectures - Adopting and Evolving a Product Line Approach*. Addison-Wesley, Harlow UK, 2000.



BTH 4-hour Architecture Evaluation

- Shorter variant of SAAM/ATAM.
- Used primarily for student projects.
- Have been used with industry partners as well.
- Will be used by you in this course.

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

Evaluation Experiences

- Size of Development Team[3]
- Clear Objective
- Present Recipients of Evaluation Document
- Moderate Pursuit of Issues
- Use Extreme Scenarios
- The Impact of the Project Manager
- Summarise Often
- 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
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 example, *"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 identifu 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.

Architecture Transformations

- Once you have identified a weak spot in the architecture, you change it.
  - . . . provided, of course, that the ROI of changing the architecture makes it worthwile.
  - . . . and also provided that you cannot change the requirements instead.
- Bosch 2000 labels this modification of the architecture as *architecture transformation*.
- The idea is to modify the architecture while keeping the domain functionality invariant
  - thus only impacting the quality attributes of the system.
- A transformation will improve some quality attributes but may detereorate others.

Steps:

1. Identify which quality requirements that are not yet fulfilled
2. Identify at what components or locations in the architecture is the quality attribute inhibited
3. Select transformation that will solve the identified problem spots in the

Architecture Transformations

- Once you have identified a weak spot in the architecture, you change it.
  - . . . provided, of course, that the ROI of changing the architecture makes it worthwile.
  - . . . and also provided that you cannot change the requirements instead.
- Bosch 2000 labels this modification of the architecture as *architecture transformation*.
- The idea is to modify the architecture while keeping the domain functionality invariant
  - thus only impacting the quality attributes of the system.
- A transformation will improve some quality attributes but may detereorate others.

Steps:

1. Identify which quality requirements that are not yet fulfilled
2. Identify at what components or locations in the architecture is the quality attribute inhibited
3. Select transformation that will solve the identified problem spots in the

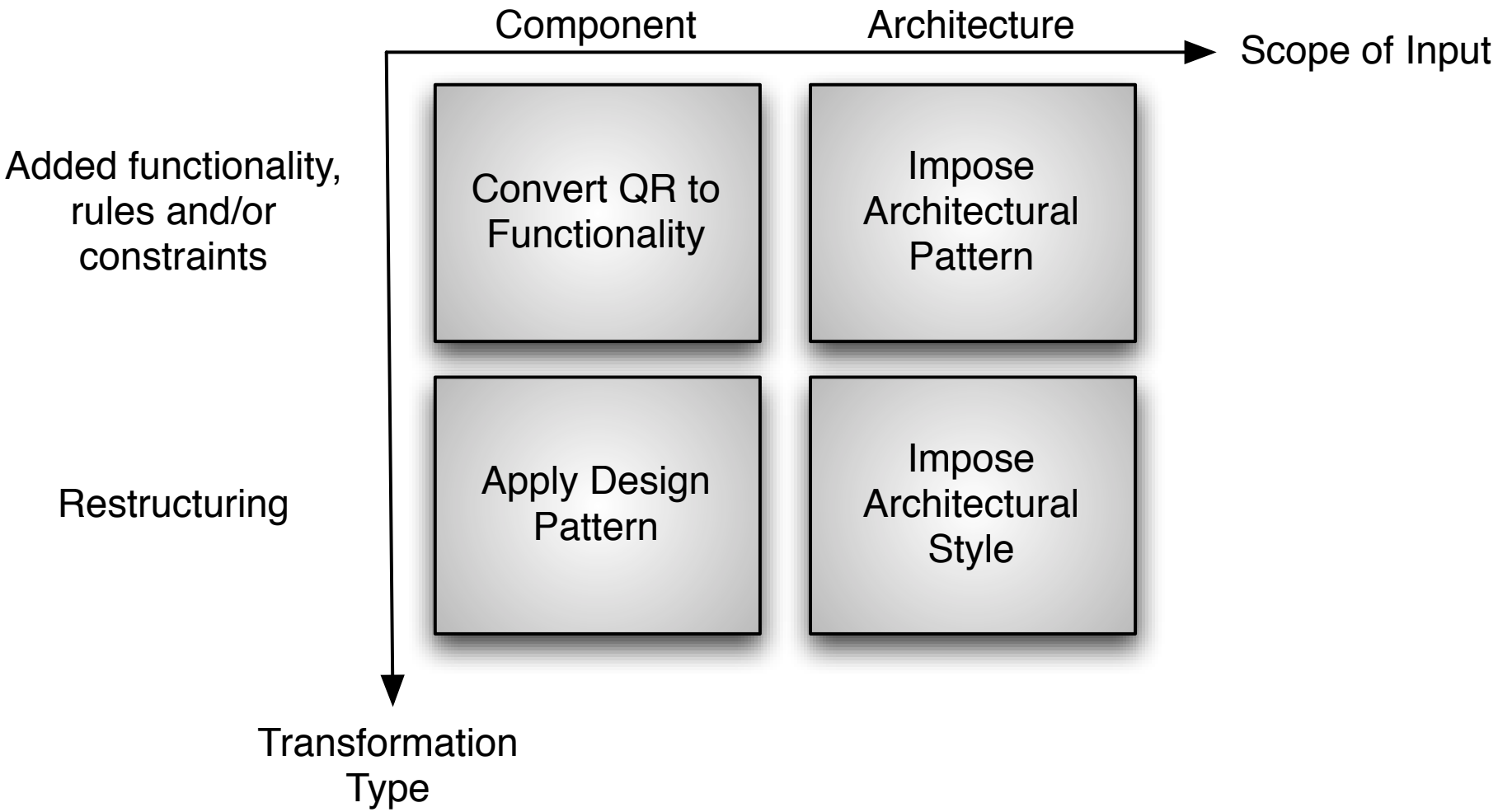
Architecture Transformations

- Once you have identified a weak spot in the architecture, you change it.
  - . . . provided, of course, that the ROI of changing the architecture makes it worthwile.
  - . . . and also provided that you cannot change the requirements instead.
- Bosch 2000 labels this modification of the architecture as *architecture transformation*.
- The idea is to modify the architecture while keeping the domain functionality invariant
  - thus only impacting the quality attributes of the system.
- A transformation will improve some quality attributes but may detereorate others.

Steps:

1. Identify which quality requirements that are not yet fulfilled
2. Identify at what components or locations in the architecture is the quality attribute inhibited
3. Select transformation that will solve the identified problem spots in the

Types of Transformations



Bosch 2000

Types of Transformations

From Biggest to Smallest:

- Impose Architectural Style
  - Reorganise the entire architecture (of the entire system *or* a specific sub-system)
  - Examples: Pipes-and-Filters, Layered, Model-View-Controller, Blackboard, etc.
- Impose Architectural Pattern
  - Maintain the components in the architecture and their functionality, but change their behaviour.
  - Examples: Introduce Concurrency, Persistence, Distributed Communication Mechanisms
- Apply Design Pattern
  - Reorganises a single component, or a small set of components according to the principles of a design pattern.
  - Examples: Facade, Observer, Abstract Factory, Strategy.
- Convert QR to Functionality
  - Add functionality, not primarily from the application domain.
  - Examples: Exception handling, self-monitoring, undo history.

Types of Transformations

From Biggest to Smallest:

- Impose Architectural Style
  - Reorganise the entire architecture (of the entire system *or* a specific sub-system)
  - Examples: Pipes-and-Filters, Layered, Model-View-Controller, Blackboard, etc.
- Impose Architectural Pattern
  - Maintain the components in the architecture and their functionality, but change their behaviour.
  - Examples: Introduce Concurrency, Persistence, Distributed Communication Mechanisms
- Apply Design Pattern
  - Reorganises a single component, or a small set of components according to the principles of a design pattern.
  - Examples: Facade, Observer, Abstract Factory, Strategy.
- Convert QR to Functionality
  - Add functionality, not primarily from the application domain.
  - Examples: Exception handling, self-monitoring, undo history.

Types of Transformations

From Biggest to Smallest:

- Impose Architectural Style
  - Reorganise the entire architecture (of the entire system *or* a specific sub-system)
  - Examples: Pipes-and-Filters, Layered, Model-View-Controller, Blackboard, etc.
- Impose Architectural Pattern
  - Maintain the components in the architecture and their functionality, but change their behaviour.
  - Examples: Introduce Concurrency, Persistence, Distributed Communication Mechanisms
- Apply Design Pattern
  - Reorganises a single component, or a small set of components according to the principles of a design pattern.
  - Examples: Facade, Observer, Abstract Factory, Strategy.
- Convert QR to Functionality
  - Add functionality, not primarily from the application domain.
  - Examples: Exception handling, self-monitoring, undo history.

Types of Transformations

From Biggest to Smallest:

- Impose Architectural Style
  - Reorganise the entire architecture (of the entire system *or* a specific sub-system)
  - Examples: Pipes-and-Filters, Layered, Model-View-Controller, Blackboard, etc.
- Impose Architectural Pattern
  - Maintain the components in the architecture and their functionality, but change their behaviour.
  - Examples: Introduce Concurrency, Persistence, Distributed Communication Mechanisms
- Apply Design Pattern
  - Reorganises a single component, or a small set of components according to the principles of a design pattern.
  - Examples: Facade, Observer, Abstract Factory, Strategy.
- Convert QR to Functionality
  - Add functionality, not primarily from the application domain.
  - Examples: Exception handling, self-monitoring, undo history.

Types of Transformations

From Biggest to Smallest:

- Impose Architectural Style
  - Reorganise the entire architecture (of the entire system *or* a specific sub-system)
  - Examples: Pipes-and-Filters, Layered, Model-View-Controller, Blackboard, etc.
- Impose Architectural Pattern
  - Maintain the components in the architecture and their functionality, but change their behaviour.
  - Examples: Introduce Concurrency, Persistence, Distributed Communication Mechanisms
- Apply Design Pattern
  - Reorganises a single component, or a small set of components according to the principles of a design pattern.
  - Examples: Facade, Observer, Abstract Factory, Strategy.
- Convert QR to Functionality
  - Add functionality, not primarily from the application domain.
  - Examples: Exception handling, self-monitoring, undo history.

Summary

- There are different times and different purposes for doing architecture evaluation
- The purpose and the time decides which method to use.
- Some of the more well-known are SAAM and ATAM for scenario-based evaluations.
- Researchers from BTH have also made some humble contributions in the area, including QASAR and Experiences from the BTH 4-hour evaluation method.
- After the evaluation, the architecture should be transformed to address the found issues. Bosch 2000 introduces four ways of transforming the architecture.
- There are other ways of evaluating (and describing) architectures. Recent endeavours include AADL, which is the focus of the next lecture.