in real life

Quality Attributes and Architecture Decisions

L02 PA1308

Mikael Svahnberg[1]

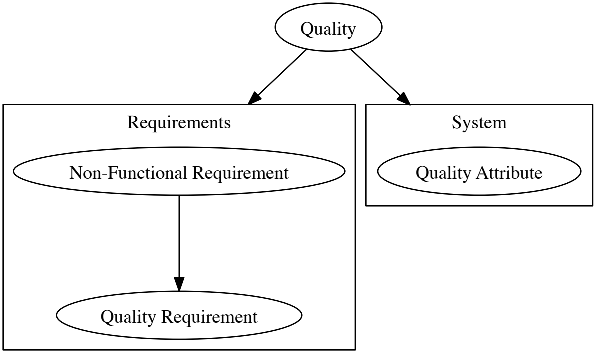
[1] Mikael.Svahnberg?bth.se
School of Computing
Blekinge Institute of Technology

September 10, 2012

Quality

- What is quality?

Terms


Quality
Requirements
Non-Functional Requirement
Quality Requirement
System
Quality Attribute

Requirement vs Attribute

- A quality attribute is always present
- A quality requirement puts a constraint on an attribute
- A quality requirement describes a service level of the system (or, more likely, a functional requirement).

Architecture and Quality Attributes

- Functionality is “easy” to implement.
- Quality requirements may *sometimes* have impact on the implementation
- More often, it impacts the *software structure* (=the software architecture).
- . . . And yet, the architecture can only describe a *potential* for achieving a particular quality level.

in real life

Examples

- Usability
  - ¬Button layout etc.
  - Certain functions (e.g. undo, data re-use).
- Modifiability
  - How is functionality divided?
  - . . . In relation to likely change scenarios.
- Performance
  - communication between components
  - division of functionality between components
  - allocation of shared resources
  - ¬choice of algorithms

Examples

- Usability
  - ¬Button layout etc.
  - Certain functions (e.g. undo, data re-use).
- Modifiability
  - How is functionality divided?
  - . . . In relation to likely change scenarios.
- Performance
  - communication between components
  - division of functionality between components
  - allocation of shared resources
  - ¬choice of algorithms

Examples

- Usability
  - ¬Button layout etc.
  - Certain functions (e.g. undo, data re-use).
- Modifiability
  - How is functionality divided?
  - . . . In relation to likely change scenarios.
- Performance
  - communication between components
  - division of functionality between components
  - allocation of shared resources
  - ¬choice of algorithms

BLEKINGE TEKNISKA HÖGSKOLA · BTH · in real life

Examples

- Usability
  - ¬Button layout etc.
  - Certain functions (e.g. undo, data re-use).
- Modifiability
  - How is functionality divided?
  - . . . In relation to likely change scenarios.
- Performance
  - communication between components
  - division of functionality between components
  - allocation of shared resources
  - ¬choice of algorithms

Examples

- Usability
  - ¬Button layout etc.
  - Certain functions (e.g. undo, data re-use).
- Modifiability
  - How is functionality divided?
  - . . . In relation to likely change scenarios.
- Performance
  - communication between components
  - division of functionality between components
  - allocation of shared resources
  - ¬choice of algorithms

Examples

- Usability
  - ¬Button layout etc.
  - Certain functions (e.g. undo, data re-use).
- Modifiability
  - How is functionality divided?
  - . . . In relation to likely change scenarios.
- Performance
  - communication between components
  - division of functionality between components
  - allocation of shared resources
  - ¬choice of algorithms

Examples

- Usability
  - ¬Button layout etc.
  - Certain functions (e.g. undo, data re-use).
- Modifiability
  - How is functionality divided?
  - . . . In relation to likely change scenarios.
- Performance
  - communication between components
  - division of functionality between components
  - allocation of shared resources
  - ¬choice of algorithms

Examples

- Usability
  - ¬Button layout etc.
  - Certain functions (e.g. undo, data re-use).
- Modifiability
  - How is functionality divided?
  - . . . In relation to likely change scenarios.
- Performance
  - communication between components
  - division of functionality between components
  - allocation of shared resources
  - ¬choice of algorithms

Examples

- Usability
  - ¬Button layout etc.
  - Certain functions (e.g. undo, data re-use).
- Modifiability
  - How is functionality divided?
  - . . . In relation to likely change scenarios.
- Performance
  - communication between components
  - division of functionality between components
  - allocation of shared resources
  - ¬choice of algorithms

Examples

- Usability
  - ¬Button layout etc.
  - Certain functions (e.g. undo, data re-use).
- Modifiability
  - How is functionality divided?
  - . . . In relation to likely change scenarios.
- Performance
  - communication between components
  - division of functionality between components
  - allocation of shared resources
  - ¬choice of algorithms

Examples

- Usability
  - ¬Button layout etc.
  - Certain functions (e.g. undo, data re-use).
- Modifiability
  - How is functionality divided?
  - . . . In relation to likely change scenarios.
- Performance
  - communication between components
  - division of functionality between components
  - allocation of shared resources
  - ¬choice of algorithms

Levels of Quality Attributes

- Business Qualities[1]
  - Time-to-Market, Cost and Benefit, Projected Lifetime, Targeted market, Rollout schedule, Legacy system integration
  - Also: Product portfolio, Requirements from Society, etc.[2]
- System Quality Attributes
  - Availability, Modifiability, Performance, Security, Testability, Usability
  - ISO 9126: Functionality, Reliability, Usability, Efficiency, Maintainability, Portability
  - There are many more classifications. See e.g. Svahnberg & Henningsson 2009[3]
- Architecture Qualities
  - Conceptual Integrity, Correctness and Completeness, Buildability

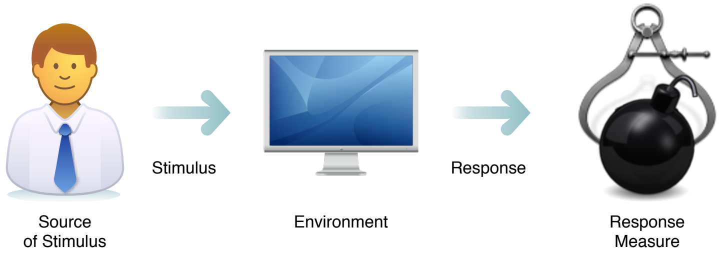
[1] Observe that I use a different order than Bass et al.

[2] T. Gorschek and A. M. Davis. Requirements engineering: In search of the dependent variables. *Information and Software Technology*, 50(1-2):67-75, 2008.

[3] M. Svahnberg and K. Henningsson. Consolidating different views of quality attribute relationships. In *7th Workshop on Software Quality, proceedings of the International Conference on Software Engineering (ICSE2009)*, pages 46-50, Los Alamitos CA, may 2009. IEEE Computer Society Press.

Achieving Quality Attributes (I)

- In order to achieve a a certain level for a quality attribute we need a *controlled process* to lead us towards an architecture *decision*.
- Quality Attribute Scenarios is one building block for this:



Example of Quality Attribute Scenario: Performance

Source: Users

Stimulus: Initiate Transactions

Environment: Under normal operations

Response: Transactions are processed

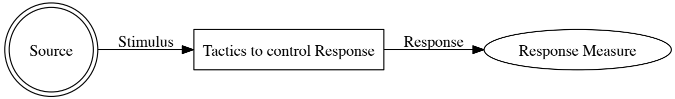
Response Measure: With an average latency of two seconds

Aspect	Value
Source	Users
Stimulus	Initiate transactions: 1000 per minute
Artifact	System
Environment	Normal mode (c.f. overload mode)
Response	Transactions are Processed
Response Measure	Latency of 2s (deadline, throughput, jitter, miss rate, data loss, etc)

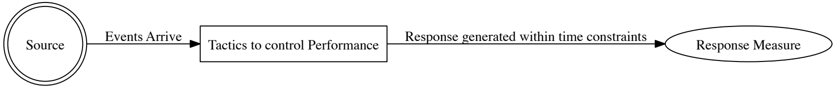
Achieving Quality Attributes (II)

- The next step is to find a solution to a Quality Attribute Scenario.
- To this, we have *tactics*.
- A tactic *can be* (but is not limited to) a design pattern, an architecture pattern, or an architectural strategy.

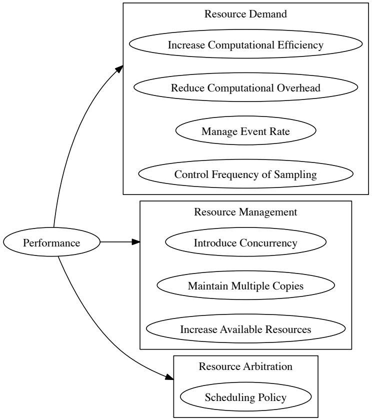
Generic:



Example (Performance):


Source
Events Arrive
Tactics to control Performance
Response generated within time constraints
Response Measure

Example: Summary of Performance Tactics



For many more and for other quality attributes, see Bass et al.

Other Concerns

- One may be led to believe that if only the quality requirements are taken care of, the rest will follow.
- Obviously, this is not the case.
- Hofmeister et al.[4] lists three sources of concerns[5]:
  - Organisational factors:
    - Management (cf. business qualities above)
    - Staff skills, interests, strenghts, weaknesses
    - Process and development environment
    - Development schedule
    - Development Budget
  - Technological factors
  - Product factors

[4] C. Hofmeister, R. Nord, and D. Soni. *Applied Software Architecture*. Addison-Wesley, Reading MA, 2000.

[5] Please note the overlap to the categories from Bass et al.

Other Concerns

- One may be led to believe that if only the quality requirements are taken care of, the rest will follow.
- Obviously, this is not the case.
- Hofmeister et al.[4] lists three sources of concerns[5]:
    - Organisational factors:
    - Technological factors
        - General-purpose hardware
        - Domain-specific hardware
        - Software technology
        - Architecture technolog
        - Standards
    - Product factors

[4] C. Hofmeister, R. Nord, and D. Soni. *Applied Software Architecture*. Addison-Wesley, Reading MA, 2000.

[5] Please note the overlap to the categories from Bass et al.

Other Concerns

- One may be led to believe that if only the quality requirements are taken care of, the rest will follow.
- Obviously, this is not the case.
- Hofmeister et al.[4] lists three sources of concerns[5]:
  - Organisational factors:
  - Technological factors
  - Product factors
    - Functional Features
    - User Interface
    - Performance
    - Dependability
    - Failure detection, reporting, recovery
    - Service
    - Product cost

[4] C. Hofmeister, R. Nord, and D. Soni. *Applied Software Architecture*. Addison-Wesley, Reading MA, 2000.

[5] Please note the overlap to the categories from Bass et al.

Working with Factor Tables (I)

- Identify factors
- Analyse changeability and flexibility
- Analyse impact of the factors

Factor	Flexibility and Changeability	Impact
O1: <Factor category>		
O1.1: <Factor name>		
<description of factor>	<What aspects of the factor are flexible or changeable>	<Components affected by the factor or changes to it>
. . .		

Working with Factor Tables (I)

- Identify factors
- Analyse changeability and flexibility
- Analyse impact of the factors

Factor	Flexibility and Changeability	Impact
O1: <Factor category>		
O1.1: <Factor name>		
<description of factor>	<What aspects of the factor are flexible or changeable>	<Components affected by the factor or changes to it>
. . .		

Identify Factors

Factors that have a “global influence”, could change during development, are difficult to satisfy, or where you have little experience:

- Can the factor’s influence be localised to one component, or is it distributed across several components?
- During which stages of development is the factor important?
- Does the factor require any new expertise or skills?

Working with Factor Tables (I)

- Identify factors
- Analyse changeability and flexibility
- Analyse impact of the factors

Factor	Flexibility and Changeability	Impact
O1: <Factor category>		
O1.1: <Factor name>		
<description of factor>	<What aspects of the factor are flexible or changeable>	<Components affected by the factor or changes to it>
. . .		

Analyse Changeability and Flexibility

Flexibility: What is negotiable about the factor?

- Is it possible to influence the factor so that it makes architecture development easier? In what way? To what extent?

Changeability: What is likely to change in the factor?

- In what way could the factor change? How likely and how often will it change? Will the factor be affected by changes in other factors?

Working with Factor Tables (I)

- Identify factors
- Analyse changeability and flexibility
- Analyse impact of the factors

Factor	Flexibility and Changeability	Impact
O1: <Factor category>		
O1.1: <Factor name>		
<description of factor>	<What aspects of the factor are flexible or changeable>	<Components affected by the factor or changes to it>
. . .		

Analyse Impact of Factors

If the factor changes, what else will have to change:

- Other factors?
- Components?
- Design Decisions?
- Modes of operation in the system?

Working with Factor Tables (II)

- Once you have identified the relevant factors that will influence your architecture, you need to decide how to address them.
- The factors are probably too many to maintain a decent overview, so we need to simplify.
- Factors and their changeability are used to identify *Issues*. An issue may arise from:
  - limitations or constraints imposed by factors. For example, agressive development schedule vs requirements overload.
  - the need to reduce the impact of changeability of factors. For example, design the architecture to reduce the cost to porting to other operating systems.
  - difficulties in satisfying product factors. For example, high throughput requirements may overload the CPU.
  - the need to have a common solution to global requirements. For example, error handling and recovery.
- Several factors may affect an issue. Factors may conflict with each other $\rightarrow$ renegotiations.

Factors and Issues, what next?

- Develop *Solutions* and *Specific Strategies*
- These are your documented, motivated, and analysed, *architectural decisions* to solve the issues.
- Your strategies should address the following goals:
  - Reduce or localise the factor's influence.
  - Reduce the impact of the factor's changeability on the design and other factors.
  - Reduce or localise required areas of expertise or skills.
  - Reduce overall time and effort.
- Once this is done, identify strategies that are related to each other, link them, and ensure that they can work together.
- So much for Hofmeister et al. I would also add (and require in the assignments):
  - Address the issue with a *software* solution, wherever possible.

Factors and Issues, what next?

- Develop *Solutions* and *Specific Strategies*
- These are your documented, motivated, and analysed, *architectural decisions* to solve the issues.
- Your strategies should address the following goals:
  - Reduce or localise the factor's influence.
  - Reduce the impact of the factor's changeability on the design and other factors.
  - Reduce or localise required areas of expertise or skills.
  - Reduce overall time and effort.
- Once this is done, identify strategies that are related to each other, link them, and ensure that they can work together.
- So much for Hofmeister et al. I would also add (and require in the assignments):
  - Address the issue with a *software* solution, wherever possible.

Issue Card

<Name of the architecture design issue>
<Description of the issue> **Influencing Factors** <List of the factors that affect this design issue and how>
Solution <Discussion of a general solution to the design issue, followed by a list of the associated strategies> **Strategy:** <Explanation of the strategy>
Related Strategies <References to related strategies and a discussion of how they are related to this design issue>

Issue Card Example

Issue i1: ***Automatically Adapt to Input Format***
The KWIC system can be fed input in many formats, such as xls, doc, rtf, tex, pdf, or txt. The system shall automatically detect the input format (without access to the file name), and process it accordingly. New input formats may be added at runtime. **Influencing Factors** P6.4 Adapt to input format P6.32 Support Runtime Updates
Solution **Strategy: ?**
Related Strategies

Bass et al: Attribute-Driven Design

Bass et al. describes a different way of designing the architecture:

1. Choose the module to decompose
2. Refine the module according to these steps:
    1. Choose the architectural drivers from quality scenarios and functional requirements
    2. Choose an architectural pattern that satisfies the architectural drivers
    3. Instantiate modules and allocate functionality
    4. Define interfaces of child modules
    5. Verify and refine use cases and quality scenarios and make them constraints for the child modules.
3. Repeat for every module that needs further decomposition.

Benefits of each

- Hofmeister et al. presents a decision rich methodology, with emphasis on traceability and documentation.
- Bass et al. comes to a point where you can start designing (making architecture decisions) sooner.
- Hofmeister et al. takes a wider perspective; Not only can the factors influence the architecture, but they can also influence each other and, as we will see, the architecture can also influence the factors.
- Bass et al. seems to assume a high-level design of the system beforehand (I may be wrong on this, though).
- Ultimately, they both land on the same point: *You have to decide how to address your issues or architectural drivers.*

Software Solutions

- In a course (or any hypothetical system that is never going to be built), it is often easy to solve issues simply by allowing more or better hardware.
- In industry, the hardware constraints are *real* and *hard*.
- For example (using low estimates):
  - Require 1GB more internal memory in the computer = 17 Euro.
  - Ship 1000 units/year = 17 000 Euro.
  - Expected lifespan of system: 10 years = 170 000 Euro.
  - Ensure availability of the right memory modules for the hardware platform for the coming 10 years: lots more.
- Contrast this with one well payed swedish developer working full-time for one year to reduce the memory consumption in software: 75000 Euro (including tax and social fees).

Whence Strategies?

- Architectyre Styles and Patterns
- Design Patterns
- Tactics
- Creativity
- Experience

Whence Strategies?

- Architectyre Styles and Patterns
- Design Patterns
- Tactics
- Creativity
- Experience

Issue Cards and Tactics

- An issue card can be viewed as a quality attribute scenario. Example:

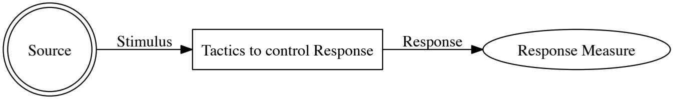
  The KWIC system can be fed input in many formats, such as xls, doc, rtf, tex, pdf, or txt. The system shall automatically detect the input format (without access to the file name), and process it accordingly. New input formats may be added at runtime.

  Influencing Factors

  P6.4 Adapt to input format

  P6.32 Support Runtime Updates

- Source: User or command shell.
- Stimulus: Any of xls, doc, rtf, tex, pdf, or txt.
- Environment: Runtime Updates, Normal Operations
- Response: List of KWIC
- Response Measure: Success or Fail to generate KWIC



Issue Cards and Tactics

- An issue card can be viewed as a quality attribute scenario. Example:

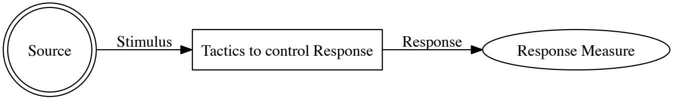
The KWIC system can be fed input in many formats, such as xls, doc, rtf, tex, pdf, or txt. The system shall automatically detect the input format (without access to the file name), and process it accordingly. New input formats may be added at runtime.

Influencing Factors

P6.4 Adapt to input format

P6.32 Support Runtime Updates

- Source: User or command shell.
- Stimulus: Any of xls, doc, rtf, tex, pdf, or txt.
- Environment: Runtime Updates, Normal Operations
- Response: List of KWIC
- Response Measure: Success or Fail to generate KWIC



Issue Cards and Tactics

- An issue card can be viewed as a quality attribute scenario. Example:

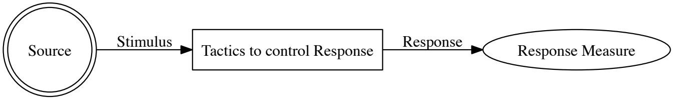
The KWIC system can be fed input in many formats, such as xls, doc, rtf, tex, pdf, or txt. The system shall automatically detect the input format (without access to the file name), and process it accordingly. New input formats may be added at runtime.

Influencing Factors

P6.4 Adapt to input format

P6.32 Support Runtime Updates

- Source: User or command shell.
- Stimulus: Any of xls, doc, rtf, tex, pdf, or txt.
- Environment: Runtime Updates, Normal Operations
- Response: List of KWIC
- Response Measure: Success or Fail to generate KWIC



Issue Cards and Tactics

- An issue card can be viewed as a quality attribute scenario. Example:

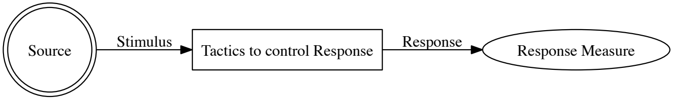
The KWIC system can be fed input in many formats, such as xls, doc, rtf, tex, pdf, or txt. The system shall automatically detect the input format (without access to the file name), and process it accordingly. New input formats may be added at runtime.

Influencing Factors

P6.4 Adapt to input format

P6.32 Support Runtime Updates

- Source: User or command shell.
- Stimulus: Any of xls, doc, rtf, tex, pdf, or txt.
- Environment: Runtime Updates, Normal Operations
- Response: List of KWIC
- Response Measure: Success or Fail to generate KWIC

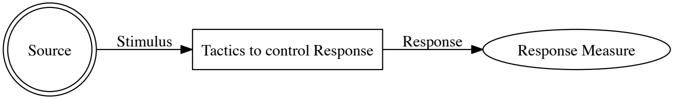


Issue Cards and Tactics

- An issue card can be viewed as a quality attribute scenario. Example:

> The KWIC system can be fed input in many formats, such as xls, doc, rtf, tex, pdf, or txt. The system shall automatically detect the input format (without access to the file name), and process it accordingly. New input formats may be added at runtime.
>
> **Influencing Factors**
>
> P6.4 Adapt to input format
>
> P6.32 Support Runtime Updates

- Source: User or command shell.
- Stimulus: Any of xls, doc, rtf, tex, pdf, or txt.
- Environment: Runtime Updates, Normal Operations
- Response: List of KWIC
- Response Measure: Success or Fail to generate KWIC



Summary

- The success of a software system is determined not only by its functionality, but even more so by its *quality*
  - Quality is a very wide concept, that has different meanings to different stakeholders.
  - Quality attributes are the sub-aspects of quality that a system exhibits.
  - The potential to meet many quality attributes is achieved through the software architecture.
  - Thus, architecture design methodologies often focus on how to address all the (potentially conflicting) quality requirements.
  - This creates a link: Quality Requirement $\rightarrow$ Design Decision $\rightarrow$ Architecture Design $\rightarrow$ Implementation
  - *However*, there are more concerns than quality requirements, cf. Hofmeister et al.
- The first rule of architecture design is to *document your decisions*
  - Hofmeister et al. provides one way to do this, through Factors, Issues, and Strategies. Factors and Issues serve as a motivation for your strategies (decisions).
  - Bass et al., and their ADD method, provides another way to do this, but leaves much more of the decision rationale open.
- Knowing how to solve a problem in the architecture is not trivial. You can base yourself on experience or creativity, but there are proven solutions available for you.
  - Architecture styles, patterns, and *tactics* are available for you to solve your architectural issues.
  - A tactic is an established solution for addressing a particular quality attribute.
- $\boxed{\text{Next Step: Documenting your Architecture, Views and Beyond.}}$