in real life

Quality Attributes

Mikael Svahnberg[1]

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

April 26, 2017

Quality

- What is quality?

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.

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

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

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

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

in real life

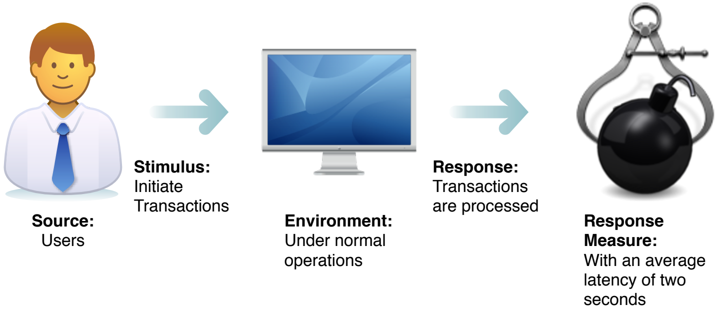
Levels of Quality Attributes

- Business Qualities
  - Time-to-Market, Cost and Benefit, Projected Lifetime, Targeted market, Rollout schedule, Legacy system integration
  - Also: Product portfolio, Requirements from Society, etc.[1]
- System Quality Attributes
  - Availability, Modifiability, Performance, Security, Testability, Usability
  - ISO 9126: Functionality, Reliability, Usability, Efficiency, Maintainability, Portability
- Architecture Qualities
  - Conceptual Integrity, Correctness and Completeness, Buildability

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

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

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.

in real life

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.[2] lists three sources of concerns[3]:
  - Organisational factors:
    - Management (cf. business qualities above)
    - Staff skills, interests, strenghts, weaknesses
    - Process and development environment
    - Development schedule
    - Development Budget
  - Technological factors
  - Product factors

---

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

[3] 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.[2] lists three sources of concerns[3]:
  - Organisational factors:
  - Technological factors
    - General-purpose hardware
    - Domain-specific hardware
    - Software technology
    - Architecture technolog
    - Standards
  - Product factors

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

[3] 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.[2] lists three sources of concerns[3]:
    - Organisational factors:
    - Technological factors
    - Product factors
        - Functional Features
        - User Interface
        - Performance
        - Dependability
        - Failure detection, reporting, recovery
        - Service
        - Product cost

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

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

in real life

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).

Architectures for different purposes

- Regular desktop applications
- Embedded applications
- Enterprise applications
- Applications for Android / IOS
- Cloud applications
- Service-Oriented Architectures (SOA)
- Software Ecosystems
- ...

in real life

What can differ?

- Instantiation into different viewpoints?
  - Trivial, the views will differ between each application anyway.
- Factors, Issues and Strategies?
  - Also trivial, for the same reason.
- The *importance* of certain factors (e.g. the importance of certain quality requirements).
- *Typical choices* of strategies for a particular domain.
- Typical choices of architecture styles for a particular domain (may be subordinate to the aforementioned)

What can differ?

- Instantiation into different viewpoints?
  - Trivial, the views will differ between each application anyway.
- Factors, Issues and Strategies?
  - Also trivial, for the same reason.
- The *importance* of certain factors (e.g. the importance of certain quality requirements).
- *Typical choices* of strategies for a particular domain.
- Typical choices of architecture styles for a particular domain (may be subordinate to the aforementioned)

in real life

What can differ?

- Instantiation into different viewpoints?
  - Trivial, the views will differ between each application anyway.
- Factors, Issues and Strategies?
  - Also trivial, for the same reason.
- The *importance* of certain factors (e.g. the importance of certain quality requirements).
- *Typical choices* of strategies for a particular domain.
- Typical choices of architecture styles for a particular domain (may be subordinate to the aforementioned)

in real life

What can differ?

- Instantiation into different viewpoints?
  - Trivial, the views will differ between each application anyway.
- Factors, Issues and Strategies?
  - Also trivial, for the same reason.
- The *importance* of certain factors (e.g. the importance of certain quality requirements).
- *Typical choices* of strategies for a particular domain.
- Typical choices of architecture styles for a particular domain (may be subordinate to the aforementioned)

What can differ?

- Instantiation into different viewpoints?
  - Trivial, the views will differ between each application anyway.
- Factors, Issues and Strategies?
  - Also trivial, for the same reason.
- The *importance* of certain factors (e.g. the importance of certain quality requirements).
- *Typical choices* of strategies for a particular domain.
- Typical choices of architecture styles for a particular domain (may be subordinate to the aforementioned)

What can differ?

- Instantiation into different viewpoints?
  - Trivial, the views will differ between each application anyway.
- Factors, Issues and Strategies?
  - Also trivial, for the same reason.
- The *importance* of certain factors (e.g. the importance of certain quality requirements).
- *Typical choices* of strategies for a particular domain.
- Typical choices of architecture styles for a particular domain (may be subordinate to the aforementioned)

BLEKINGE TEKNISKA HÖGSKOLA · BTH ·
in real life

What can differ?

- Instantiation into different viewpoints?
  - Trivial, the views will differ between each application anyway.
- Factors, Issues and Strategies?
  - Also trivial, for the same reason.
- The *importance* of certain factors (e.g. the importance of certain quality requirements).
- *Typical choices* of strategies for a particular domain.
- Typical choices of architecture styles for a particular domain (may be subordinate to the aforementioned)

Embedded Applications I

This is a large and diverse field, and there are many quality requirements that may be in focus for different applications. However, there are some overall constraints that hold true for *most* applications in this domain:

- **Hardware cost**
  - Keep memory footprint low
  - Keep CPU usage low
  - Optimise for wear and tear
- **Hardware availability for entire product life expectancy**
  - Keep memory and CPU usage low.
  - Cull system regularly to remove stuff that is no longer needed.
  - Low-cost growth mechanisms.

Embedded Applications II

- **Testability**
  - Testable software – the usual stuff
  - Testable hardware – system test software, test harness, etc.
  - Report error states (e.g. through flashing diodes).
- **Reliability**
  - Error detection
  - Error recovery
  - Report error states visibly (e.g. on-line, flashing diodes)

Embedded Applications III

- **Energy consumption**
  - Low-effort computing
  - Reduce display time and display size (if any)
  - Powersave modes
  - Lazy evaluations – deferred processing
- **Network communications**
  - Standard communications platform
  - Robust transfers (e.g. in outdoor environments)
  - Operate by “dead reckoning”

Enterprise Applications I

- Enterprise architectures only has very little to do with software architecture – and yet it has *everything* to do with the software architecture.
- Organisational, Technological, and Product factors can be considered subsets, or limited views, of the enterprise architecture.

Enterprise Applications II: List of Concerns

- Persistent Data[4]
- Large amounts of data
- Large scale concurrent access
- Many data views (user interface screens)
- Need to integrate with other enterprise applications
- Multiple interpretations of data (conceptual dissonance)
- Complex business logic rules (business “illogic”)
- Various types of enterrpise systems
  $\{B, C\}2\{B, C\} \wedge B, C \in \{s, m, l, xl, xxl\}$

[4] M. J. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, Boston MA, 2003.

in real life

Enterprise Applications IV: Typical Architecture Styles

- Function-centric (Transaction script)
- Domain concept-centric (Domain model)
- Data representation-centric (Table module)

. . . I could go on, but the book (Fowler 2003) is rather thick and full of patterns that dig deeper and deeper into the application.

Android/IOS Applications

Somewhere between embedded and desktop applications. Any number of quality concerns may be relevant for any application. The platform itself imposes some concerns.

Hardware:

- Restrict battery usage
- Small screen
- Unorthodox input methods (e.g. thumb)
- Not the fastest CPU, restricted RAM.
- Variety of hardware available on a particular phone model.

Software:

- Interruptible applications (e.g. for phone calls)
- Interoperable applications
- Reuse wherever possible, yet enable customisation.
- Intuitive user experience that supports how users operate their handheld device (e.g., avoid deep meny hierarchies).
- Varity of services available on a particular phone.

A/IOS Applications: Example of Energy Needs

- *Jon Summers, University of Leeds* studied energy consumtion of *Gangnam Style*
- Downloaded $1.8 * 10^9$ times
- 4.13 minutes, 17MB
- Energy consumption for streaming it: 0.0002 kWh per minute.
- $\sum$ 312 GWh !
- That's more than 10 million Africans have for their whole nation (e.g. the whole country of Burundi) in a year!

A/IOS Applications: Example of Energy Needs

- *Jon Summers, University of Leeds* studied energy consumtion of *Gangnam Style*
- Downloaded $1.8 * 10^9$ times
- 4.13 minutes, 17MB
- Energy consumption for streaming it: 0.0002 kWh per minute.
- $\sum$ 312 GWh !
- That's more than 10 million Africans have for their whole nation (e.g. the whole country of Burundi) in a year!

A/IOS Applications: Example of Energy Needs

- *Jon Summers, University of Leeds* studied energy consumtion of *Gangnam Style*
- Downloaded $1.8 * 10^9$ times
- 4.13 minutes, 17MB
- Energy consumption for streaming it: 0.0002 kWh per minute.
- $\sum$ 312 GWh !
- That's more than 10 million Africans have for their whole nation (e.g. the whole country of Burundi) in a year!

A/IOS Applications: Addressing Concerns

- Battery usage
  - Event-driven applications (BroadcastReceivers and IntentFilters)
- Interruptible applications, “flat” user interfaces
  - Loosely connected applications
  - Event-driven applications
  - Application as a set of screens, each screen a separate process
  - Separate Activities (interaction-based) from Services (runs in the background)
  - Persistent storage as a system service
- interoperable applications, reuse wherever possible, variety of services available, yet customisable
  - Late binding
  - Loosely connected applications
  - Event-driven applications
  - Common communication platform: Intent and IntentFilters.
  - Separate Activities and Services from ContentProviders.

Cloud Applications

- The concept of a cloud application is simple: It is essentially a client-server solution, where rather than maintaining the server yourself, you rent virtual servers from a cloud vendor.
- *One* definition[5] of a cloud service
  - The service is accessible via a web browser or web services API
  - Zero capital expenditure is necessary to get started
  - You pay only for what you use as you use it.
- Another definition [6]
  - Pooled Resources, Virtualisation, Elasticity, Automation, Metered Billing

[5] G. Reese, *Cloud Application Architectures*, O'Reilly, 2009.
[6] Rosenberg et al., *The Cloud at your Service*, Manning Publications co., 2011.

in real life

Levels of Cloud Services

- Software as a Service (SaaS)
  - e.g. Google Docs, Yahoo!, SalesForce.com, Valtira, etc.
- Platform as a Service (PaaS)
  - e.g. Google App Engine, Microsoft Azure, etc.
- Infrastructure as a Service (IaaS)
  - e.g. Amazon Elastic Compute Cloud (EC2), Microsoft Azure, RackSpace, etc.
- . . .

Factors that “push” you towards the cloud

- Transference – Move your on-site solution as-is to the cloud for e.g. economic reasons.
  - Challenges: Setting up a similar environment in the cloud as you have locally.
- Internet Scale – Scaling up to handle more users.
  - Challenges: Database design may become a bottleneck.
- Burst Compute – Large swings in capacity requirements.
  - Challenges: Strategy for load balancing, database access.
- Elastic Storage – Scaling up to handle (much) more data.
  - Challenges: need also to consider where the data is processed.

Challenge: Internet Scale

Issue:

- Your database's working sets are too large
- Too many writes

Solution:

- Partition the data (Sharding)

in real life

Challenge: Cloudbursting

Issue:

- Occasional peaks of traffic that pushes infrastructure over its capacity

Solution:

- Use on-demand capacity (Cloud) for the peaks
- Load-balancer that divides work between in-house servers and cloud servers
- Render static data views

in real life

Summary

- Each application has its own set of unique challenges, but the *class* of applications may also have typical challenges and quality concerns
- These shape the solutions. Sometimes only a little, sometimes by dictating a certain architecture style.
- In this lecture a select few application classes have been introduced: Embedded, Mobile, Cloud, and Ecosystems.
- Embedded and Mobile application constraints are due to technical limitations.
- Cloud application constraints come from the users.