

Chapter

8



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 8: **Specifying Quality Attributes** **("Non-Functional Requirements")**

DTU course 02264

Abstract

- So far, we have dealt with functional requirements, but we have not covered non-functional requirements, more appropriately described as Required Quality Attributes (RQAs).
- They are by no means less important than functional and interface requirements, quite the opposite: if major RQAs are not addressed, an implementation is very likely to completely unusable, irrespective of how good the rest may be.
- Also, because RQAs are very often crosscutting concerns, it is much more difficult to retrofit an existing system with a new RQA than with (many) Features.

Contents

1. Issues of Software Qualities and RQAs
2. Select and Justify Qualities
3. Making Qualities Concrete and Measureable
4. Determining the right level of RQAs



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 8.1:

Issues of Software Qualities and Required Quality Attributes

DTU course 02264

Qualities of Software Systems

- **There are many different qualities of software systems.**
 - Quality is by itself a vague and subjective term. In the philosophy of mind, “Qualia” is a technical term referring to a subjective, conscious experience, such as pain, color perception, or the taste of wine.
 - Another account is “the ways things seem to us.” (D.C. Dennett).
- **Assume, we observe issues with the LMS and want to describe them in terms of the qualities that are missing from the system.**

Observation	Lacking Qualities
Users have problems specifying catalog searches.	Usability
Processing queries takes too long.	Performance
LMS sometimes crashes.	Availability Reliability
My catalog search queries become publicly accessible.	Integrity
My catalog search queries become publicly accessible.	Maintainability
My catalog search queries become publicly accessible.	Operatability

Examples of RQAs

- **In order to ask for qualities in a system under specification, we specify Required Quality Attributes (RQAs).**
 - RQAs are also known as non-functional requirements or “ilities”.
- **Typical (bad) examples for RQAs are**
 - “The system should be easy to use”,
 - “The system should be very fast”,
 - “The system should be highly reliable“, or
 - “The system should protect the privacy of its users”.
- **In comparison to features, RQAs are more difficult to handle.**
 - This includes elicitation, measurement, estimation, correction, and planning.
 - Adding insult to injury, we have less tools and theory to help us with RQAs,
 - Finally, many RQAs are rather technical in nature and difficult to communicate to a customer (e.g. maintainability, testability).
 - At the same time RQAs are often in conflict with each other, requiring complex trade-offs that involve a lot of experience and judgment.

RQAs are often critical

- **Features may be more or less important, but most RQAs are critical: if they are not met, the system may be completely useless, irrespective of the rest.**
 - Your airbag is really cheap, but inflates too late (performance, availability), not at all, or at a time when it ought not to inflate (reliability).
 - Your e-banking site has all the features in the world, but it takes 5min to load (performance), you can't operate it (usability), and exposes your password (privacy).
 - Your MP3 player looks really cool, but skips beats every few seconds (performance, reliability).
- **Usability is often just a nuisance under normal conditions, and may thus be mistaken for a second rate quality.**
 - However, when the human operators are challenged and stressed, or when the system's usual operating parameters are violated, a bad user interface provokes errors.
 - Thus, lack of usability may turn into a safety/security problem.
 - The Therac-25 incident was, among others, a usability problem
 - Destruction of the Iran Air Flight 655 in 1988 has been attributed to poor usability of the Aegis electronic warfare system.
 - The Warsaw incident, although officially qualified as human error, might have been prevented by a better user interface.
- **Many integrity requirements derive directly from legislation. Violating them may amount to negligence or worse.**

RQAs are difficult to validate

- **For many qualities, 99% is not good enough.**
 - Thus, testing is not effective: *“tests can only demonstrate the presence of bugs, not their absence”* (Dijkstra).
 - Even if we can test in theory, it may not be practical. Testing makes no sense for requirements like “The system fails only once in 10 years”.
 - Formal verification techniques (e.g., theorem proving, model checking) do apply, but are often not practically or economically viable.
 - If we manage to turn today’s UML models into formal entities suitable for verification, we get specifications for free (almost): formal verification may be closer than we think.
- **Other qualities may be validated by testing.**
 - This requires substantial effort in elaborating the requirements accordingly.
 - Usability is typically tested empirically, which is difficult and very expensive.
 - Also, there is margin for error: Unexpected usage conditions may be outside the tested conditions so our results are meaningless there.

RQAs are often difficult to retrofit

- **Most features are relatively easy to add, remove, or change, even after the construction.**
 - If we miss out some product use cases in the implementation of the LMS, we can always add them.
 - Even if it's a very important feature like *"return books"*, it is usually no problem to provide a work-around and add this feature later.
- **The reason for that is that most functionalities are isolated and self-contained.**
 - Notable exceptions to this rule include undo/redo, logging, or business rules and policies, such as *"All lending activities shall be recorded"*.
- **Many RQAs, on the other hand, are not implemented as isolated pieces of code, but by design-decisions, technological choices, or elements of the software architecture.**
 - Thus, RQAs are very often tightly intertwined with many other requirements ("crosscutting").
- **For example, increasing the level of availability is difficult, and often impossible.**
- **Similarly, if a system has not been built for testability from the beginning, it is very hard to add this quality afterwards.**
 - It is difficult to understand complex systems, and systems that do not have adequate test coverage usually also lack documentation.
 - Developers don't know whether something is like it is on purpose by accident.
 - Also, some test frameworks/tool impose certain restrictions on the software to be tested. Examples include disallowed language elements, limited choice of testable middleware, demands on certain patterns or architectures, ...

RQAs are difficult to specify (1)

- **RQAs must be clear and precise, without providing too much detail.**
 - It is very tempting to specify a solution rather than only requirement.
 - This might exclude better solutions we can't think of at this moment.
- **Example**
 - Many technology-minded people are tempted to specify the following.
R1a: LMS Users are identified by a unique login name and a secret password.
R1b: Every LMS user possess a machine readable card by which the users are uniquely identified to the system using card readers attached to all terminals.
 - Both of these requirements demand specific features rather than asking for a RQA. The underlying RQA that both **R1.a** and **R1.b** implement may be this.
R1: The LMS.Lending subsystem may only be used by authorized persons.

RQAs are difficult to specify (2)

- **However, R1: “The LMS.Lending subsystem may only be used by authorized persons” is not detailed enough for a contract or to be checked against a solution.**
 - Our supplier may implement R1 by restricting physical access, or by using special equipment (HW-tokens, smartcards).
 - Also, R1 allows for an implementation that grants all rights to all authorized users.
 - And what about authorizing other machines that use the LMS-Lending subsystem online?
- **Refining R1 to R1.1...R1.4 provides more details, and still allows different solutions (e.g., R1a and R1b).**
 - R1.1:** Only users equipped with the appropriate capability level may use the associated functions of the LMS.Lending subsystem.
Specifies which system has the access restrictions and indicates that there are different capability levels that vary with user.
 - R1.2:** The authentication solution must be easy to use by all readers.
Excludes all of the technologically advanced solutions,
 - R1.3:** Every LMS terminal is suitable to be used by any user.
Guarantees a minimum accessibility
 - R1.4:** Every LMS function is associated with a capability level.
Allows fine control, but requires some capabilities table.

RQAs are difficult to specify (3)

- **Specifying a solution may make follow-up requirements necessary.**
- **Example**
 - Assume that, after all, we do decide on passwords as the means of authentication (R1.a). This may ask for the following additional requirements.
 - R1.a1:** Every user must change the initial default password after the first login.
 - R1.a2:** The initial password may never be reused and is valid only one week.
 - R1.a3:** Passwords are stored in such a way, that they cannot be stolen.
 - If, conversely, we decide for a keycard as the means of authentication, we may have the following additional requirement.
 - R1.b1:** The keycard must be embossed in Braille letters
 - These requirements build on previous decisions, but again, they are open to the extent that we may still choose from a variety of solutions.
 - Name/reader no./CPR no. as ID, strength of password, Question/Reply, ...
 - RFID cards, bar code, magnetic strip, SmartCard, ...

Issues with Specifying RQAs

- Leaving the implementation and technological issues aside, there are mainly three issues in the specification of RQAs.
1. **Turning the subjective priorities and perceptions of qualities into an objective assessment that is accountable and justifiable.**
 - We do this by tying the RQAs to goals and stakeholders, similar to the way we justify Features.
 2. **Refining the imprecise notions of quality to operational metrics.**
 - We do this by following the Goal-Question-Metric (GQM) approach of breaking down the top level quality notions into more and more detailed attributes, that, eventually, can be measured.
 3. **Aiming for just the right level of these qualities.**
 - We will consider several approaches for this in the last section in this chapter.



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 8.2:

Select and Justify RQAs

DTU course 02264

Tailoring the set of RQAs

- **It is important to consciously decide on just the right set of RQAs.**
- **On the one hand, we must not miss critical RQAs, or specify too low levels for them.**
 - This may cripple the system and thus threaten the whole project.
 - Most systems rely strongly only on a small set of quality attributes.
- **On the other hand, trying to “be on the safe side” is no solution either.**
 - If we add too many or too stringent demands this doesn't necessarily help (consider the Ariane 501 case).
 - Also, quality is expensive (both in terms of money and time) and has to be justified by the business proposition.
 - Finally, adding too many constraints may reduce the software designer's too far.
- **Recall that we can't start with any set of RQAs and change it as we go along, we really must get this right first time.**

Tailoring the set of RQAs

Create a good (i.e., viable, justifiable) set of RQAs in these four steps.

1. Select RQA list

- Decide, as a project, on a catalog of pre-existing software quality attributes, e.g. the ISO 9126, the lists by Boehm or McCall, or a company specific catalog.

2. Remove irrelevant RQAs

- Go through your software quality attribute catalogues and tick off irrelevant RQAs, as determined by the system type and application area, the goals, and the stakeholder's opinion.

3. Shift emphasis and add new items

- Conversely, issues that are rather minor in any of the software quality attribute catalogues may be really important in the present project.
- Such issues should be emphasized, e.g. by turning them into a separate chapter in the requirements specification.

4. Establish benchmarks and targets

- For project management as well as for marketing it is important to know what level of quality a development is going for.

RQAs by System Type (1)

■ Performance

- In an embedded system, efficiency is usually very important, but for desktop systems often secondary.
- For system critical applications however, it is often a direct cost factor since twice the required computing power will mean twice the number of servers/data centers.

■ Integrity

- For administration system of the tax office, integrity is so important, that usually, there are no physical connections to the internet.
- For other systems, this may not be possible, such as a online retail system or the AMADEUS flight booking system.

■ Availability

- Many embedded systems need effectively 100% availability, where a failure means the replacement of the system.

RQAs by System Type (2)

Quality Attribute	Word	Magic Draw	ESP	AMADEUS	Compiler	Excel Macro	ERH	iPod
Reliability Re-startability Availability	--	-	++	+	--	--	+	-
Performance Capacity Endurance	-	+	++	+	+	--	+	++
Integrity Safety Security Privacy	-	--	--	++	--	--	++	--
Usability Understandability Learnability	++	+	--	--	-	-	-	++
Operability Interoperability Robustness	++	+	--	++	-	-	++	--
Maintainability Portability Testability	+	+	+	-	-	--	++	--

RQAs by Stakeholder

- **If you specify an RQA, make sure that you link it to the goals that it serves.**
 - Which qualities are relevant, how important they are, and to what degree they are essential depends on the individual perspective.
 - Use the stakeholder's point of view, and prioritize by their importance.

- **For instance, what are the qualities the stakeholders for the LMS will find most important to them?**

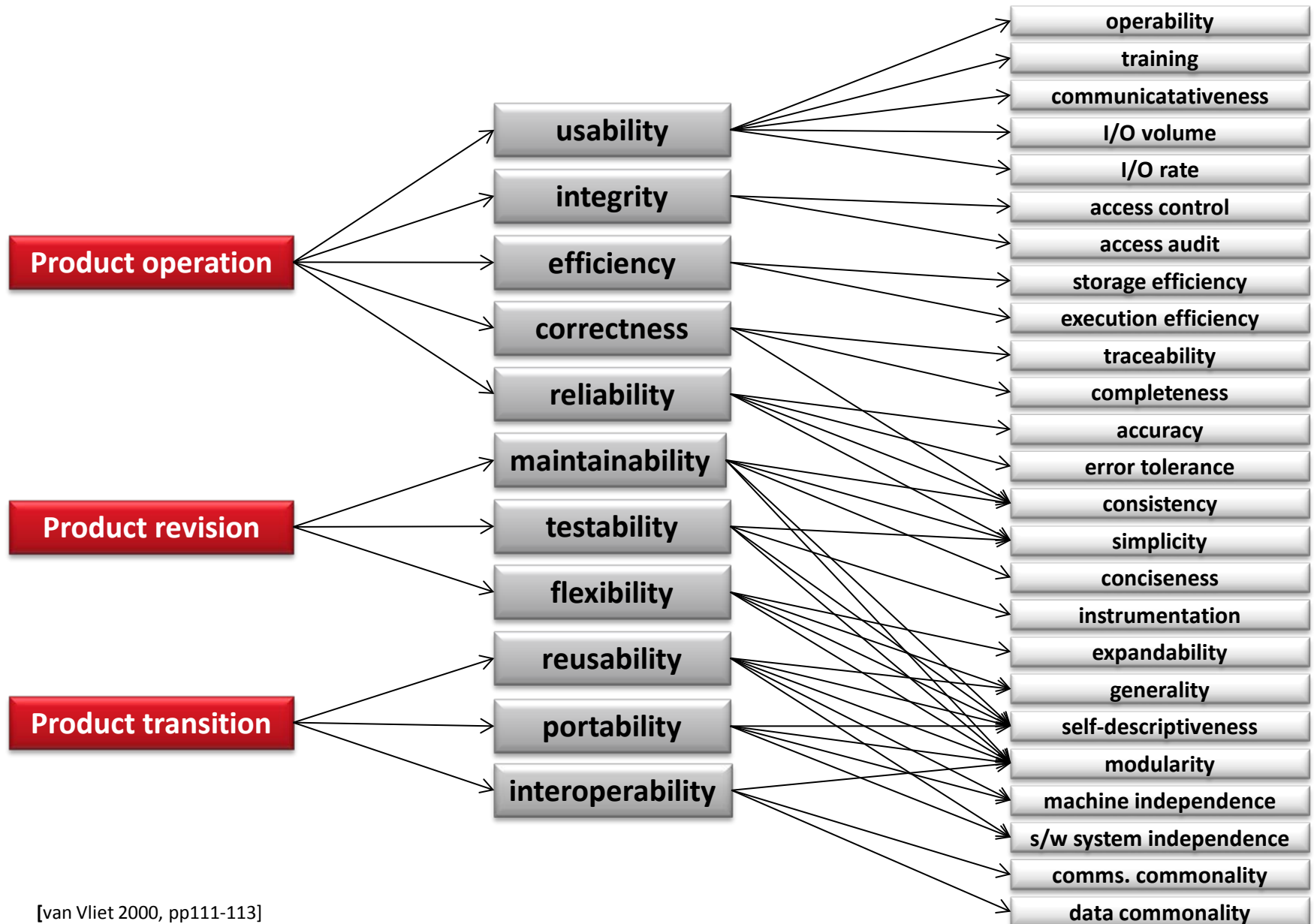
Stakeholders	Desired System Qualities
Readers	usability, privacy, performance
Librarians	usability, stability, reliability, availability, performance
Administrators	instalability, customizability, interoperability, performance
Developers (Initial)	testability
Developers (Maintenance)	portability, testability, well structuredness
Community Council	operatability, maintainability (→ cost!)

Dependencies among Quality Attributes

	Availability	Efficiency	Flexibility	Integrity	Interoperability	Maintainability	Portability	Reliability	Reusability	Robustness	Testability	Usability
Availability								+		+		
Efficiency			-		-	-	-	-		-	-	-
Flexibility		-		-		+	+	+			+	
Integrity		-			-				-		-	-
Interoperability		-	+	-			+					
Maintainability	+	-	+					+			+	
Portability		-	+		+	-			+		+	-
Reliability	+	-	+			+				+	+	+
Reusability		-	+	-	+	+	+	-			+	
Robustness	+	-						+				+
Testability	+	-	+			+		+				+
Usability		-								+	-	

Legend

- + increasing the row attribute improves the column attribute
- increasing the row attribute deteriorates the column attribute



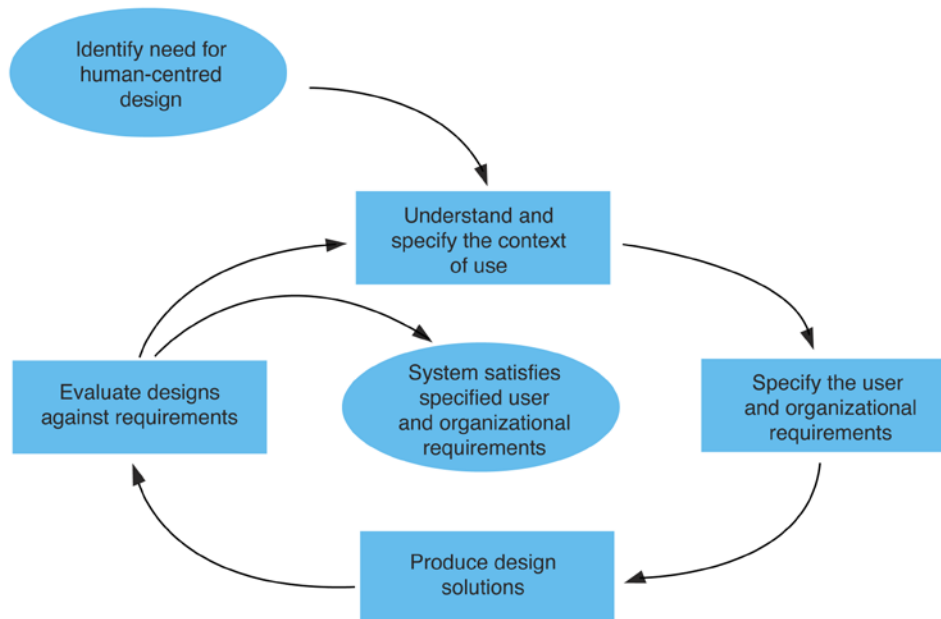
RQA-Sources (2):

Common Criteria

- **The common criteria recognition agreement is an international attempt to define and validate integrity requirements (see www.commoncriteriaportal.org).**
- **The common criteria are not restricted to software, they cover all kinds of IT products.**
 - The notion of IT product ranges from pure software products like data bases, operating systems, and middleware like Tivoli via integrated hardware/software systems like access control solutions and smart cards (e.g. ActivCard) to such unlikely “IT products” like German ID cards and passports.
 - The id card is merely a (optically) machine readable picture id.
 - The passport however also features a RFID transmitter and stores the passport data together with biometric features (fingerprints) in an encrypted format.
 - Processing and storing such ids is, of course an IT based process.
 - Similarly, the verification of such an ID is an automatic process.
- **The common criteria are the most widely accepted security requirements framework.**

RQA-Sources (3): Usability Standards

- **The two most important standards in the field are**
 - ISO 13407 Human centred design processes for interactive systems, and
 - ISO 9241 “Ergonomics of human-system interaction”.
- **They contain a wealth of information on usability.**
- **If usability really is an important quality of a system, usability experts are needed to specify usability requirements.**



Sw. Quality Attributes (ISO 29148:2011)

- **ISO 29148 defines systems requirements engineering processes for software products and services throughout the life cycle.**
 - It supersedes the earlier IEEE Std 830-1993, and relates to ISO/IEC 12207:2008 and 15288:2008.

Product requirements

- Functions
- Performance
- Interfaces
- Ressource Reqs.
- Security
- Quality
- Reliability

Process requirements

- Documentation
- Verification
- Acceptance

External requirements

- Maintainability
- Portability
- Safety
- Operational



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 8.3:

Making Qualities concrete and measurable

DTU course 02264

Refine RQAs

- **There are several approaches to refine Software Qualities that we can reuse for refining RQAs.**
 - Notable examples are the McCall Attribute list and the ISO 9126 standard (now ISO/IEC 250xx series) “Quality Attributes for Software Systems”.
- **In order to turn vague quality concepts into measureable quantities, we**
 - break the concepts down into a set of (more concrete) properties,
 - define a metric and specify an operational procedure to determine it, and
 - aggregating the measurements according to our property break down structure, yielding a quantitative value for the respective quality.

ISO 9126

Quality Attributes for Software Systems

Internal Quality

Portability

Adaptability
Installability
Co-Existence
Replaceability

Maintainability

Analysability
Changeability
Stability
Testability

External Quality

Usability

Understandability
Learnability
Operability
Attractiveness

Reliability

Maturity
Fault tolerance
Recoverability

Efficiency

Time behavior
Resource utilisation

Functionality

Suitability
Accuracy
Interoperability
Security

Operationalizing Metrics for RQAs

Quality Attribute Learnability

Quality Metric Effort necessary to learn to do a specific task with a specific level of efficiency

Measurement Procedure

- Specify Users in terms of capabilities (e.g. level of experience and qualification)
- Specify Tasks by goals, expected result, difficulty
- Specify Efficiency level in terms of effort (time, load), and quality (errors)

Quality Attribute Reliability

Quality Metric Mean time to failure for specific failure types and side conditions

Measurement Procedure

- Run program with a given set of constraints and stimuli, take time
- Count and classify issues (e.g. by analysing a log file)
- Compute MTTF

Quality Attribute Complexity

Quality Metric Coupling and binding of modules

Measurement Procedure

- Create call dependency graph of program
- Count call dependencies within and across module boundaries
- Compute size of modules
- Compute coupling and binding

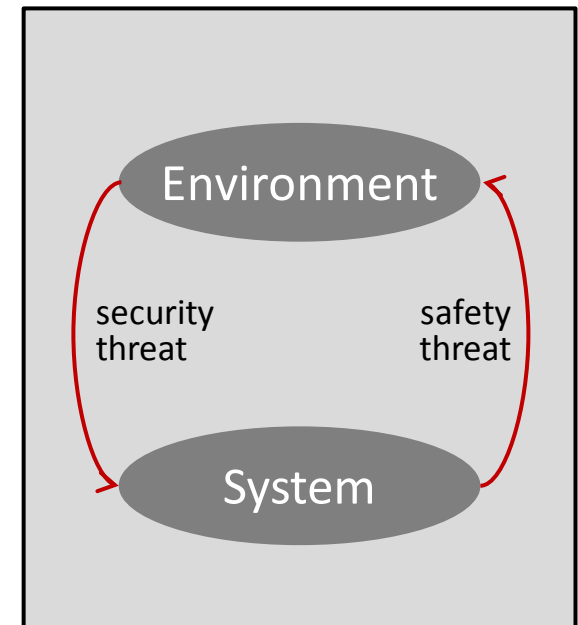
- **Usability is the ease of using a system accurately and efficiently, and learning to do so.**
 - Usability involves a great deal of psychological and cultural questions, which makes it inherently difficult to measure.
 - Also, it is still not always accepted as a first-class concern by many technologically-minded people.
- **Demanding that “The system must be easy to use” is not sufficient.**
 - R1:** In the learning phase, 90% of experienced users need no more than 2h to become so proficient with the system, that requirement 2 is satisfied.
 - R2:** During normal operation (i.e. after a learning phase, see R2), 90% of all users must be able to perform the following tasks:
 - R3:** A trained librarian shall be able to process the lending or returning of 10 books by 1 user in less than 5 seconds.
- **In order to avoid cluttering the requirements, we can use tables to specify the scenarios in greater detail.**

	task	amount	duration avg/max	coverage of cases [%]
front desk activity	accept returned book	1 book	3s / 5s	95
	accept returned book	10 books	20s / 30s	95
	accept returned book	10 media (any type)	25s / 35s	95
	prolong or return book	1 medium	10s / 20s	50
...

- **Another approach is to externalize requirements and use pre-existing standards.**
 - R4:** The application shall follow the company GUI style guide.

Integrity

- **Integrity comprises safety, security, and privacy.**
 - **Safety** is protection of the environment against damages by system malfunction.
 - **Security** is the protection of the system against tampering or outside attacks;
 - **Privacy** is the protection of individual data against disclosure to unauthorized parties.
- **These three aspects are complementary, and often governed by law.**



Integrity (Examples)

- R1:** The LMS.Lending subsystem may only be used by authorized persons. **SECURITY**
- R2:** Reader shall be accurately identified in all activities that could lead to cost on behalf of the reader. **SAFETY**
- R3:** Personal data shall be displayed to and changeable only by the reader. **PRIVACY**
- R4:** All user inputs are validated before being processed. **SAFETY**
- R5:** The system shall terminate any operation within 1s if the measured tank pressure exceeds 95% of the specified maximum pressure. **SAFETY**
- R6:** The radiation beam shield remain open only through continuous computer control. The shield shall automatically fall into place if computer control is lost for any reason. (→ Therac-25). **SAFETY**
- R7:** Patient records must only be exposed to medical staff with proper authorization. **PRIVACY & SECURITY**

Performance

- **Performance is the amount of work done with given resources.**
 - For example the client may wish to operate the new LMS without buying new machines, and so may demand *“The LMS shall be able to manage the current traffic of business processes and events with the currently installed hardware.”*
- **Performance may be broken down into Capacity and Endurance/Degradation.**
- **System capacity is often straightforward to specify: the numbers, frequencies, and sizes of domain objects, processes and so on are usually known.**
 - You should include required minimum numbers of users, objects of different kinds, amount of data and process instance per time, and so on.
 - Ideally, it should also include a forecast of the future development of these dimensions to achieve maintainability qualities (“scalability”).
 - When the cost or performance of the underlying hardware is critical (e.g. very large or embedded systems), it may also be necessary to specify the available resources in detail (i.e. processors and cores, clock time, available primary and secondary memory and so on).
- **Here are two examples for the LMS**
 - R1:** The LMS will have to handle at least 2,000 readers (3,000 within 3 years).
 - R2:** The LMS will have to handle at least 50,000 media (150,000 within 5 years).

- **Software reliability is the probability of executing without failure for a specific period of time in an environment in which it was intended to be used.**
 - Reliability may be decomposed into availability (ratio of up/down time), robustness (or “survivability”, handling of unexpected conditions), and restartability (how fast/easy we can come back).
- **As before, we need to differentiate between different types/severities of failures, and the services and service-levels offered.**
 - Maybe, we can accept lower service levels for some services and can accept some kinds of failures. What is sensible and cost-efficient for our system?
 - We need to be specific about the intended operation environment:
 - the working state *which services should be available?*
 - the operating hours, and *what days, what times?*
 - the operation conditions. *what other services are required?*
- **Examples**
 - “No more than 2 out of 100.000 successful scans of book or reader ids may result in a wrongly identified number.”
 - „The system must run 99.999% of the time during normal operation hours.“

Availability

- Availability is the degree to which a system is in a normal working state during expected operating hours, under expected operation conditions.

- Availability may be measured as averages of up-time versus down-time.

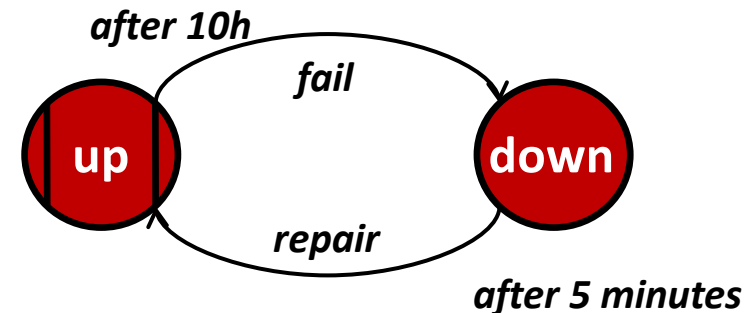
$$availability = \frac{MTTF}{MTTF + MTTR}$$

MTTF: mean time to failure (“up”)

MTTR: mean time to repair (“down”)

Example

- Consider a server which, on average, fails once every 10 hours. Restarting it takes 5 minutes.
- Its availability is $600/600+5 = 99.17\%$.



Availability	Down time
99.999%	ca. 5 minutes / year
99.99%	ca. 1h/year
99.5%	3:54 from 7:00-20:00 each day

Maintainability

- **Maintainability is the degree to which the software may be deployed, maintained, ported, and reused.**
- **By definitions, such activities take place in the future. Thus, it is difficult to select the right level of maintainability.**
 - As a rule of thumb, systems live ten times as long as one would imagine (→ Y2K).
 - Trying to prepare for every eventuality, however, is neither effective nor justifiable.
 - Planning for future system stages (cf. Domain architectures) can help.
- **As before, detail must be added to make maintainability operational.**
 - R1:** A maintenance programmer who has at least six months of experience supporting this product shall be able to develop a new report with at most as many output fields as report R123 including all modifications and test in less than 10 hours of work.
 - R2:** A programmer with at least 2 years of experience programming Java but no prior knowledge of the AID system shall be able to create a new AID Cartridge with three simple widgets and one new kind of transition event in less than 20 hours of work.

Installability

- **Installability is the ease with which a system deployment may be installed on a target platform.**
- **Software systems need varying degrees of installability.**
 - End user software systems must be installable by everybody
 - Many embedded systems are only "installed" at assembly
 - Large scale enterprise applications are only installed by trained specialists
- **Web applications (RIAs) have drastically reduced installability requirements for a great number of applications.**
- **Typical measures to ensure / increase installability of traditional applications include**
 - Release notes
 - Readme files
 - Installation scripts and wizards
 - Installation tutorials and manuals

Interoperability

- **Interoperability is the degree to which it is possible to exchange data or services with other systems.**
- **As before, we need to be specific about which systems and which data and services this refers to. Again, this may change in the future.**
- **Examples**
 - The MX system shall be able to read all files satisfying the XMI 2.1 DTD or earlier. The MX system shall be able to process all data satisfying the UML meta model version 2.2.0 or earlier.
 - The LMS shall be able to access all catalogs implementing the Common Danish Catalog Interface (CDCI) in the formulation of 1998.
 - The XYZ shall use the Windows System clipboard to send and receive data from other applications.
 - The XYZ shall integrate with the Tivoli system management.

Measuring/Testing High Quality

- **High levels of any quality are difficult to validate.**
 - Assume we demand an average downtime per year of less than 3s.
 - Ignoring the fact that this requirement is not very helpful to begin with, since 3s is too small a time frame to do, say, any maintenance or a restart.
 - The system must run for 10 years to be able to confirm by testing the requirement has been met with any degree of confidence. So, we can't really verify this requirement.
- **In this case, formal verification may be the only tool available.**
- **Similarly, consider the usability requirement “999 out of 1000 users without prior knowledge of the system must be able to carry out task x without help in less than y minutes”.**
 - In order to have a reliable reading, we have to test a very high number of subjects, or y has to be grossly over-dimensioned (e.g., 5 times as long as it usually takes).



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 8.4:

Determining the right level of RQA

DTU course 02264

“High Quality” can mean different things

- Depending on the system type, “high” quality can mean completely different things.
 - Even a seemingly precise requirement like “99.9999% (‘six sigma’) reliability” is not entirely clear. When considering a telephony switch system, this might mean the following.
 - At most 1 out of 10^6 phone calls may be lost.
 - The switch system is down no more than 5 minutes per year.
 - When considering a hospital patient monitoring system, we may intend something like this.
 - At most 1 out of 10^6 measurements may be lost.
 - The system may go down, as long as the ward is being notified.
 - The same is true for availability. For instance, for a web-shop or a business process support system we might specify:
 - “The system shall be at least 99.5% available Mon-Fri between 7:00 and 20:00 local time and at least 99.95% available on Mon-Fri between 9:00 and 17:00 local time.”
- whereas for a flight control system the specification might ask for:**
- “The system shall be at least 99.999% available Mon-Sun between 5:00 and 24:00 local time and at least 99.99% available at all other times.”

Quality Levels

- **Everybody wants the best quality, but nobody wants to pay for it, or has the time to wait.**
 - Thus, very high quality is usually only found in highly critical systems.
 - The essential ingredient in producing quality are not any tools or systems, but the software process as such; all the other factors follow from that.
- **For low or average levels of quality, it is sufficient to consider the large factors only (i.e. CMM/SPICE-level 1, aka. *any process at all*).**
 - In order to achieve very high quality, we need to cover not just the big factors, but also a large number of smaller factors.
 - This corresponds to a larger degree of rigidity or a higher CMM/SPICE level.
 - Interestingly, this is also a more cost-efficient process.
- **Agility in a project comes at a (high) cost, either in terms of effort, or in terms of not being able to *guarantee* properties (such as RQAs).**

RQAs derived by physical constraints

- Sometimes, it is possible to derive the right level RQA from other constraints, so all we have to do is some maths.
- Consider a traffic speed trap system that takes photos of car drivers exceeding the speed limits. There will be a RQA like this:
 - **R1:** The product shall detect a speed violation and take a picture of the offending driver within 0.5 seconds.
- Obviously, we need to perform this operation in a certain fixed amount of time.
 - The time will be determined by physical factors such as the distance to the camera, the detecting range, the absolute speed of the car and so on.
 - We just have to compute it.
- Of course, this is not always possible. For instance, what is the right x in R2?
 - **R2:** A newly acquired medium shall be available in the catalog within x minutes.

RQAs differ with project set-up

- **The project set-up may have a significant impact on the RQAs.**
- **Consider again R2.**
- **In-house development (Taarbæk Municipal IT Office develops LMS)**
 - MITO might not make an effort to achieve this quality, and we end up with 24h instead of 30min.
- **Several suppliers competing for a tender**
 - Going for a quicker response time might increase the chances of winning the contract.
 - On the other hand, a quicker response time might decrease other qualities or the functionality, or it might increase cost.
- **Customers nowadays have a tendency to ask for too much result for too little money.**
 - From a certain size on, tenders have to be offered publicly in Europe.
 - In Germany, there is a law requiring public customers to accept the cheapest offer, no matter what. → ALG-II disaster, BVK tender, ibiza tender

RQAs from clients

- **Clients are often unreliable as sources of RQAs for a number of reasons.**
 - Some clients want to cut costs and feel they need to pressure their supplier in order to not be cheated. If they are also less knowledgeable about software than we are (a common case), they might underestimate cost and difficulty.
 - Some clients tend to exaggerate their requirements because it underlines their own importance or they are afraid to take responsibility. Trying to “be on the safe side” is not a good solution.
 - (→ “you don’t get fired for buying IBM”)
- **One way to approximate the right quality level is to ask for the highest unacceptable level.**
 - For instance, when looking for acceptable response times we may ask *„what response time is unacceptable?“*. The first time they tend to accept is the right level.
 - Similarly, when asking for privacy concerning lending records, we may ask *„Is everyone allowed to read/write all lending records?“*
„Is everyone allowed to read/write their own lending records?“
And so on.

Reusing RQAs from Service Providers

- **In order to find good sets of RQAs, we might start at existing sets.**
 - Such sets may be available to our client through previous engagements, or from the system we are replacing (a very common case)
- **For instance, a good starting point for capacity requirements could be derived from an existing Service Level Agreement (SLA).**
 - Many companies provide such data on a quid-pro-quo basis as part of a benchmarking program.
 - Also, outsourcing providers have this kind of data and may be helpful in obtaining it.
- **Here is a small part of a SLA by Lufthansa Systems, provided as part of their service offering to hosts third party applications at their data centers.**

Dimension	Quantity	Unit
Maximum number of transactions	2,000,000	per month
Maximum number of transactions	125,000	per day
Records to be loaded	300	per minute
Maximum numbers of records in DB table	25,000	total
Maximum number of concurrent users	20	total

Reusing RQAs from Competitors

- **Another source for reusing RQAs are freely available industry best practices or publicized data.**

- Here is an example from the Gebühreneinzugszentrale (GEZ), the German Broadcasting Fees Authority.

Dimension	Quantity	Unit
Participant accounts	40,000,000	total
Transactions (average)	100,000 ... 140,000	Per day
Transactions (maximum)	2,000,000,000	Per day
Bookings	17,000,000	Per batch

[Computerzeitung 24, 12.6.2006,
p. 16, ##3300]

- **The problem is of course, that the replacement system will have completely different operating parameters so that the figures are not directly comparable.**

- For instance, the new system may keep track of the complete history of each case, it may provide a GUI where the old system only had a text interface, or it may have to offer completely different functionality like real time processing rather than batch processing, and so on (→ ERH).

The "Open Target/Metric" approach

- We might leave the target open, i.e. the following required quality.
 - R3:** The newly acquired medium shall be available in the general catalog within ___ minutes.
- This way, the supplier has some leeway, thus avoiding unjustified costs by unnecessary tight quality levels.
- The problem with this is again, that the supplier may not strive for the best possible deal for the customer.
- To limit this effect, the customer may additionally specify his expectations.
 - R4:** The newly acquired medium shall be available in the general catalog within ___ minutes (customer expects 15 minutes).
- This establishes a soft lower limit to the quality provided.

Quality Grid

The quality grid by Christiansen & Lauesen helps us consider only the relevant quality attributes, and assess their importance.

Here is the quality grid for a Hotel Reservation system.

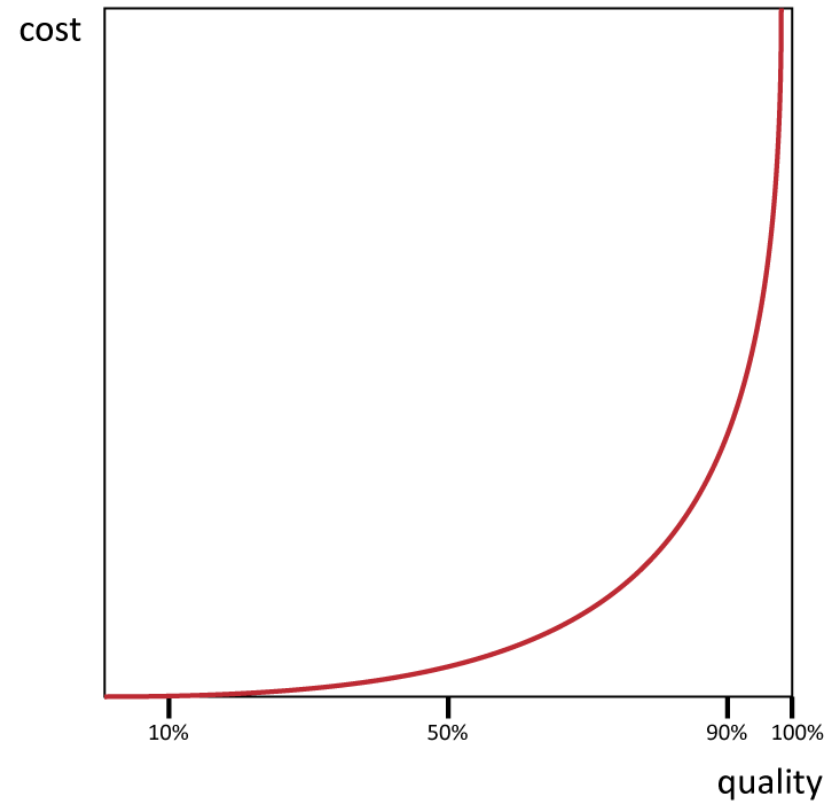
	Critical	Important	As usual	Unimportant	Ignore
Operation					
Integrity/security			X		
Correctness			X		
Reliability/availability		1			
Usability		2			
Efficiency			X		
Revision					
Maintainability			X		
Testability			X		
Flexibility			X		
Transition					
Portability					X
Interoperability	3			4	
Reusability					X
Installability		5			

Concerns:

1. Hard to run the hotel if system is down. Checking in guests is impossible since room status is not visible.
2. We aim at small hotels too. They have less qualified staff.
3. Customers have many kinds of account systems. They prioritize smooth integration with what they have.
4. Integration with spreadsheet, etc. unimportant. Built-in statistics suffice.
5. Must be much easier than present system. Staff in small hotels should ideally do it themselves.

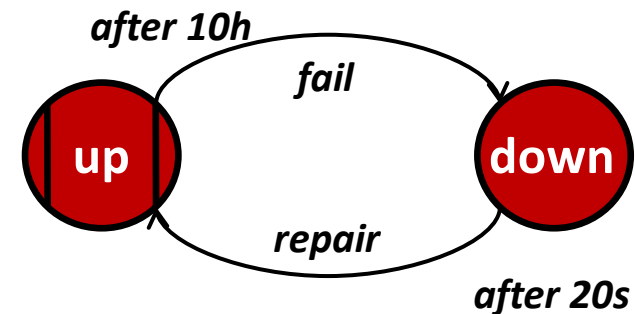
Arbitration among RQAs

- Ideally, we would like 100% quality, but that may be too expensive.
- However, we rarely really need 100%.
 - Often, 100% of one quality can be replaced by 90% of two (other) qualities.
 - This way, one RQA may be traded in for another, thus reducing the overall cost or complication.



Arbitrating among RQAs: Example 1

- Consider the qualities Availability and Restartability.
- When a system is not 100% available, it will fail sometime, and the system must be restarted.
 - In order to increase availability, we can either reduce the frequency of failures, or accelerate the repair process.
 - Assume you are aiming for 99,95% availability. This can be achieved in two ways:
 - a MTTF of 1000h and 30min of restart duration; or
 - a MTTF of 10h and 20s of restart duration.
- In the example, we might be able to isolate the subsystems and their services, and require different sets of RQAs for them, e.g., starting them separately before connecting them.



Arbitrating among RQAs: Example 2

- **It is usually very difficult to guarantee that capacity requirements are met.**
 - Adding a small security margin is ok, and also contribute to maintainability, but adding too much of a margin is not cost effective.
 - Also, similar to reliability, capacity considerations will affect the overall architecture (e.g. storage in file vs. data base, processing with/without application server, or single vs. multiple servers required).
- **Similarly, verifying capacity requirements may be difficult.**
 - Waiting till the case becomes reality is no good, obviously.
 - Producing fake data for a stress test is a lot of effort, and it may be difficult to make sure they have the same characteristics as real production data.
- **It may be easier to consider degradation, i.e. what happens, if the limits of the requirements are exceeded (“stress”).**
 - Suppose, the Taarbæk Library is very successful, and all of a sudden it has twice as many readers as specified. Hiring librarians is easy – but how can we enhance the system to handle more traffic?

Arbitrating among RQAs: Example 3

- **Assume we demand 99.9999% reliability for an airbag controller.**
 - Assume that the controller determines whether or not to fire every 0.1s when the car is running.
 - Assume further that a car is operated no more than 1000h/year, which amounts to an operation period of $3.6 \cdot 10^6$ s/yr and thus $3.6 \cdot 10^7$ ops/yr for the airbag controller.
- **Six-nines reliability (99,9999%) corresponds to 1 failure in 10^6 operations.**
 - Ensuring this for the whole system lifespan of, say, 20 years will be exceedingly expensive.
- **Solutions**
 - Suppose that we can easily achieve an Airbag Mean-time-to-failure (MTTF) of 80.000 operating hours (approx. 22 years at 10h/day), but asking for more is very expensive.
 - Routine replacement of the airbags after 20.000 operating hours quadruples the reliability; most likely, even a taxi or a bus will have an accident before that or be decommissioned altogether.
 - Other solution: increase sample rate, ask for three confirming measurements in a row

Refining usability to functionality

- Like many quality attributes, usability requirements may be refined to a level where they turn into functional requirements.
- So, instead of providing an RQA level, we might provide a refinement to features. Here is an example.
 - R1:** A trained librarian shall be able to process the lending or returning of 10 books by 1 user in less than 5 seconds.
- This already rather detailed usability requirement may be replaced by a sequence of more detailed requirements realizing it.
 - R1.1:** All main functions (lend, return, reserve, prolong) and the search functions (simple and expert search) have both a key shortcut and a button on the main screen.
 - R1.2:** The core functions (lend, return) may be triggered by scanning a book and confirming with the [SPACE] key.
- Other usability requirements may translate into similar requirements.
 - R3.1:** The function search with suggestions may be triggered by scanning a book and timing out after 2s or the [S] key.
 - R3.2:** The functions prolong and search with suggestions may be triggered by scanning a book and confirming with the [P] key.
 - R3.3:** The keys used in key shortcuts are underlined on the buttons shown at the main screen. Pressing [ALT] and any of the accelerator keys will execute the respective command
 - R3.4:** Pressing the [F1] key will bring up a “Help” dialog that contains explanations of the main functions and the keyboard shortcuts.

Quality level by system type (1)

- **Consider the following requirements for a flight control system.**
 - The system shall trace the movements of up to 50 aircraft.
 - New positions of all aircraft shall be displayed at least once per 5 seconds.
- **But what happens if the 51st plane arrives?**
 - a. The system aborts and stops working altogether.
 - b. The system keeps tracking the first 50 planes, but ignores the 51st plane.
 - c. The system prints the error message “requirements violation”.
 - d. The system notifies the pilot of the 51st plane to leave the sector at once.
 - e. The system tracks all 51 planes but increases the update interval to 7s.
- **If there is no requirement stating the behavior in case of degradation, the designers (and implementers) are free to choose any of these options.**
- **And they can't be held responsible for it – but we as requirements engineers can and will be held responsible.**
- **For an air traffic control system, options (a) through (c) are clearly unacceptable, and option (e) is the best.**

Quality level by system type (2)

- **For a telephone switch system, similar requirements might be specified:**
 - The system shall handle up to 50 calls simultaneously at any time.
 - New calls shall be connected within 5 seconds.
- **What happens if the 51st call arrives?**
 - a. The system aborts and stops working altogether.
 - b. The system keeps handling first 50 calls, but ignores the 51st one.
 - c. The system notifies the operator of the error “requirements violation”.
 - d. The system notifies the 51st caller to hang up.
 - e. The system handles all 51 calls but increases connection time to 7s.
- **For this system, option (a) is clearly unacceptable, but all other options are ok – occasionally losing (b) or balking (d) a call is not too bad. Option (c) is not much use, and (e) – clearly the best option here, too – will rarely happen.**

Requirements by Role

- **As a client, you should sign for qualities rather than features**
 - they are much more comprehensive and easily cover many issues
 - They leave open the exact implementation, thus allow for unexpected improvements.
 - However, qualities are more difficult to handle than features.
- **Try to commit as late as possible.**
 - Any decision reduces your freedom to react on new information or improved insight.
- **As a provider, on the other hand, you should sign for features rather than qualities.**
 - It is much easier to provide and check features rather than qualities.
 - You may get away with providing less than what the client expects and still satisfying your contract, but unhappy clients don't return.
- **Try to get commitment.**
 - Any committed decision is either reliable, or the client can be charged for changing them.



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 8.5: Features vs. Qualities

DTU course 02264

Specification Level

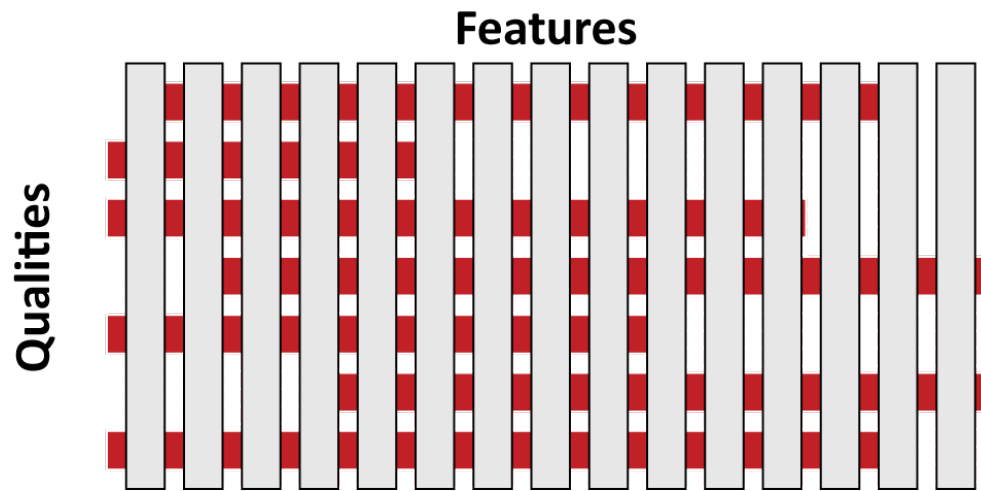
- **Many Qualities are abbreviations for bundles of features.**
 - For instance, we may find a requirement like this: *“Expert users shall be empowered to operate the application very quickly.”*
- **For a classical WIMP-interface used on an enterprise critical IS, we might translate it into these features.**
 - “All graphical User interfaces support a Tab-sequence.”
 - “All major functions have keyboard shortcuts”.
- **For a consumer product with a touch interface, we might interpret it differently, though.**
 - “Response time for all operations must be below 40ms for the start of the animation, and below 2s for completion of the operation.”
 - “All application data is cached locally on the device to compensate for slow or patchy network.”
- **Other qualities decompose into architectural decisions.**
 - This is usually the case for availability, performance, and so on.

Discrete vs. Continuous

- **Features come mostly in discrete quantities.**
 - For features it is always possible to state whether they are implemented or not, or exactly under which conditions they are available.
 - For example the required feature *“The system shall support undo/redo for all operations”* will be either implemented or not.
 - Usually, it is implemented with in some scope, e.g., operations like load/save are not maintained, the undo/redo history does not span across several launches of the system and so on.
 - Observe that such restrictions are not shared by many RDB products.
- **Qualities, on the other hand, are always continuous.**
 - It is meaningless to say a product is highly maintainable, available, etc.
 - When we say that, it is just a matter of speaking, a colloquial abbreviation for something more verbose and unwieldy such as *“The overall system has an availability of 95%, with a recovery time of 2 minutes or less and an annual window for scheduled maintenance of 5 hours”*.
- **Qualities are measured with continuous scales, Features with discrete ones.**
 - Qualities may be approximated by a sets of features (discretization).

Cross-cutting vs. Self-Containedness

- **Many features are self-contained, that is, they can be added, changed, and removed from a system without affecting much of the rest of the system.**
 - Of course, interfaces may have to be implemented causing additional effort.
 - Also, and there may be synergies arising from adding a feature.
 - However, technically, adding most features has little to no impact on existing features.
- **Qualities, on the other hand, are often cross-cutting.**
 - Any change to the required level of a given quality is likely to affect a large part of the system, i.e., other features and/or qualities (e.g. performance).
 - Some features are also cross-cutting, such as undo/redo. Notable exceptions to this rule include undo/redo, logging, or business rules and policies.



- **As a consequence of the properties discussed above, required quality attributes are usually not implemented by some additional code, but by the overall design (“architecture”) that acts as a blueprint for the implementation of the functionality.**
- **Therefore, it is more difficult to retrofit required quality attributes than to retrofit (most) required features.**
 - One could say that most required quality attributes are extremely crosscutting, but many required features are not.
 - There are crosscutting features, though, that are also difficult to retrofit, such as undo/redo, logging, or business roles/policies.
 - Sometimes, however, the situation is not that bad. For instance, demanding higher execution speed may be achievable by small changes (simple “tweaking”).
- **This is part of the reason why architecture is considered more difficult than “line development”, and why the professional profiles differ significantly.**

Quality Arbitrage

- **Qualities interact not just with features, they also interact with each other.**
 - Higher reliability will increase availability, because reduced error probability increases up-time.
 - Highly unreliable systems, however, can be just as available, if the repair time is very short.
 - So, if outage is not a safety risk as such (as in a nuclear reactor), “unreliable” and “safe” are not opposite ends of the same spectrum.
- **It would be extremely helpful to be able to do this kind of triage early in the development cycle.**
 - Assuming we have the RQAs in place is not enough, we also need analytical procedures to examine and simulate the software architecture.
 - Such methods are theoretically available, but not practically.
 - Recent advances include the AADLv2 (Architecture Analysis and Design Language, SAE AS-5506A).
 - RED is set to add these features in the near future.

Discovery Time

- **If important RQAs are not satisfied, the system as such may be entirely unusable and has to be scrapped completely.**
 - Think of an MP3-player: if the audio data is not processed fast enough, the user cannot listen to music—the whole product is completely useless.
- **That means that it is absolutely essential to have just the right RQAs up front of a development process whereas required features do not necessarily have to be available at that time.**
 - Discovering features late usually costs no more than discovering them early.
 - Discovering qualities late, on the other hand, is often crippling expensive.
- **Features can be managed much more flexibly.**
 - Any and all of the lightweight processes models (“agile”) depend on this property, and break down when it is not present. That is, for qualities.
 - Existing code will have to be scrapped (“refactoring” in agile lingo).

Specification Formalism

- **Features and qualities also differ in terms of which methods and notations/tools are suitable for specifying them.**
 - There are many different formalisms to specify features, we know more about them, and they are generally easier to handle.
 - Specifying qualities is tricky, and much less forgiving than specifying features.
- **Luckily, much software development does not rely heavily on professional specification of required quality attributes.**
 - In practice, most systems that are being built at all are either management information systems (MIS) or consumer products (CP).
 - They have mostly feature requirements and only weak quality requirements, mainly usability, performance, and integrity/compliance.
 - System software and embedded/real time (ERT) software has much more focus on qualities.
 - Revisit the classification of system types and the ensuing project complexity laid out in chapter 1.

Logical Status

- **Features and qualities are also different on logical level.**
- **A feature refers to a possibility, that is, an existentially quantified predicate.**
 - Features are effectively liveness conditions.
 - The requirement *“The reader may lend a book”* effectively means that there is a state of the system in which the function *“lend book”* is enabled.
 - If we find one such state, the property is satisfied.
- **A quality refers to a necessity, that is, a universally quantified predicate.**
 - Qualities are effectively safety conditions.
 - The requirement *“The reader may lend at most 10 books”* means that in all reachable states of the system, the number of lent books may not exceed 10.
 - All states have to be visited to be sure about this property.

Software vs. Buildings

- **Many people metaphorically equate software with buildings, and it also fits very well with the discussion in this section.**
 - Features: placement of doors and windows, interior decoration, plumbing details, ...
 - Qualities: how much space, how many rooms/floors, climatic conditions, energy consumption, basic plumbing (e.g., central heating or not, floor heating vs. convectors), ...
- **In Architecture, major additions to completed houses are difficult or infeasibly expensive, adding a basement, say.**
 - Nobody in their right minds would ask for it. Yet, in software, we are routinely asked to do just that.
- **However, we might often be able to make everybody happy with something else entirely.**
 - Maybe, the additional basement is demanded to house a hobby workshop. Instead of the basement, we might build a garage or a semi-detached shed.
 - They would have mostly the same function, but a totally different implementation: it does use more real estate, and might not look nice, but it solves the problem at reasonable cost.