


Requirements Engineering: Quality Models

CECS 590

COLLEGE OF ENGINEERING

CALIFORNIA STATE UNIVERSITY, LONG BEACH



Dean Forouzan Golshani and the Dean's Advisory Council
invite you to attend the
Engineering Distinguished Lecture Series

The Age of Drones and New Societal Concerns

Thursday, April 23, 2015

5:00- 5:15 p.m. Registration

5:15- 7:00 p.m. Panel of Experts/Audience Q & A

The Pointe Event and Conference Center at the Walter Pyramid, CSULB

Capstone projects with Intel

- Who is going to do their capstone projects starting in fall?
- Interested in doing stuff in RE?
- Be in touch with me.

Summer research assistant

- Who is here over summer and wants to do some research?
- I got a small grant for working with a student on a toolkit I want to develop.
- Come talk to me.

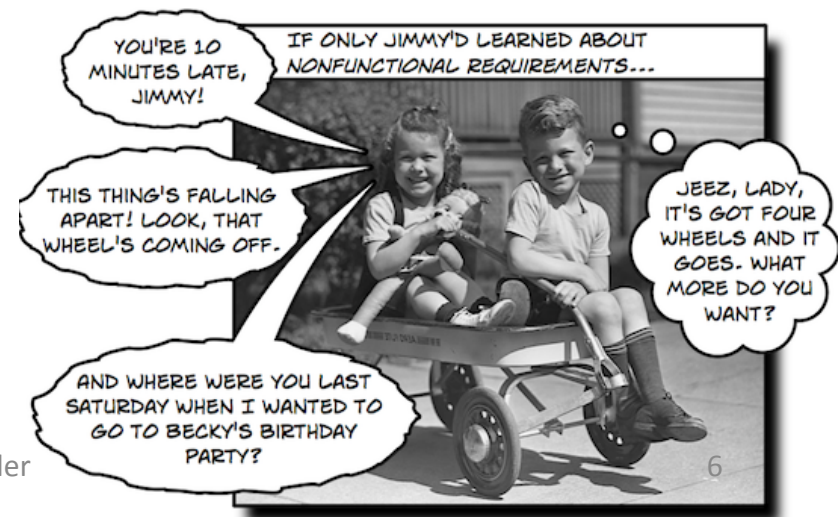
Website Web Developer Intern

- True Potential Counseling in Long Beach
- 3 month unpaid internship that has the potential to become a paid hourly position (10 hour per week) after the completion of the internship
- Candidate who knows both the front end side of web development, but also be able to link the site to back end technology (like aweber, teleseminar, stripe, 4square and paypal payment options, membership sites, automated system set-up) and do SOE

Recap time! NFRs



- What are NFRs?
- What are classification dimensions?
- What are the challenges?



Requirements Engineering – Outline

- WHY do we need Requirements Engineering and what is it?
- Principles: Definitions, process, roles, problem/solution view, artifact orientation
- System Models: Decomposition and abstraction, system views
- Frameworks: What reference structures can I use for requirements?
- Business Case Analysis: Why are we building this system?
- Stakeholders: Who are the people to talk to about requirements?
- Goals and Constraints: What are the major objectives for the system?
- System Vision: What exactly do we want to achieve?
- Domain Models: What are the surrounding systems ours interacts with?
- Usage Models: How will the system interact with the user?
- Quality requirements: How to specify which qualities need to be met?
- **Software quality models: How to determine the quality characteristics?**
- Process requirements: How to specify constraints for development?
- Towards a system specification: How to hand over to design?
- Quality assurance: How to ensure that RE is done in a good way?
- Change management: How to evolve requirements?

Motivation



[Gonalo Borrga: The truth about non-functional requirements. Blog entry, 2013.
<http://www.outsystems.com/blog/2013/03/the-truth-about-non-functional-requirements-nfrs.html>]

Functional Requirements and Their Poor Cousins: The Truth About NFRs

- **W**henever anybody says Functional Requirement, I think of princesses. I think of Arielle and Cinderella. I think of how each is central to her story and embodies a specific identity, and then I think of the princess who stands out as a true metaphor for functional requirements – the one who reflects the role perfectly. I think of Snow White.
- Snow White is a functional requirement if I ever saw one. She is at once central to her story, its main protagonist, its *raison d'être*, yet surrounded by a host of supporting characters – dwarfs – whose roles are necessary for the story to be complete. No dwarfs, no real story. It's that simple. If a single dwarf is missing, the entire story is compromised. Snow White is compromised.

[Gonçalo Borrêga: The truth about non-functional requirements. Blog entry, 2013.
<http://www.outsystems.com/blog/2013/03/the-truth-about-non-functional-requirements-nfrs.html>]

Functional Requirements and Their Poor Cousins: The Truth About NFRs

- And it's nearly the same with applications. If an application is the story and Snow White is the functional requirement, then we can think of dwarfs as non-functional requirements (NFRs). If a single NFR is missing from an application then that application is compromised. Deploying a squirrely application is the same as adding maintenance issues and technical debt directly to your application portfolio. Eventually, that application will have to be addressed, those NFRs will have to be added, and the IT department who deployed the app will have to contend with the cost of changing software.
- How does this happen? Changing and maintaining software is hard and therefore expensive to do and IT departments are often rushed or underfunded. However, if they are in a 'Just do it' mode then those operational, non-functional requirements that make an application complete are easy to leave out or are 'forgotten.' The consequences of leaving these NFRs out of an application lead directly to the aforementioned maintenance problems and increased technical debt.
- Here's the list of most common NFRs that we see our customers worrying about:

[Gonçalo Borrêga: The truth about non-functional requirements. Blog entry, 2013.
<http://www.outsystems.com/blog/2013/03/the-truth-about-non-functional-requirements-nfrs.html>]

Common NFRs



1. Maintainability

The ease with which the system can be changed, whether for bug fixes or to add new functionality. This is important because a large chunk of the IT budget is spent on maintenance. The more maintainable a system is, the lower the total cost of ownership will be.



2. Portability

The ease with which software can be installed on all necessary platforms and the platforms on which it is expected to run.

[Gonçalo Borrêga: The truth about non-functional requirements. Blog entry, 2013.
<http://www.outsystems.com/blog/2013/03/the-truth-about-non-functional-requirements-nfrs.html>]

Common NFRs



3. Reliability

The capability of the software to maintain its performance over time. Unreliable software fails frequently, and certain tasks are more sensitive to failure (for example, because they cannot be restarted, or because they must be run at a certain time).



4. Scalability

Software that is scalable has the ability to handle a wide variety of system configuration sizes. The nonfunctional requirements should specify the ways in which the system may be expected to scale up (by increasing hardware capacity, adding machines, etc.). Scalability ensures usability for five users or five thousand.

[Gonçalo Borrêga]

Common NFRs



5. Flexibility

If the organization intends to increase or extend the functionality of the software after it is deployed, that should be planned from the beginning; it influences choices made during the design, development, testing, and deployment of the system. Flexibility is the ease with which the system can be reused, deployed, and tested.



6. Auditability

When something goes wrong, you need to understand the root cause of it. So it is normal that auditability is a common NFR. The problem is that you hardly remember to have all the checkpoints in the process, all the exceptions logged, and to ensure that the subsystem to support it does not interfere with your application performance.

[Gonçalo Borrêga]

Common NFRs



7. Documentation

It's hard to keep complex system documentation up to date. Even if you're able to document the initial version of your application, by the time you've finished, the application has changed and its documentation is outdated.



8. Performance

The performance constraints specify the timing characteristics of the software. Certain tasks or features are more time-sensitive than others; the nonfunctional requirements should identify those software functions that have constraints on their performance.

[Gonalo Borrga]

Common NFRs



9. Security

Integrity requirements define the security attributes of the system, restricting access to features or data to certain users and protecting the privacy of data entered into the software.



10. Usability

Usability relates to how easily users can learn how to use a system and how efficient they are while using it. Highly usable systems reduce the effort required to read or input data and prevent users from making errors resulting in increased operational efficiency.

[Gonçalo Borrêga]

Functional Requirements and Their Poor Cousins: The Truth About NFRs

- Like Snow White's dwarfs these NFRs are necessary to completing the story of the application. While you might consider 2 or 3 important NFRs for a project (like performance and security), you'll probably not cover the others extensively enough, or you might miss out on them all together.
- And if you do allocate time to deal with them all, when the project schedule slips, the NFRs may be the first thing you'll drop... because no one really sees them, and your team will be looking at the functional requirements instead.
- So, whether you plan for NFRs or not, chances are high you won't cover them 100% of the time in your development project. You'll compromise and not think of the whole story – the whole application.
- But you should try. You should try to avoid adding technical debt and maintenance nightmares to your future portfolio whenever possible. The cost of change is real, and the moment you deploy your app, you'll have to address its problems.
- From your experience what are the NFRs you constantly see developers and project teams dropping most often?

Quality models and dealing with NFRs

- Usage of Quality models in RE
- Exemplary quality models
- Dealing with NFRs in AMDiRE

K Rayker, stock.xchng

Usage of quality models in RE

What are quality models?

→ Conceptual models for the description of quality.

Usage of quality models in RE

- Definition and assessment of software quality – beginning in RE
- Quality assurance, for example of artefacts in RE
- **Also: Classification** of requirements according to characteristics

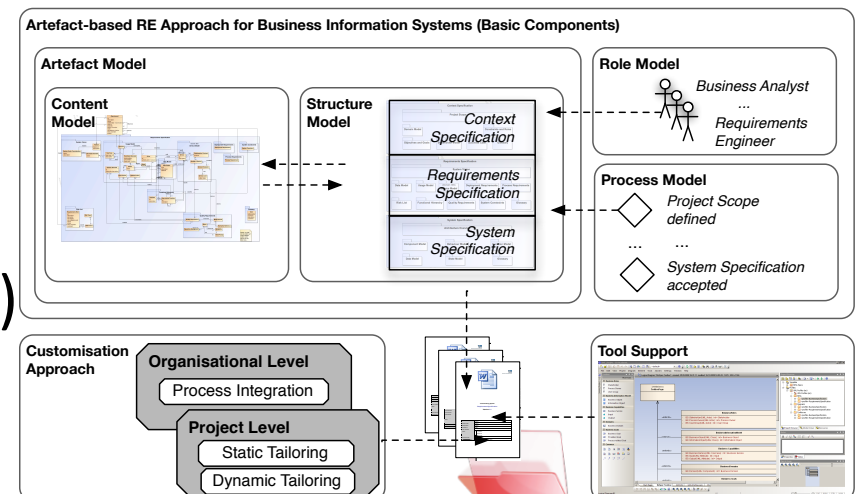
Usage of quality models in RE

Classification on the basis of quality models

- **Classification** of non-function requirements according to characteristics.
 - Which different classes of requirements exist?
 - Which aspects are important to consider?
 - Which modeling concepts and interdependencies are important to consider?

Delimitation: Artefact models

- Ideally build on quality models (compare to system models)
- Concept model (Content Model)
- Structure of the concept model (Structure Model)



Quality models and dealing with NFRs

- Usage of Quality models in RE
- Exemplary quality models
- Dealing with NFRs in AMDiRE

K Rayker, stock.xchng

Excursion: Quality models

Quality models

- Determine which quality aspects and concepts exist and how these are related
 - Support the structured elicitation and modeling of quality requirements
- E.g. via a taxonomy of quality attributes

Examples:

- Classification acc. to Boehm (1978)
- IEEE 29148 Software Requirements Documentation Standard (2011)
- ISO/IEC 9126 (1993, revised 2001)
- TUM S&SE Quality Model

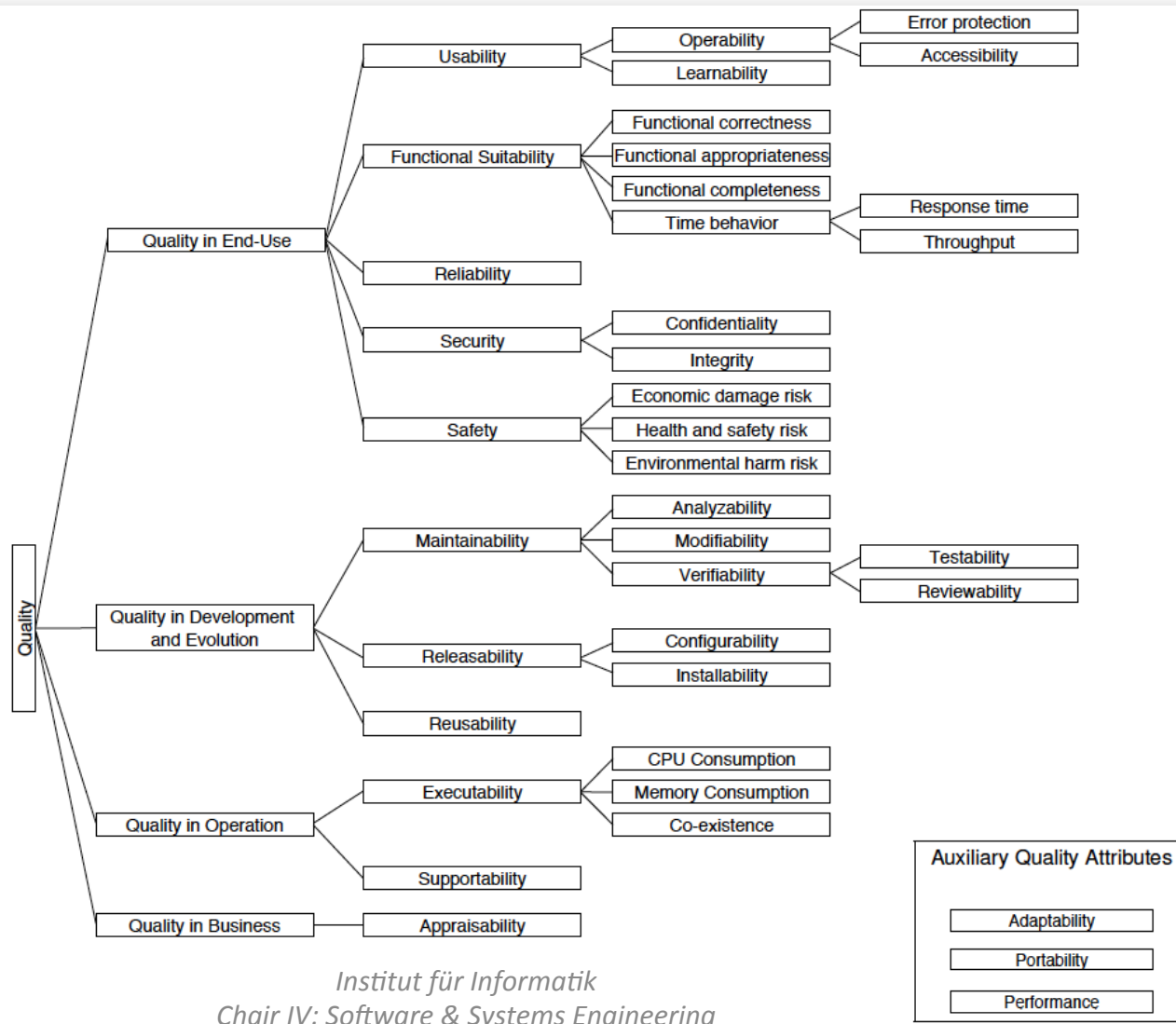
Challenges:

- Many different quality aspects and relations between them
- Systematics in their application in RE (as well as application in assessment)



[ISO Std.] (Excerpt)

Example: Taxonomy of quality attributes (QuaMoCo)



Assessment of quality models

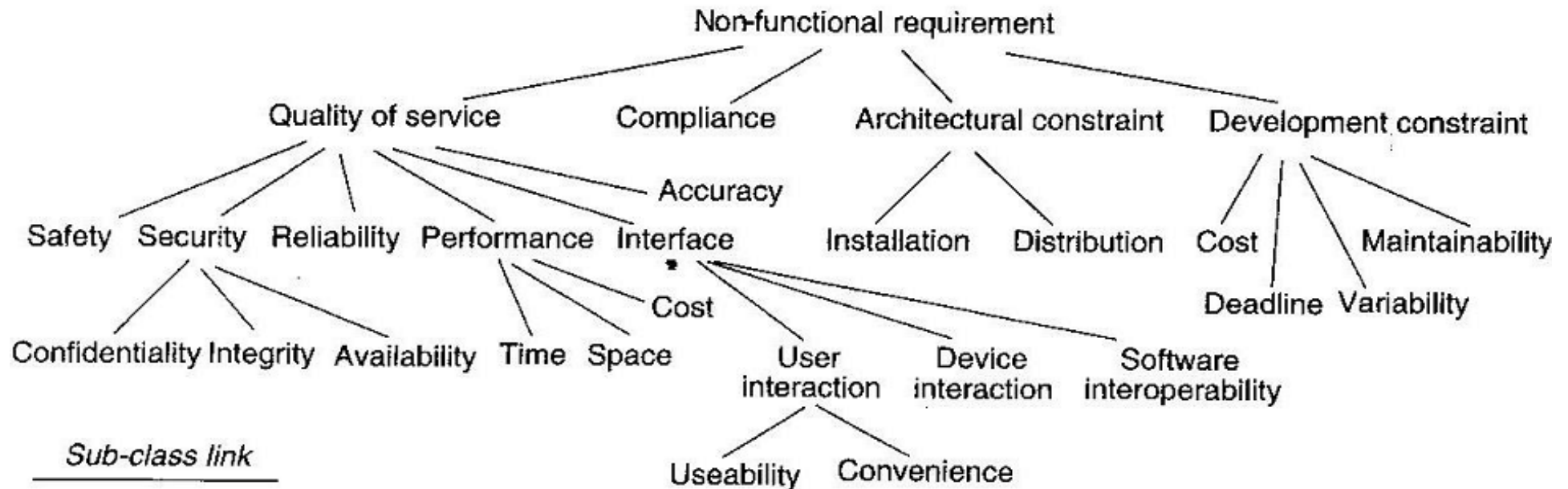
- Exemplary quality models determine the taxonomy of quality criteria and attributes

Critical aspects

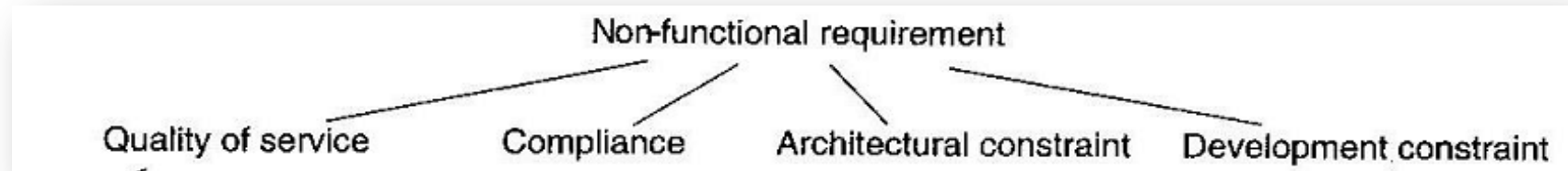
- The models often stay on the abstract level of „-ilities“
 - No statement about measurability and assessment of quality criteria
- No direct applicability to RE
 - No clues for *explicit* deduction of required system characteristics
Example: The demand for multi-language documentation supports maintainability, but also usability.
 - Other non-functional aspects are often not covered

But: quality models are the basis for the classification of NFRs.

NFR Taxonomy acc. to Axel van Lamsweerde



ATM Examples for NFR Taxonomy by Lamsweerde



- Security of personalized data
- Availability of the AMT
- ...

The security concept has to adhere to IEC 61508 Standard. (→ adequate architecture)

The ATM uses data encryption.

The full functionality shall be available by May 2014.



Summary

- Significance of existing NFR Taxonomies for RE:
 - Support structuring of NFRs
 - Define terminology

Assessment w.r.t. challenges:

1. Crosscutting Concerns

→ Interdependencies with fctl. requirements/ behavior models not clearly defined

2. Elicitation, assessment and evaluation

→ Refinement of non-fctl. requirements across various levels of abstraction?

→ Evaluation of non-functional requirements w.r.t.

- Interdependencies with other (NF) requirements?
- Implications for implementation/costs?

→ Measurable specification of non-functional requirements?