Quality Requirements – A New Look at an Old Problem

Martin Glinz

www.ifi.uzh.ch/rerg

JKU
JOHANNES KEPLER
UNIVERSITY LINZ

January 19, 2011

University of Zurich[UZH]

Department of Informatics

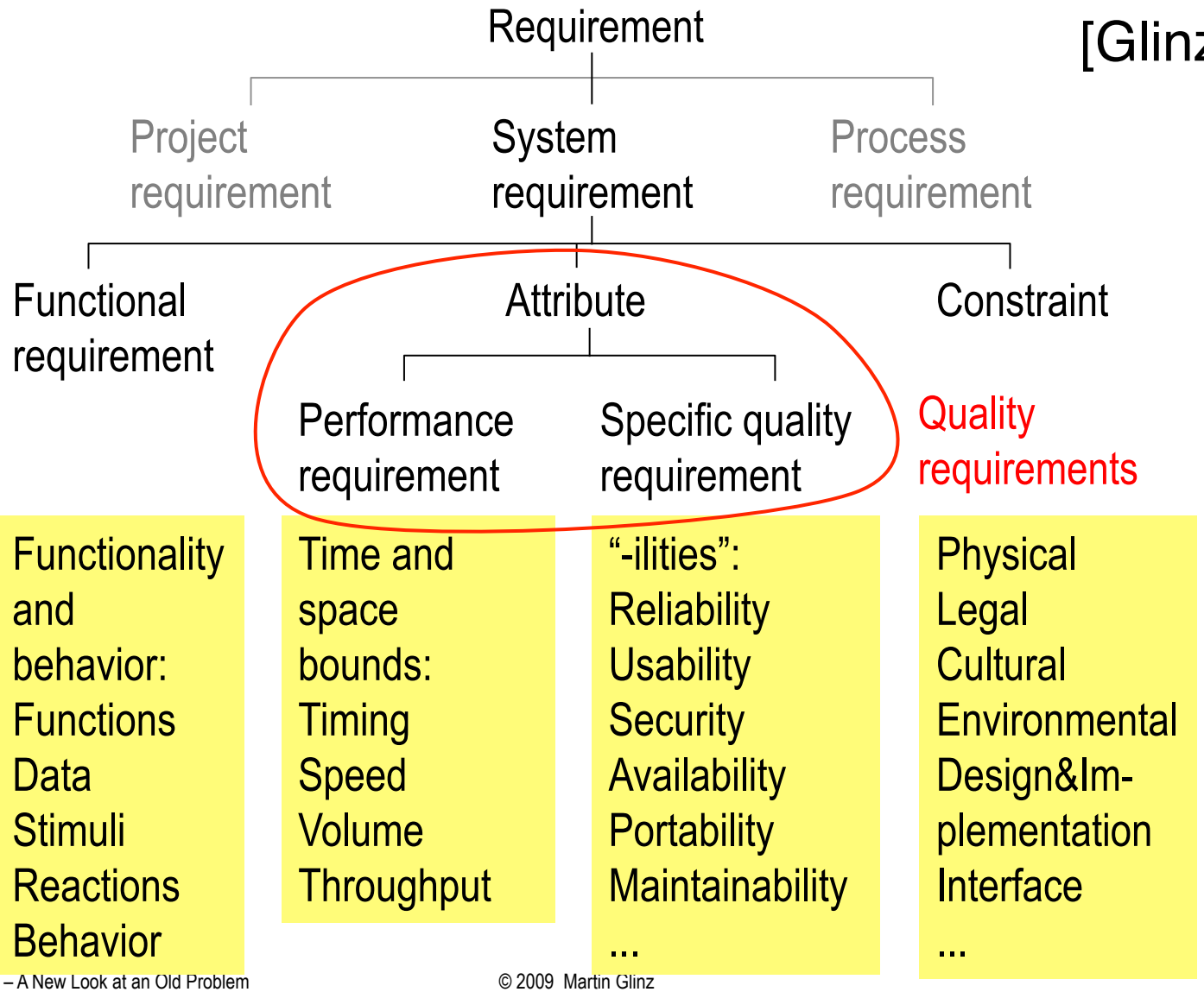
Requirements Engineering Research Group

Software Quality and Requirements

- Quality – The degree to which a set of inherent characteristics fulfils requirements [ISO 9000:2000]
- Software Quality – The degree to which software meets its requirements
- Requirement – A condition or capability that must be met or posessed by a system (or a system component) [IEEE 610.12]
- Quality requirements?

Classifying requirements

[Glinz 2007]



Quality requirements

Quality requirement – Those requirements that pertain to a system's attributes such as performance attributes or specific qualities.

Examples:

- "The system shall be user friendly."
- "The time interval between two consecutive scans of the temperature sensor shall be below two seconds."
- "The probability of successful, unauthorized intrusion into the database shall be smaller than 10^{-6}."

The problem: qualitatively stated requirements

"We need a secure system"

- Ambiguous
- Difficult to verify

Potential problems:

1. System delivers less than stakeholders expect
2. System delivers more than stakeholders need
3. Developers and customers disagree whether the system meets the requirements

The eleventh commandment

Thou shalt quantify.

Unambiguous • Verifiable • Low risk

If it's not measurable, make it measurable:

- Define / agree upon indirect measures for the desired property
- Demonstrate empirically that these indirect measurements are highly correlated with the quality we actually want to measure

There's a price tag

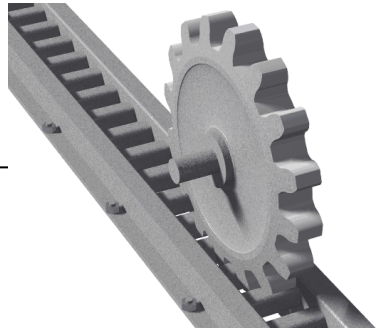
- Defining metrics costs
- Demonstrating validity of metrics costs

- Example:

  Quantifying "The system shall be user-friendly" according to ISO-IEC 9126 usability characteristics

  - Elicitation: Elicit values for 28 subcharacteristics
  - Verification: Compute values for 28 metrics

Operationalization



If you can't quantify it, make it operational.

- Expressing quality requirements as functional properties
- An alternative way of making quality requirements verifiable and unambiguous
- Solution-oriented, design decisions involved
- Connection to original quality requirements frequently lost

Any alternatives?

Is there life beyond
quantification and operationalization?

- Treat operationalization as a design task
- Consider alternatives to quantification in requirements

Requirements are a means, not an end

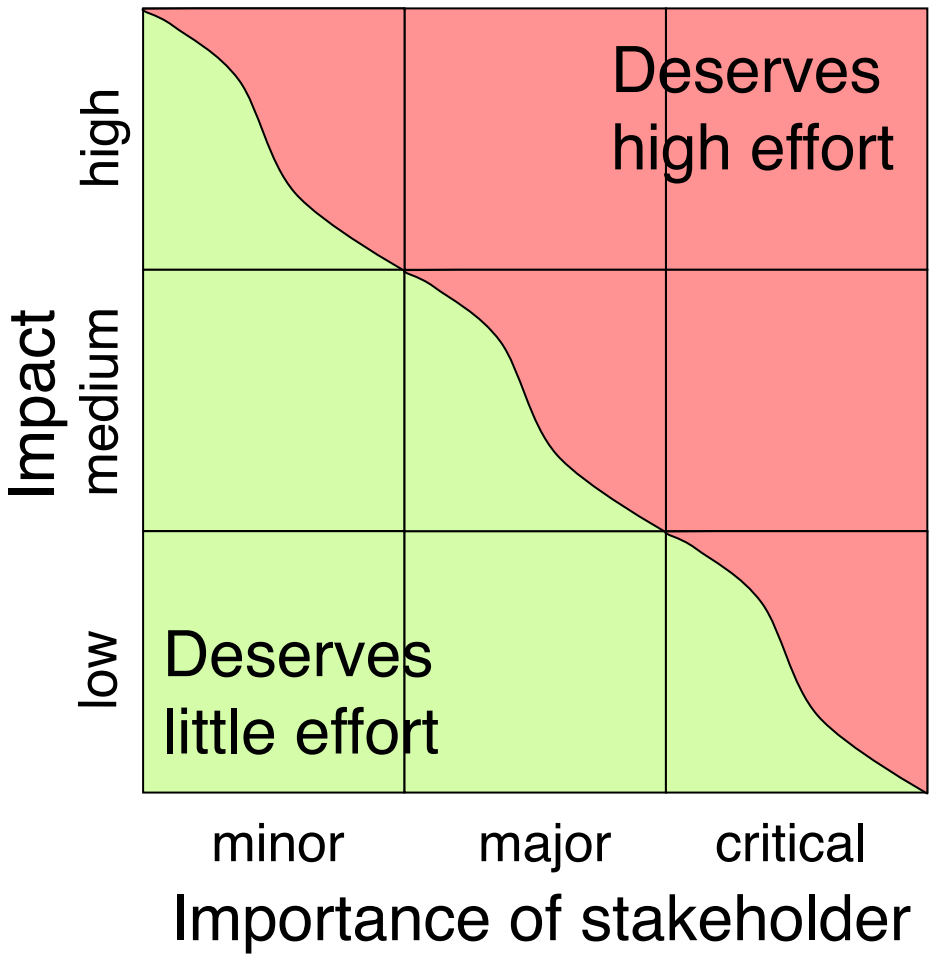
- Requirements shall deliver value
- Value of a requirement:
  - The benefit of reducing development risk (i.e. the risk of not meeting the stakeholders' desires and needs)
  - minus the cost of specifying the requirement

A new look: risk-based, value-oriented

- Classic thinking:
  Only a quantified quality requirement is a good quality requirement.

- New approach:
  A quality requirement should be represented such that it delivers optimum value.
  - Broader perspective
  - Does not dismiss full quantification
  - Choosing representation based on assessment of risk

Assessing risk



- Assess the criticality of the requirement
- Consider other factors (next slide)
- Use requirements triage techniques

Assessing risk: other factors

- Quantification effort
- Validity of obtained measurements
- Distinctiveness
- Shared understanding
- Reference systems
- Length of feedback-cycle
- Kind of customer-supplier relationship
- Certification required

The range of adequate representations

Situation	Representation	Verification
1. Implicit shared understanding	Omission	Implicit
2. Need to state general direction Customer trusts supplier	Qualitative	Inspection
3. Sufficient shared understanding to generalize from examples	By example	Inspection, (Measurement)
4. High risk of not meeting stake-holders' desires and needs	Quantitative in full	Measurement
5. Somewhere between 2 and 4	Qualitative with partial quantification	Inspection, partial measurement

A case study: Jane's volunteer driver service

An organization of volunteers who drive elderly or disabled people

- When a person needs transportation, he calls the number of the volunteer drivers' service.
- Two dispatchers, Jane and Peter, alternate in servicing incoming calls, making schedules, and calling volunteer drivers, giving them driving orders.
- They use a spreadsheet to create schedules.
- Jane, with help from Peter and his wife, founded the service three years ago. Jane and Peter are friends and know each other's work habits and preferences pretty well.

Jane's volunteer driver service – 2

- Today, the service has grown to 20 volunteers and is still growing.
- The board has decided they need a computer system to support the ordering and dispatching processes.
- In his professional life, Peter is the owner and chief engineer of a 10-person software company.
- He has decided to contribute by building an open source, free system by his company.

Stakeholder analysis

- Critical:
  - Dispatcher
- Major:
  - Service user
  - Volunteer driver
  - System operator
- Minor:
  - Developer
  - Executive board member
  - Person calling the service for somebody else
  - Any other stakeholder

Some samples and how to treat them

Lucy (a 76-year-old service user): “It would be great if I could make reservations over the Internet some weeks in advance and also view a list of my current reservations—I am a bit forgetful, you know. However, it must be very simple and easy to use.”

“Some weeks in advance”

Major stakeholder, low impact, easy to quantify ➪ Quantify.

“Very simple and easy to use”

Major stakeholder, low to medium impact, Hard to quantify ➪ Don’t quantify. Reduce risk with user interface prototyping and let selected service users inspect the fulfillment of this requirement by working with the prototype.

Some samples and how to treat them – 2

John (a deaf service user): "As I can't make calls myself, I need a Web-based reservation option. It should have the same service level as provided by calling today, in particular immediate confirmation in most cases."

"Same service level as provided by calling today"

Major stakeholder, low to medium impact, reference system available, hard to quantify ➪ Don't quantify. Instead, use current system as a reference system; maybe elaborate some examples that illustrate the current service level. Let selected service users inspect the fulfillment of this requirement by working with the prototype.

"Immediate confirmation in most cases"

Major stakeholder, low to medium impact, easy to quantify ➪ Quantify. For example, quantify "immediate" as "in less than 30 seconds" and "most cases" as "in at least 90% of all cases."

Some samples and how to treat them – 3

Jane (dispatcher): “My biggest concern is that the system must support the growing number of service requests but remain as simple to use as our current spreadsheet.”

“As simple to use as our current spreadsheet”

Critical stakeholder, medium impact, shared understanding between stakeholder and system architect, hard to quantify ➪ Don’t state explicitly as a requirement.

Some samples and how to treat them – 4

Peter (the dispatcher and head of the development team): “I primarily want the system to help me work faster— that is, on average, I’ll need less time to service an incoming request than today.”

“Need less time to service an incoming request than today”

Critical stakeholder, high impact, distinctive, easy to quantify ➪ Quantify

Threats / obstacles

- Stakeholders striving for process or personal benefit instead of value
- Regulatory complience required
- Distrust
- Need for “waterproof” contracts
- Inability to establish requirements validation procedures beyond test and measurement

Impact on architecture and implementation

What happens to quality requirements?

(1) Become part of the functionality

"secure" "simple and easy to use"

(2) Inform decisions on architectural, design and coding level

"highly portable" "response time < 0.5 s"

(3) Inform the software process

"maintainable"

Some change ahead

- Operationalization of quality requirements should become part of system design
- Role of architecture / design in contracts needs to be rethought
- Strictly applied waterfall models / V-models are harmful
- Pragmatics over dogmatics
- Requirements@runtime

Conclusion

- A new look at an old problem: getting optimum value from your effort to specify software quality
- Traditional quantification/operationalization of every quality requirement does not always deliver optimum value
- This value-based, risk-oriented approach
  - extends the classic approach
  - helps treat quality requirements adequately over a wide range of project situations
  - changes the requirements ←→ architecture relationship
  - ⇨ helps advance software quality

Further reading

focus

quality requirements

A Risk-Based, Value-Oriented Approach to Quality Requirements

Martin Glinz, *University of Zurich*

This value-oriented approach to specifying quality

When quality requirements are elicited from stakeholders, stated qualitatively, such as "the response time must be fast" a highly available system." (See the "Defining Quality R sidebar for a definition of quality requirements.) However

IEEE Software
Vol 25, No 2
March/April 2008
pp. 34-41

References and further reading

A. Davis (2005). *Just Enough Requirements Management*. Dorset House.

N.E. Fenton and S.L. Pfleeger (1998). *Software Metrics: A Rigorous and Practical Approach*, 2nd ed. PWS Publishing.

T. Gilb (1988). *Principles of Software Engineering Management*. Addison-Wesley.

T. Gilb (2005). *Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*. Butterworth-Heinemann.

M. Glinz (2007). On Non-Functional Requirements. *Proceedings of the 15th IEEE International Requirements Engineering Conference*, Delhi, India. 21-26.

Glinz, M. (2008). A Risk-Based, Value-Oriented Approach to Quality Requirements. *IEEE Software* **25**, 2. 34-41.

M. Glinz, R. Wieringa (2007). Stakeholders in Requirements Engineering. *IEEE Software* **24**, 2. 18-20.

IEEE (1990). *Standard Glossary of Software Engineering Terminology*. IEEE Std 610.12-1990.

IEEE (1998). *IEEE Recommended Practice for Software Requirements Specifications*. IEEE Standard 830-1998.

References and further reading – 2

ISO/IEC 25020:2007. *Software Engineering—Software Product Quality Requirements and Evaluation (Square)—Measurement Reference Model and Guide*. International Organization for Standardization.

ISO/IEC 9126-1:2001. *Software Engineering—Product Quality—Part 1: Quality Model*. International Organization for Standardization.

ISO/IEC TR 9126-2:2003. *Software Engineering—Product Quality—Part 2: External Metrics*. International Organization for Standardization.

ISO/IEC TR 9126-3:2003. *Software Engineering—Product Quality—Part 3: Internal Metrics*. International Organization for Standardization.

ISO/IEC TR 9126-4:2004. *Software Engineering—Product Quality—Part 4: Quality in Use Metrics*. International Organization for Standardization.

ISO/IEC 25020:2007. *Software Engineering—Software Product Quality Requirements and Evaluation (Square)—Measurement Reference Model and Guide*. International Organization for Standardization.

ISO/IEC 25030:2007. *Software Engineering—Software Product Quality Requirements and Evaluation (Square)—Quality Requirements.* International Organization for Standardization.