Non-functional Requirements

Ze Shi (Zane) Li

thesegalgroup.org

lize@uvic.ca

Outline

Non functional requirements

NFRs in practice

Software Quality

Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

[Pressman, 1997]

Implications

Software *requirements* are the foundation from which quality is measured. Lack of conformance to requirements is a lack of quality.

Specified *standards* define a set of development criteria that guide the manner in which software is engineered.

There is a set of *implicit requirements* that often go unmentioned (e.g. the desire for good maintainability).

Functional vs Non-Functional Requirements

functional requirements describe fundamental functions of the system

non functional requirements (NFRs) describe
- Constraints on the system
- Constraints from the application domain

On NFRs (from Glinz)

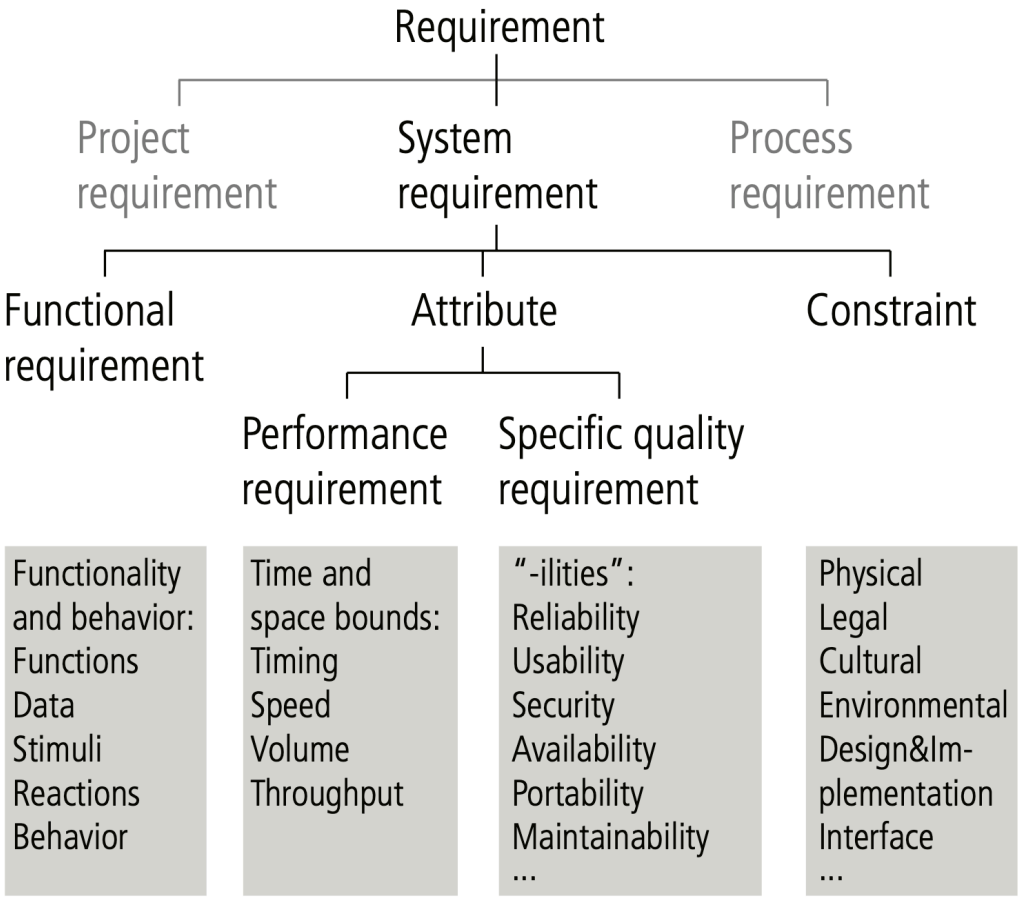


Figure 2. A concern-based taxonomy of requirements

Some Definitions

Reliability: extent to which a program can be expected to perform its intended function with required precision

Efficiency: amount of computing resources and code required by a program to perform a function

Integrity: extent to which access to software or data by unauthorized persons can be controlled

Usability: effort required to learn, operate, prepare input and interpret output of a program

Maintainability: effort required to test a program to ensure that it performs its intended function

Flexibility: effort required to modify an operational program

Portability: effort required to transfer a program from one hardware and/or software environment to another

Reusability: extent to which a program (or parts thereof) can be reused in other applications

Interoperability: effort required to couple one system with another

Connect NFRs to Design

NFRs often "architecturally significant"

Get these wrong and cost/effort increase

Frequently "fuzzy" and hard to sell to management

Solution: "quality attribute scenarios"

Quality attribute scenarios (QAS)

Are measurable, priorities 'tests' of a system's quality attributes

In simple form:

Quality attribute, stimulus, response, response measure

Quality attribute scenarios (QAS)

Quality attribute, stimulus, response, response measure

"Maintainability: when a new feature is added, the new feature is deployed in <4 hours"

"Security: there is never any exposure of PII"

"Availability: servers in user's region are unavailable within 100ms less than 1 hr a month"

Why QAS

Measurable = objective

Test design or implementation

Be aware of specifying 100% as the expected value for reliability or availability

Acceptance criteria - when does someone sign off on the system?

Examples of Non Functional Requirements

Performance requirements

Reliability, e.g. system must have less than 1 hr downtime per year.

Security, e.g. permissions for access

time- or space related, e.g. the system must handle 1,000 transactions per second

Survivability, e.g. the system must survive fire

Operating requirements

Physical or environmental *constraints*, e.g. weight, temperature)

Interface requirements

Usability, related to user-interfaces and "user-friendliness"

Interfaces with other systems in the operational environment

Life-cycle requirements

Maintainability, portability

Economic requirements

Restrictions on *development costs*

Software quality *attributes* into Software quality *criteria*

Software quality *attributes*

(*customer*-related concerns, NFR's observable by the user)

Requirements

E.g. maintainability, efficiency, correctness, interoperability, (re-)usability, survivability.

Realizes

Software quality *criteria*

(design criteria, *development*-oriented concerns)

Design

E.g. visibility, consistency.

External

Internal

Treatment of NFRs

Quality attributes (NFRs) and software quality criteria are interrelated

E.g. **correctness** related to **consistency**

During design:

Identify the relative importance of each *quality attribute* (NFR)
Identify the design criteria on which these factors depend (*software quality criteria*)
Make the requirements measurable.

Maintainability related to **software visibility**

Software visibility: **measuring** the amount of branching in a system

➔ **(Measurable) Requirement:** "there shall be no more than X branches per 1,000 lines of code"

Making requirements measurable

Difficult to do, but examples exist

NFR	Possible metric	Possible measurement
Usability	Time taken to learn how to use	Minutes taken for some user task
Complexity	Information flow between modules	Number of procedure calls
Reliability	Mean time to failure	Number of crashes in a specified period of time

Software Reliability

Probability of failure on demand: a measure of the likelihood that the system will behave in an unexpected way when some demand is made on it

Mean time to failure (MTTF): a measure of the time between observed failures

Availability

A measure of how likely the system is to be available for use

For example, an availability of 998/1000 means that in 1000 time units, the system is likely to be available for 998 of these time units

Availability	Downtime / Year	Downtime / Month	Downtime / Week	Downtime / Day
99.999%	5.256 Minutes	0.438 Minutes	0.101 Minutes	0.014 Minutes
99.995%	26.28 Minutes	2.19 Minutes	0.505 Minutes	0.072 Minutes
99.990%	52.56 Minutes	4.38 Minutes	1.011 Minutes	0.144 Minutes
99.950%	4.38 Hours	21.9 Minutes	5.054 Minutes	0.72 Minutes
99.900%	8.76 Hours	43.8 Minutes	10.108 Minutes	1.44 Minutes
99.500%	43.8 Hours	3.65 Hours	50.538 Minutes	7.2 Minutes
99.250%	65.7 Hours	5.475 Hours	75.808 Minutes	10.8 Minutes
99.000%	87.6 Hours	7.3 Hours	101.077 Minutes	14.4 Minutes

Exercise

Within your project team

Pick a relevant NFR for your developer project

Create a Quality Attribute Scenario

NFRs in Research

Privacy In Industry

GDPR study in a small software organization

How do they adopt privacy in their organization

Lack of Shared Understanding of NFRS in Continuous Software Engineering (RE20)

CSE (CI/CD more commonly used in software organizations)

What contributes to a lack of shared understanding of NFRs?

What NFRs are most associated with a lack of shared understanding?

What amount of a lack of shared understanding of NFRs is **accidental** versus **essential**

Multi-Case Study

3 small, agile organizations using CSE and cloud based platforms

In depth analysis of 30K development tasks (bug, feature, story, epic)

Coded tasks in relation to NFRs/Lack of shared understanding/ Rework with partner organizations

Multi-Case Study

We analyzed 2.5k tasks

348 were identified as rework, validated 174 with partners

41 confirmed as rework due to lack of shared understanding of NFRs

Causes for Lack of Shared Understanding

Fast pace of change (80%)

- usability, deployability, extensibility

Lack of domain knowledge (71%)

- maintainability, extensibility, scalability

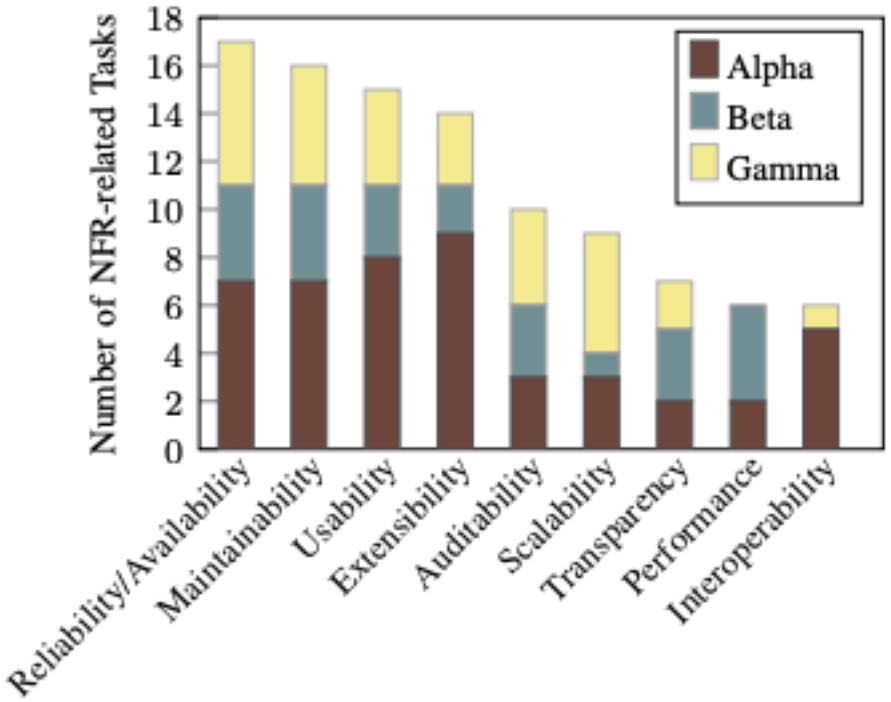
Inadequate communication (37%)

- reliability, maintainability, performance

Most associated with Lack of Shared Understanding

Any guesses?

Most associated with Lack of Shared Understanding



Amount of Lack of Shared Understanding is Accidental

78% of lack of shared understanding was accidental

22% is essential

How can we avoid this?

Shared development standards

- more development standards for maintainability

- standardizing deployability

- standardizing development for usability

How can we avoid this?

Adequate communication and documentation

- communication issues often between support analysts, testers, product manager, managers

- documentation, code review, walk through

Documentation is communication that takes the form of explicit shared understanding

References

Pressman, R. Software Engineering: A practitioner's approach, McGraw-Hill, 1997

Mylopoulos, J., Chung, L. and Nixon, B. Representing and using nonfunctionalrequirements: a process-oriented approach, IEEE Transactions of software engineering,18(6), 1992

Bass, Clements, Kazman. "Software Architecture in Practice". Addison-Wesley, 2020, 4th ed.

Glinz, M. "On Non-Functional Requirements", IEEE International Requirements Engineering Conference (RE07),

Ernst, N, https://github.com/uvic-seng321/course/tree/main/modules/overview