Lecture 6: Non-Functional Requirements (NFRs)

→ What are non-functional requirements

→ Product-oriented qualities

→ Process-oriented qualities

→ Traceability between requirements

What are Non-functional Requirements?

→ Functional vs. Non-Functional

功能性要求描述系统应该做什么

- 事情可以被捕捉在用例中
- 事情可以通过绘制序列图、状态图等方式分析
- 功能性要求将可能映射到程序的单个部分

非功能性要求是软件系统的全局约束

- 例如：开发成本、运营成本、性能、可靠性、可维护性、可移植性、健壮性等
- 往往被称为“-ilities”
- 通常无法在单个程序的模块中实施
What are Non-functional Requirements?

→ Functional vs. Non-Functional

Functional requirements describe what the system should do
- things that can be captured in use cases
- things that can be analyzed by drawing sequence diagrams, statecharts, etc.
- Functional requirements will probably trace to individual chunks of a program

Non-functional requirements are global constraints on a software system
- e.g. development costs, operational costs, performance, reliability, maintainability, portability, robustness etc.
- Often known as the “-ilities”
- Usually cannot be implemented in a single module of a program

Goals as selection criteria
The challenge of NFRs

- **Hard to model**

- **Usually stated informally, and so are:**
  - often contradictory,
  - difficult to enforce during development
  - difficult to evaluate for the customer prior to delivery

- **Hard to make them measurable requirements**
  - We’d like to state them in a way that we can measure how well they’ve been met

Example NFRs

→ **Interface requirements**
  - how will the new system interface with its environment?
    - User interfaces and “user-friendliness”
    - Interfaces with other systems

→ **Performance requirements**
  - time/space bounds
    - workloads, response time, throughput and available storage space
    - e.g. "the system must handle 1,000 transactions per second"
  - reliability
    - the availability of components
    - integrity of information maintained and supplied to the system
    - e.g. "system must have less than 1hr downtime per three months"
  - security
    - E.g. permissible information flows, or who can do what
  - survivability
    - E.g. system will need to survive fire, natural catastrophes, etc
Example NFRs

→ Operating requirements
  - physical constraints (size, weight),
  - personnel availability & skill level
  - accessibility for maintenance
  - environmental conditions
  - etc

→ Lifecycle requirements
  - “Future-proofing”
    - Maintainability
    - Enhanceability
    - Portability
    - expected market or product lifespan
  - limits on development
    - development time limitations,
    - resource availability
    - methodological standards

→ Economic requirements
  - e.g. restrictions on immediate and/or long-term costs.

Approaches to NFRs

→ Product vs. Process?
  - Product-oriented Approaches
    - Focus on system (or software) quality
    - Aim is to have a way of measuring the product once it’s built
  - Process-oriented Approaches
    - Focus on how NFRs can be used in the design process
    - Aim is to have a way of making appropriate design decisions

→ Quantitative vs. Qualitative?
  - Quantitative Approaches
    - Find measurable scales for the quality attributes
    - Calculate degree to which a design meets the quality targets
  - Qualitative Approaches
    - Study various relationships between quality goals
    - Reason about trade-offs etc.
Framework for Evaluation of Functional RE-tool Requirements

**FEF1. Representation dimension**
- **FEF1.1.** Specify uniquely identifiable description using informal language.
- **FEF1.2.** Specify requirements using semi-formal language(s).
- **FEF1.3.** Specify requirements using formal language(s).
- **FEF1.4.** Define traceable associations between requirements and the different elements of requirements specification.
- **FEF1.5.** Connect seamlessly with other tools and systems, by supporting interoperable protocols and standards.

**FEF2. Agreement dimension**
- **FEF2.1.** Maintain an audit trail of changes, archive baseline versions; and engage a mechanism to authenticate and approve change requests.
- **FEF2.2.** Classify requirements into logical user-defined groupings.
- **FEF2.3.** Support secure, concurrent cooperative work between members of a multidisciplinary team, which may be geographically distributed.
- **FEF2.4.** Maintain a comprehensive data dictionary of all project components and requirements in a shared repository.

**FEF3. Specification dimension**
- **FEF3.1.** Collect and store a common system’s and a product family’s domain requirements.
- **FEF3.2.** Generate predefined and ad hoc reports, documents that comply with standard industrial templates, with support for presentation-quality output and in-built document quality controls.
- **FEF3.3.** Generate the complete specification, expressed using formal language (informal and semi-formal languages might also be included), commonly agreed by all stakeholders.

Framework for Evaluation of Non-Functional RE-tool Requirements

**NF1. Process requirements**
- **NF1.1.** Support the RE process and requirements specification standards.
- **NF1.2.** Support the modelling perspectives.
- **NF1.3.** Support the software development models.
- **NF1.4.** Support the interfaces with the editing, modelling and implementation tools.

**NF2. Product requirements**
- **NF2.1.** Provide a user satisfying usability of the system.
- **NF2.2.** Provide a user satisfying reliability of the system.
- **NF2.3.** Provide a user satisfying performance of the system.
- **NF2.4.** Provide a user satisfying supportability of the system.

**NF3. External requirements**
- **NF3.1. Organisational requirements**
  - **NF3.1.1.** Decide about the biggest costs the organisation is able to pay for the RET evaluation?
  - **NF3.1.2.** Evaluate the decision making approach in the organisation.
  - **NF3.1.3.** Investigate organisational experience of the software usage.
  - **NF3.1.4.** Define the legitimate constraints social norms.
- **NF3.2. Requirements for business parties**
  - **NF3.2.1.** What facilities the business parties are suggesting for user training?
  - **NF3.2.2.** Do the business parties provide the adequate system maintenance?
  - **NF3.2.3.** Do the business parties provide the adequate system support?
R.1. FEF1.1, FEF1.2, FEF1.3 and NF1.2 – relationship between the modelling perspective and the languages supported by the RE-tool.
R.2. NF1.4 and FEF1.5 – relationship between the tools already used and the RE-tool.
R.3. NF1.3 and RE-tool functional features consider the software development model, used by the user.
R.4. NF1.1 and FEF3.3 – relationship between the RE process and specification standards.
R.5. Non-functional RE-tool features (NF2.1, NF2.2, and NF2.3) and functional RE-tool features - relationship between the RE-tool usability, efficiency, reliability and functionality.
R.6. NF2.4 and the functionality consider the need to maintain the non-functional RE-tool features against the functional characteristics by the customer.
R.7. NF3.2.2, NF3.2.3 and NF2.4 – relationship of maintenance and support between the customer and the business parties.
R.8. NF3.2.2, NF3.2.3 and the functionality – relationship of the RE-tool maintenance and support by the business parties.
R.9. NF3.1.3 and NF3.2.1 – relationship between the customer’s knowledge and the need for training.
R.10. NF1.1 and NF3.1.4 – relationship between the standards and the social, organisational, law and cultural factors.
Software Qualities

→ Think of an everyday object
  - e.g. a chair
  - How would you measure it’s “quality”?
    ➢ construction quality? (e.g. strength of the joints,…)
    ➢ aesthetic value? (e.g. elegance,…)
    ➢ fit for purpose? (e.g. comfortable,…)

→ All quality measures are relative
  - there is no absolute scale
  - we can sometimes say A is better than B…
    ➢ … but it is usually hard to say how much better!

→ For software:
  - construction quality?
    ➢ software is not manufactured
  - aesthetic value?
    ➢ but most of the software is invisible
    ➢ aesthetic value matters for the user interface, but is only a marginal concern
  - fit for purpose?
    ➢ need to understand the purpose

Fitness

→ Software quality is all about fitness to purpose
  - does it do what is needed?
  - does it do it in the way that its users need it to?
  - does it do it reliably enough? fast enough? safely enough? securely enough?
  - will it be affordable? will it be ready when its users need it?
  - can it be changed as the needs change?

→ Quality is not a measure of software in isolation
  - it measures the relationship between software and its application domain
    ➢ cannot measure this until you place the software into its environment…
    ➢ …and the quality will be different in different environments!
  - during design, we need to predict how well the software will fit its purpose
    ➢ we need good quality predictors (design analysis)
  - during requirements analysis, we need to understand how fitness-for-purpose will be measured
    ➢ What is the intended purpose?
    ➢ What quality factors will matter to the stakeholders?
    ➢ How should those factors be operationalized?
Boehm’s NFR list
Source: See Blum, 1992, p176

McCall’s NFR list
Source: See von Vliet 2000, pp111-3
Making Requirements Measurable

→ We have to turn our vague ideas about quality into measurables

The Quality Concepts
(abstract notions of quality properties)

Measureable Quantities
(define some metrics)

Counts taken from Design Representations
(realization of the metrics)

Examples...

- reliability
  - mean time to failure?
- complexity
  - information flow between modules?
- usability
  - time taken to learn how to use?
  - minutes taken for some user task?

Example Metrics

<table>
<thead>
<tr>
<th>Quality</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>transactions/sec</td>
</tr>
<tr>
<td></td>
<td>response time</td>
</tr>
<tr>
<td></td>
<td>screen refresh time</td>
</tr>
<tr>
<td>Size</td>
<td>Kbytes</td>
</tr>
<tr>
<td></td>
<td>number of RAM chips</td>
</tr>
<tr>
<td>Ease of Use</td>
<td>training time</td>
</tr>
<tr>
<td></td>
<td>number of help frames</td>
</tr>
<tr>
<td>Reliability</td>
<td>mean-time-to-failure,</td>
</tr>
<tr>
<td></td>
<td>probability of unavailability</td>
</tr>
<tr>
<td></td>
<td>rate of failure, availability</td>
</tr>
<tr>
<td>Robustness</td>
<td>time to restart after failure</td>
</tr>
<tr>
<td></td>
<td>percentage of events causing failure</td>
</tr>
<tr>
<td>Portability</td>
<td>percentage of target-dependent statements</td>
</tr>
<tr>
<td></td>
<td>number of target systems</td>
</tr>
</tbody>
</table>
Making Requirements Measurable

→ Define ‘fit criteria’ for each requirement
  ➜ Give the ‘fit criteria’ alongside the requirement
  ➜ E.g. for new ATM software
    ➢ Requirement: “The software shall be intuitive and self-explanatory”
    ➢ Fit Criteria: “95% of existing bank customers shall be able to withdraw money and deposit cheques within two minutes of encountering the product for the first time”

→ Choosing good fit criteria
  ➜ Stakeholders are rarely this specific
  ➜ The right criteria might not be obvious:
    ➢ Things that are easy to measure aren’t necessarily what the stakeholders want
    ➢ Standard metrics aren’t necessary what stakeholders want
  ➜ Stakeholders need to construct their own mappings from requirements to fit criteria

Usability requirements

Usability = Fit for use + Ease of use

→ Ease of learning
  ➜ How easy is the system to learn for various groups of users?

→ Task efficiency
  ➜ How efficient is it for the frequent user?

→ Ease of remembering
  ➜ How easy is to remember for the occasional user?

→ Subjective satisfaction
  ➜ How satisfied is the user with the system?

→ Understandability
  ➜ How easy is it to understand what the system does?

<table>
<thead>
<tr>
<th>Satisfactory</th>
<th>Rating</th>
<th>Quality Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response time (sec)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt; 2</td>
<td>Exceeds expectation</td>
<td>3</td>
</tr>
<tr>
<td>2-5</td>
<td>Within the target range</td>
<td>2</td>
</tr>
<tr>
<td>6-10</td>
<td>Minimally acceptable</td>
<td>1</td>
</tr>
<tr>
<td>&gt;10</td>
<td>Unacceptable</td>
<td>0</td>
</tr>
</tbody>
</table>
Usability requirements

→ Q1: It should be easy for novice users to do tasks Q and R.
→ Q2: Novice users should perform tasks Q and R in a short time.
→ Q3: Experienced users complete tasks Q, R, and S quicker than novice users.
→ Q4: Recording breakfast shall be easy using keyboard.

→ Problem counts
    Q1: At most 1 of 20 novices shall encounter critical problems during tasks Q and R.

→ Task time
    Q2: Novice users shall perform tasks Q and R in 15 minutes.
    Q3: Experienced users complete tasks Q, R and S in 2 minutes.

→ Keystroke counts
    Q4: Recording breakfast shall be possible within 5 keystrokes per guest.
Efficiency

- **Efficiency** - the capability of the software to provide the required performance relative to the amount of resources used, under stated conditions

Which statement is objectively defined?

1: Product shall be able to process a lot of payment transactions in a short time even during peak load

2: Product shall be able to process 100 payment transactions per second in peak load.
Efficiency

• **Efficiency** - the capability of the software to provide the required performance relative to the amount of resources used, under stated conditions

Which statement is objectively defined?

1: Scrolling one page up or down in a 200 page document shall take at most 1s. Searching for a specific keyword shall take at most 5s.  
2: Scrolling one page up or down in a large document shall take an efficient time. Searching for a specific keyword shall a reasonable time

Efficiency

• **Efficiency** - the capability of the software to provide the required performance relative to the amount of resources used, under stated conditions

Which statement is objectively defined?

1: A simple report shall take less a short time. None shall take the long waiting time  
2: None shall take longer than the simple report preparation in the majority of the cases.
Maintainability

→ Maintenance performance
  Q1: Supplier’s hotline shall analyse almost all reports in a short period
  Q2: When repairing a defect, a number of related non-repaired defects should be very low

→ Development process
  Q3: Every program module must be assessed for maintainability according to organisation’s standards OST-1.12.x. Majority of the modules have to be “High maintainable” (as defined in the standard) and none “poor” (as defined in the standard)
  Q4: Development must use regression test allowing full re-testing in a short period

→ Program complexity
  Q5: No method in any object may contain a lot of code lines

Maintainability

→ Maintenance performance
  Q1: Supplier’s hotline shall analyse 95% of reports within 2 work hours.
  Q2: When repairing a defect, related non-repaired defects shall be less than 0.5 coverage

→ Development process
  Q3: Every program module must be assessed for maintainability according to organisation’s standards OST-1.12.x. 70% of modules must obtain “High maintainable” (as defined in the standard) and none “poor” (as defined in the standard).
  Q4: Development must use regression test allowing full re-testing in 12 hours.

→ Program complexity
  Q5: No method in any object may exceed 200 lines of code
Lecture 6:
Non-Functional Requirements (NFRs)

→ What are non-functional requirements

→ Product-oriented qualities

→ Process-oriented qualities

→ Traceability between requirements