Lecture 7: Requirements Modeling III

Last Week:
Modeling II
Information
Structure
Behaviour

This Week:
Modeling System Qualities
Non-functional Requirements
Satisficing Softgoals
Quality measures

Next Week:
Specification and Validation
Specification Languages
Documentation Standards
Reviews and Inspections

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 interaction 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

→ 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

→ 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
  - E.g development time limitations,
  - resource availability
  - methodological standards
  - etc.

→ 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.

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

Source: Budgen, 1994, pp58-9

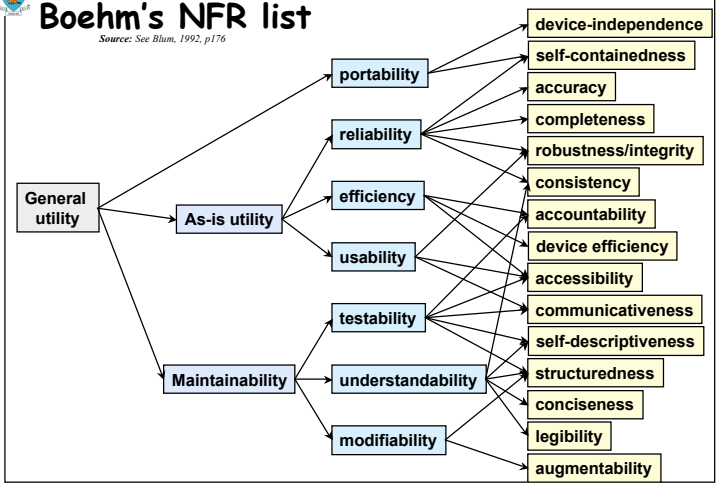
- Software quality is all about fitness to purpose
  - does it do what is needed?
  - does it do it in the way that its users need it to?
  - does it do it reliably enough? fast enough? safely enough? securely enough?
  - will it be affordable? will it be ready when its users need it?
  - can it be changed as the needs change?
- But this means quality is not a measure of software in isolation
  - it is a measure of the relationship between software and its application domain
    - might not be able to measure this until you place the software into its environment…
    - …and the quality will be different in different environments!
  - during design, we need to be able to ***predict*** how well the software will fit its purpose
    - we need to understand that purpose (requirements analysis)
    - we need to look for quality predictors (design analysis)

Factors vs. Criteria

- Quality Factors
  - These are customer-related concerns
    - Examples: efficiency, integrity, reliability, correctness, survivability, usability,…
- Design Criteria
  - These are technical (development-oriented) concerns such as anomaly management, completeness, consistency, traceability, visibility,…
- Quality Factors and Design Criteria are related:
  - Each factor depends on a number of associated criteria:
    - E.g. correctness depends on completeness, consistency, traceability,…
    - E.g. verifiability depends on modularity, self-descriptiveness and simplicity
  - There are some standard mappings to help you…
- During Analysis:
  - Identify the relative importance of each quality factor
    - From the customer's point of view!
  - Identify the design criteria on which these factors depend
  - Make the requirements measurable

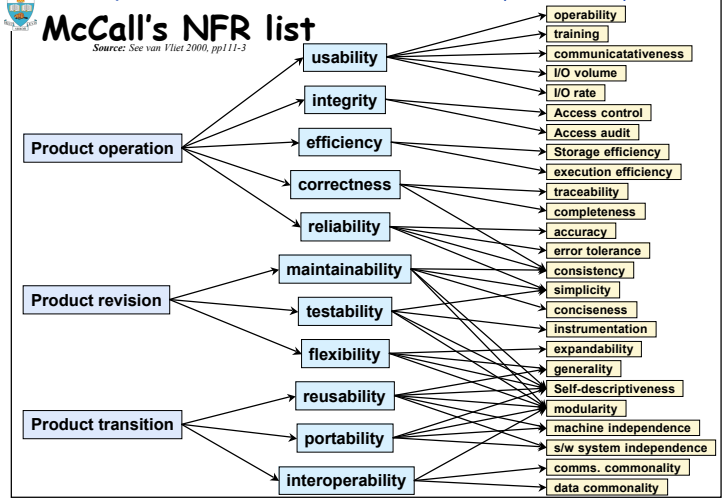
Boehm's NFR list

Source: See Blum, 1992, p176



McCall's NFR list

Source: See van Vliet 2000, pp111-3



Making Requirements Measurable

Source: Budgen, 1994, pp60-1

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

The Quality Concepts
(abstract notions of quality properties)

Measurable Quantities
(define some metrics)

Counts taken from Design Representations
(realization of the metrics)

examples...

reliability	complexity	usability
mean time to failure?	information flow between modules?	time taken to learn how to use?
run it and count crashes per hour???	count procedure calls???	minutes taken for some user task???

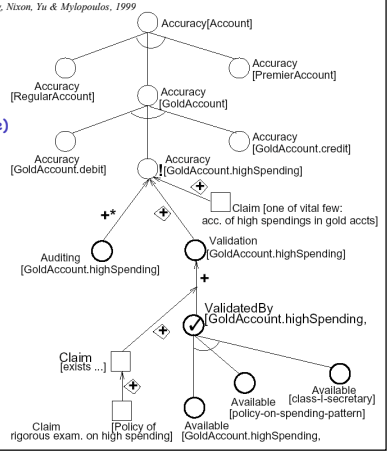
Softgoals: the NFR framework

Source: Chung, Nixon, Yu & Mylopoulos, 1999

→ Goal types:
- Non-functional Requirement
- Satisficing Technique (e.g. a design choice)
- Claim (supporting/explaining a choice)

→ Contribution Types:
- AND links (decomposition)
- OR links (alternatives)
- Sup links (supports)
- Sub links (subgoal)

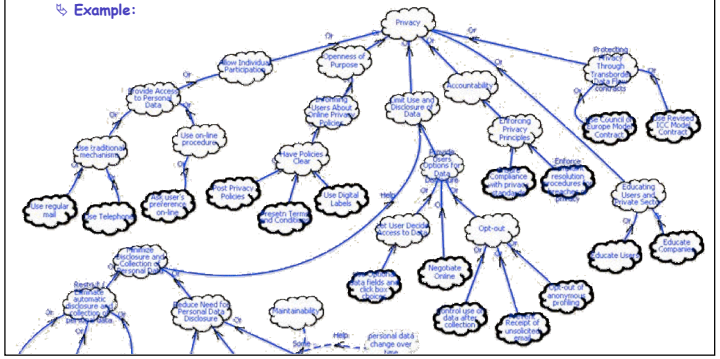
→ Evaluation of goals
- Satisficed
- Denied
- Conflicting
- Undetermined



NFR Catalogues

Source: Cysneiros & Yu, 2004

→ Predefined catalogues of NFR decomposition
- Provides a knowledge base to check coverage of an NFR
- Provides a tool for elicitation of NFRs
- Example:



Portability

→ **Definition:**
- the degree to which software running on one platform can easily be converted to run on another

→ **Considerations:**
- Portability is hard to quantify:
  - it is hard to predict on what other platforms will the software be required to run
- Portability is strongly affected by design choices:
  - E.g. choice of languages, operating systems and tools that are universally available and standardized
- Portability requirements should be given priority for systems that may have to run on different platforms in the near future.

Reliability

→ **Definition**
- the ability of the system to behave consistently in a user-acceptable manner when operating within the environment for which it was intended.

→ **Comments:**
- Reliability can be defined in terms of a percentage (say, 99.999%)
- This may have different meaning for different applications:
  - Telephone network: the entire network can fail no more than, on average, 1hr per year, but failures of individual switches can occur much more frequently
  - Patient monitoring system: the system may fail for up to 1hr/year, but in those cases doctors/nurses should be alerted of the failure. More frequent failure of individual components is not acceptable.
- Best we can do may be something like:
  - "...No more than X bugs per 10KLOC may be detected during integration and testing; no more than Y bugs per 10KLOC may remain in the system after delivery, as calculated by the Monte Carlo seeding technique of appendix Z; the system must be 100% operational 99.9% of the calendar year during its first year of operation..."

Measuring Reliability…

→ **Example reliability requirement:**
- "The software shall have no more than X bugs per thousand lines of code"
- ...But how do we measure bugs at delivery time?

→ **Use debugging**
- a number of seeded bugs are introduced to the software system, then testing is done and bugs are uncovered (seeded or otherwise)

$$\text{Number of bugs in system} = \frac{\#\text{ of seeded bugs} \times \#\text{ of detected bugs}}{\#\text{ of detected seeded bugs}}$$

- ...BUT, not all bugs are equally important!

Other Reliability Metrics

→ **How to identify suitable metrics**
- Analyze the loss incurred by software system failure,
  - eg., destruction of the panet, destruction of a city, death of some people, injury to some people, major financial loss, major embarrassment, minor financial loss.
- Different metrics are more appropriate in different situations

→ **Example metrics**
- Probability of failure on demand.
  - measures the likelihood that the system will behave in an unexpected way when some demand is made of it. This is most relevant to safety-critical systems.
- Rate of Failure Occurrence (ROCOF).
  - measures the frequency of unexpected behaviour. For example, ROCOF=2/100 means that 2 failures are likely to occur within every 100 time units.
- Mean Time to Failure (MTTF)
  - Measures average interval between failures
- Availability
  - Measures the likelihood that the system will be available for use.
  - This is a good measure for applications such as telecommunications, where the repair/restart time is significant and the loss of service is important.

Safety

- **When is safety important?**
  - Safety is a critical requirement for certain types of software systems…
    - e.g., nuclear plants, airplanes, X-ray machines,
  - …where failure may result in loss of human life.
- **Techniques for analyzing safety:**
  - Hazard analysis
    - A hazard is a condition which may cause human death or injury (a "mishap")
    - Traces from problems to hazards, or from hazards back to problems
  - Fault tree analysis
    - Creates a tree showing cause and effect of each failure
- **Risk Analysis:**
  - Probability of hazard
    - Measures how likely the hazard is to occur
  - Severity of a hazard
    - Measures the worst possible damage caused if the hazard does occur
  - Risk
    - measures the overall risk (combines probability with severity)