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

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.

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

Easterbrook
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 quality? (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?

To understand the purpose.

Fitness

Source: Budgen, 1992, p176

→ 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?

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

General utility

portability

device-independence

self-containedness

accuracy

completeness

reliability

robustness/integrity

consistency

accountability

device efficiency

accessibility

communicativeness

self-descriptiveness

structuredness

conciseness

legibility

augmentability

As-is utility

efficiency

usability

testability

understandability

modifiability

Maintainability
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

Softgoals: the NFR framework

→ 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 (necessary subgoal)

→ Evaluation of goals
  % Satisfied
  % Denied
  % Conflicting
  % Undetermined
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 bebugging
  % a number of seeded bugs are introduced to the software system, then
testing is done and bugs are uncovered (seeded or otherwise)

Number of bugs = \(\frac{\# \text{ of seeded bugs} \times \# \text{ of detected bugs}}{\# \text{ of detected seeded bugs}}\)

% …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 planet, 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.

→ 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…"