Lecture 18: Non-Functional Requirements (NFRs)

→ Definitions
  % Quality criteria; metrics
  % Example NFRs

→ Product-oriented Software Qualities
  % Making quality criteria specific
  % Catalogues of NFRs
  % Example: Reliability

→ Process-oriented Software Qualities
  % Softgoal analysis for design trade-offs

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
      % available storage space
    % 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 catastrophe, etc.

→ Operating requirements
  % Physical constraints (size, weight)
  % Personnel availability & skill level
  % Accessibility for maintenance
  % Environmental conditions
  % etc.

→ Lifecycle requirements
  % “Future-proothing”
    % 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 sequence diagrams, statecharts, etc.
  % 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: Reliability

Catalogues of NFRs

Quality criteria; metrics

Easterbrook

Survivability

Security

Reliability

Time/space

with

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

E.g. "system must handle 1,000 transactions per second"
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?
  % aesthetic value?
  % 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?

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

General utility
  As-is utility
  Maintainability
  Understandability
  Modifiability
  Legibility
  Augmentability
  Portability
  Reliability
  Robustness/integrity
  Consistency
  Accountability
  Device efficiency
  Accessibility
  Communicativeness
  Self-descriptiveness
  Structuredness
  Conciseness
  Device-independence
  Self-containedness
  Accuracy
  Completeness
  Testability
  Usability
Making Requirements Measurable

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

The Quality Concepts

- **Reliability**: The system must remain operational during the calendar year.
- **Complexity**: The number of bugs per 10KLOC.
- **Usability**: The time it takes to learn how to use it.

Measurable Quantities

- **Mean time to failure**
- **Information flow between modules**
- **Time taken for some user task**

Counts taken from Design Representations

- **Run it and count crashes per hour??**
- **Count procedure calls??**

Example: Measuring 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%)
- 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 X; 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 is it possible to measure bugs at delivery time?

→ Use bebugging
  % Measures the effectiveness of the testing process
  % 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!

Example model: Reliability growth

→ Motorola’s Zero-failure testing model
  % Predicts how much more testing is needed to establish a given reliability goal
  % basic model:
    \[ \text{failures} = b(t) \]

→ Reliability estimation process
  % Inputs needed:
    % \( f_d \) = target failure density (e.g. 0.03 failures per 1000 LOC)
    % \( t_f \) = total test failures observed so far
    % \( t_h \) = total testing hours up to the last failure
  % Calculate number of further test hours needed using:
    \[ \ln \left( \frac{f_d}{0.5 + f_d} \right) \times t_h \]
    \[ \ln \left( \frac{0.5 + f_d}{t_f + f_d} \right) \]
  % Result gives the number of further failure free hours of testing needed to establish the desired failure density
  % if a failure is detected in this time, you stop the clock and recalculate

% Note: this model ignores operational profiles!

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

Using softgoal analysis

→ 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

Source: Chung, Huang, Tsai & Holtzblatt, 1999

© 2000-2003, Steve Easterbrook
- Predefined catalogues of NFR decomposition
  - Provides a knowledge base to check coverage of an NFR
  - Provides a tool for elicitation of NFRs
  - Example: