Non Functional Requirements (NFR)

Vania Marangozova-Martin

acs.forge.imag.fr

What are Non-functional Requirements (NFR)?

- 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
  - 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
  - We'd like to state them in a way that we can measure how well they've been met

Example NFRs (1)

- Interface requirements
  - how will the new system interface with its environment?
    - User interfaces and "user-friendliness"
    - Interfaces with other systems
- Operating requirements
  - physical constraints (size, weight),
  - personnel availability & skill level
  - accessibility for maintenance
  - environmental conditions

Example NFRs (2)

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

Example NFRs (3)

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

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

Approaches to NFRs

- 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 (1)

- Think of an everyday object
  - e.g. a chair
  - How would you measure its "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!

Software Qualities (2)

- 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 (define) the purpose

Software 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?

Software Fitness

- 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

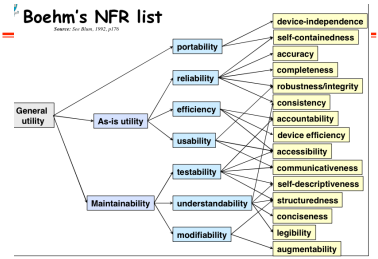
- Quality Factors
  - These are customer-related concerns
  - Examples: efficiency, integrity, reliability, correctness, survivability, usability,...
- Design Criteria
  - These are technical (development-oriented) concerns such as anomaly management, completeness, consistency, traceability, visibility,...
- Quality Factors and Design Criteria are related

Classification of NFRs

Acquisition Concern	*User Concern*	*Quality Factors*
Performance	Resource utilization security, confidence, performance under adversity, ease-of-use	efficiency integrity reliability survivability usability
Design	Conform to reqs?... easy to repair?... verified performance?	correctness maintenability verifiability
Adaptation	Easy to expand? ...upgrade function or performance? ...change? ...interface with another system? ...port? ...use in another application?	expandability flexibility interoperability portability reusability

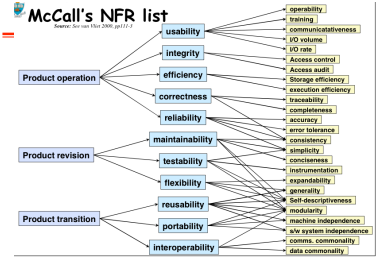
Boehm's NFR list

Source: See Blum, 1992, p178

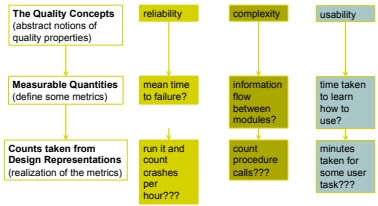


McCall's NFR list

Source: See van Vliet 2000, pp111-3



Making Requirements Measurable



NFR and AspectJ

- AspectJ gives the means for managing "cross-cutting" concerns
  - = quality concerns
- We will learn about AspectJ
  - show how the code may evolve with it
  - logging and authentication lab

References

- This lecture reuses material from
  - *© 2000-2003, Steve Easterbrook, University of Toronto*
  - *© 2004 John Mylopoulos*, University of Toronto