

University of Toronto Department of Computer Science

## 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 tradeoffs

© 2000-2003, Steve Easterbrook 1

University of Toronto Department of Computer Science

## 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
- **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

© 2000-2003, Steve Easterbrook 2

University of Toronto Department of Computer Science

## 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.

© 2000-2003, Steve Easterbrook 3

University of Toronto Department of Computer Science

## 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.

© 2000-2003, Steve Easterbrook 4

University of Toronto Department of Computer Science

## Software Qualities

- 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!
- 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

© 2000-2003, Steve Easterbrook 5

University of Toronto Department of Computer Science

## 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?
- 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?

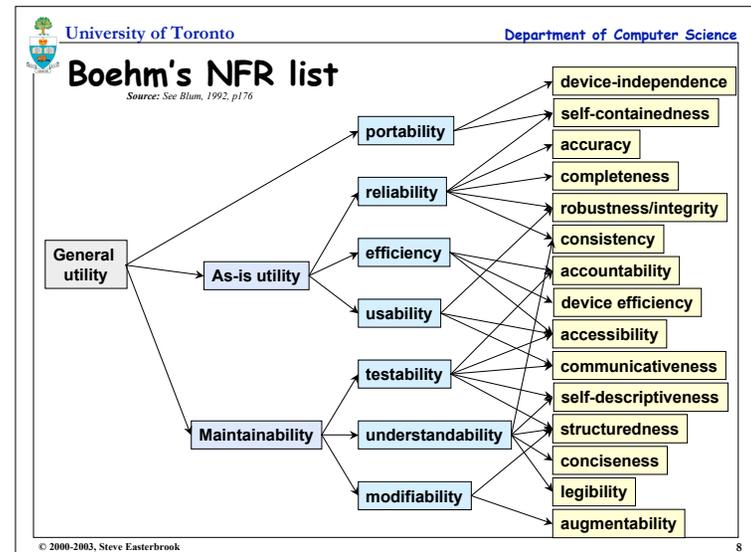
© 2000-2003, Steve Easterbrook 6

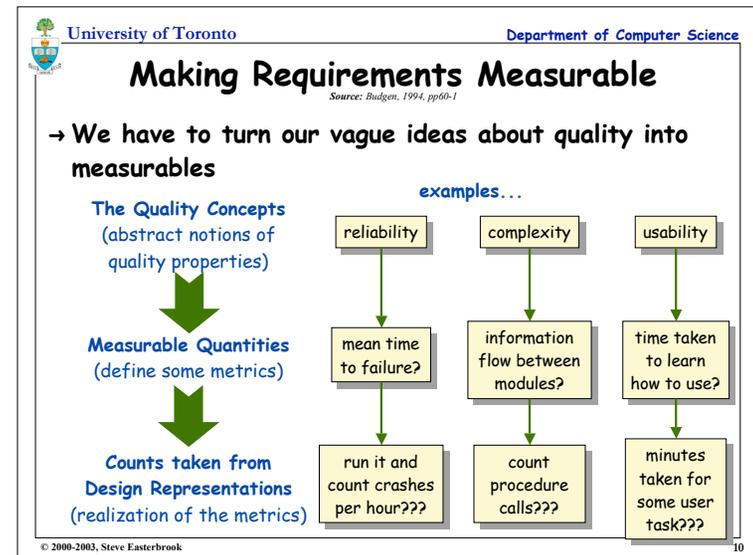
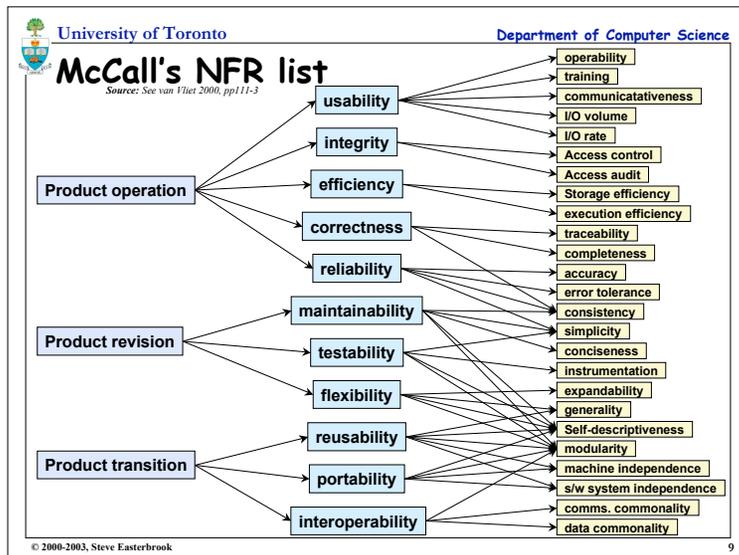
University of Toronto Department of Computer Science

## 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

© 2000-2003, Steve Easterbrook 7





University of Toronto Department of Computer Science

## Example Metrics

Quality	Metric
Speed	transactions/sec response time screen refresh time
Size	Kbytes number of RAM chips
Ease of Use	training time number of help frames
Reliability	mean-time-to-failure, probability of unavailability rate of failure, availability
Robustness	time to restart after failure percentage of events causing failure
Portability	percentage of target-dependent statements number of target systems

© 2000-2003, Steve Easterbrook 11

University of Toronto Department of Computer Science

## 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%)
- 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..."

© 2000-2003, Steve Easterbrook 12

University of Toronto Department of Computer Science

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

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

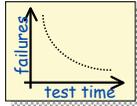
© 2000-2003, Steve Easterbrook 13

University of Toronto Department of Computer Science

## Example model: Reliability growth

Source: Adapted from Pflieger 1998, p359

- **Motorola's Zero-failure testing model**
  - ↳ Predicts how much more testing is needed to establish a given reliability goal
  - ↳ basic model:  $\text{failures} = ae^{-b(t)}$ 
    - empirical constants
    - testing time
- **Reliability estimation process**
  - ↳ Inputs needed:
    - > fd = target failure density (e.g. 0.03 failures per 1000 LOC)
    - > tf = total test failures observed so far
    - > th = total testing hours up to the last failure
  - ↳ Calculate number of further test hours needed using:
 
$$\frac{\ln(fd/(0.5 + fd)) \times th}{\ln((0.5 + fd)/(tf + fd))}$$
  - ↳ 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!



© 2000-2003, Steve Easterbrook 14

University of Toronto Department of Computer Science

## Making Requirements Measurable

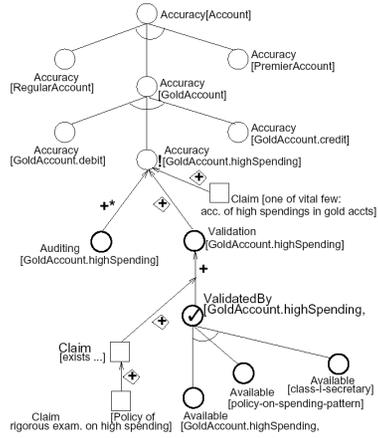
- **Define 'fit criteria' for each requirement**
  - ↳ Give the 'fit criteria' alongside the requirement
  - ↳ E.g. for new ATM software
    - > Requirement: "The software shall be intuitive and self-explanatory"
    - > Fit Criteria: "95% of existing bank customers shall be able to withdraw money and 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

© 2000-2003, Steve Easterbrook 15

University of Toronto Department of Computer Science

## 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



© 2000-2003, Steve Easterbrook Source: Chung, Nixon, Yu & Mylopoulos, 1999 16

University of Toronto Department of Computer Science

## NFR Catalogues

Source: Cysneiros & Yu, 2007

→ **Predefined catalogues of NFR decomposition**

- ↳ Provides a knowledge base to check coverage of an NFR
- ↳ Provides a tool for elicitation of NFRs
- ↳ Example:

© 2000-2006 Steve Easlerbrook 17