XII. Non-Functional Requirements (or, Quality Factors)

What are Non-Functional Requirements (NFRs)?
Classification of NFRs
Criteria and Factors
Portability, Reliability, Performance
Example NFR for an Automated Money Machine

Non-Functional Requirements (NFRs)

- Define global constraints on a software system, such as development costs, operational costs, performance, reliability, maintainability, portability, robustness etc.
- Should not be confused with **functional requirements**, which impose requirements on the function of a system
- Are generally stated informally, are often contradictory, difficult to enforce during development and to evaluate for the customer prior to delivery

How do we specify them?

Types of NFRs

- ***Interface requirements*** -- describe how the information system is to interface with its environment, users and other systems; include user interfaces and their qualities (e.g., "user-friendliness")
- ***Performance requirements*** -- describe performance constraints:
  - ✓ ***time/space bounds***, such as workloads, response time, throughput and available storage space, e.g., "system must handle 1,000 transactions per second");
  - ✓ ***reliability*** involving the availability of components and 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, who can do what;
  - ✓ ***survivability***, such as system will survive fire, natural catastrophes.
- ***Operating requirements*** -- include physical constraints (size, weight), personnel availability, skill level considerations, accessibility for maintenance, environmental conditions...

Types of NFRs

- ***Lifecycle requirements*** -- can be classified under two sub-categories:
  - ✓ Quality of the design, such as maintainability, enhanceability, portability; expected market or product lifespan,...(these don't affect initial system but may lead to increased maintenance costs or early obsolescence.)
  - ✓ Limits on development, other software lifecycle phases, such as development time limitations, resource availability, methodological standards etc.
- ***Economic requirements*** -- immediate and/or long-term costs.

(Different) Classification of NFRs

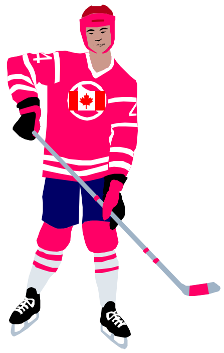
Acquisition Concern	User Concern	Quality Factors
Performance	Resource utilization security, confidence, performance under adversity, ease-of-use	efficiency integrity reliability survivability usability
Design	Conform to reqs?... easy to repair?... verified performance?	correctness maintenability verifiability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

Factors and Criteria

- ***Factors*** are customer-related concerns, such as efficiency, integrity, reliability, correctness, survivability, usability,...
- ***Criteria*** -- technical (development-oriented) concerns such as anomaly management, completeness, consistency, traceability, visibility,...
- Each factor depends on a number of associated criteria, e.g.,

  correctness depends on ***completeness***, ***consistency***, ***traceability***,...

  verifiability depends on ***modularity***, ***self-descriptiveness*** and ***simplicity***



(…A Year ago…)
Canada - USA 5-2

Factors vs Criteria

factors -- customer-related concerns
criteria -- technical concerns

Criteria	Efficiency	Integrity	Reliability	Survivability	Usability	Correctness	Verifiability	Flexibility	Portability	Reusability
	Performance					Design		Adaptation		
Performance										
Accuracy			X							
Anomaly Mngt			X	X						
Autonomy				X						
Distributedness				X						
Effectiveness storage	X									
Operability					X					
System accessibility		X								
Training					X					
Design										
Completeness						X				
Consistency						X				
Traceability						X				
Visibility							X			

Criteria	Efficiency	Integrity	Reliability	Survivability	Usability	Correctness	Verifiability	Flexibility	Portability	Reusability
	Performance					Design		Adaptation		
Adaptation										
Applic. indepen										X
Augmentability										
Commonality										
Doc. accessibility										X
Functional overlap										
Functional scope										X
Generality								X		X
Independence									X	X
System clarity										X
System compatibility										
General										
Modularity				X			X	X	X	X
Self descriptiveness							X	X	X	X
Simplicity			X				X	X		X

Quality 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*

Portability

Portability is the degree to which software running on one platform can easily be converted to run on another

- Portability is hard to quantify, because it is hard to predict on what other platforms will the software be required to run
- Portability for a given software system can be enhanced by using languages, operating systems and tools that are universally available and standardized, such as FORTRAN, COBOL or C (for languages), or such as Unix, Windows or OS/2 (operating systems).
- Portability requirements should be given priority for systems that may have to run on different platforms in the near future.

Reliability

Reliability of a software system is defined as the ability of the system to behave consistently in a user-acceptable manner when operating within the environment for which it was intended.

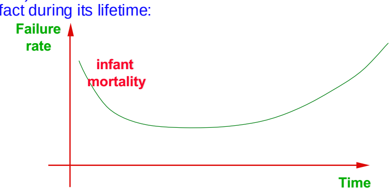
Reliability can be defined in terms of a percentage (say, 99.999%) may have different meaning for different system:

- ✓for a telephone, it might mean that the telephone should break down, on average, 1hr per year;
- ✓for a patient monitoring system, it may mean that the system may fail for up to 1hr/year, but in those cases doctors/nurses should be alerted of the failure.

Reliability: Adopting Techniques from Hardware

Theory and practice of hardware reliability are well established, some try to adopt them for software.

Most popular metric for hardware reliability is ***mean-time-to-failure*** (MTTF). "Bathtub" curve characterizes the failure rate of an artifact during its lifetime:



Reliability: Counting Bugs

- Sometimes reliability requirements take the form: "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***: just before testing, 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 = $\frac{\text{\# of seeded bugs} \times \text{\# of detected bugs}}{\text{\# of detected seeded bugs}}$

- The theoretical underpinnings of the approach are founded in Monte Carlo statistical analysis techniques for random events.

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

Reliability Metrics

Reliability requirements have to be tied to the loss incurred by software system failure, eg., destruction of mankind, destruction of a city, destruction of some people, injury to some people, major financial loss, major embarrassment, minor financial loss. Different metrics are more appropriate in different situations:

- ***Probability of failure on demand***. This 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). This 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). Discussed earlier.
- ***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.

Failure Classes

- One way to qualify reliability requirements is to characterize system failures into:
  - ✓Transient -- occur only with certain inputs
  - ✓Permanent -- occur with all inputs
  - ✓Recoverable -- system can recover with no human intervention
  - ✓Unrecoverable -- operator intervention needed for recovery;
  - ✓Non-corrupting -- failure doesn't corrupt data
  - ✓Corrupting -- failure corrupts data
- For an Automated Money Machine (AMM) example,

Failure class	Example	Reliability
Permanent	Can't read card magnetic strip	***1/100K transactions***
Transient, non-corr	Failure to read mag strip on one card	***1/10K***
Transient, corr	Cards issued by foreign bank corrupt DB	***1/20M***
Recoverable, corr	Loss of user input	***1/50K***
Recoverable, corr	Loss of mag strip data	***1/5K***

A Sample Reliability Requirement

- Combines several of the reliability metrics mentioned earlier:

"...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..."

[Musa87]

Efficiency

- ***Software efficiency*** refers to the level of use of scarce computational resources, such as CPU cycles, memory, disk space, buffers and communications channels.
- Efficiency can be characterized along a number of dimensions:
  - ✓***Capacity*** -- maximum number of users/terminals/ transactions/... the system can handle without performance degradation
  - ✓ "...The system shall handle up to and including 20 simultaneous terminals and users performing any activities without degradation of service below that defined in section X.Y.Z; other systems may make short requests, at a maximum rate of 50/hr and long requests at the rate of 1/hr..."
  - ✓***Degradation of service*** -- what happens when a system with capacity X widgets per time-unit receives X+1 widgets? We don't want the system to simply crash! Rather, we may want to stipulate that the system should handle the load, perhaps with degraded performance.

Efficiency: Timing Requirements

- Let ***stimulus*** refer to an action performed by the user/environment, ***response*** is a system-generated action.
- Four types of timing requirements [Dasarathy85]:
  - ***Stimulus-response*** -- e.g., "...the system will generate a dial tone within 2secs from the time the phone is picked up...", or "...the system will arm the alarm no sooner than 1min after the 'alarm on' button is pressed..."
  - ***Response-response*** -- e.g., "...the system will commit an ATM transaction no later than 1min after it is completed..."
  - ***Stimulus-stimulus*** -- e.g., "...the user will type her password within 15secs from typing her login name..."
  - ***Response-stimulus*** -- e.g., "...user must dial phone number within 1min from getting dial tone..."

Safety

- 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.
- Analysis of safety requirements often entails ***hazard analysis*** and ***fault trees***; these are techniques adopted from engineering disciplines.
- A hazard is a condition which may cause human death or injury (a "mishap")
- Severity of a hazard measures the worst possible damage caused by a hazard. Risk measures the probability of damage to humans.

Safety Hazards

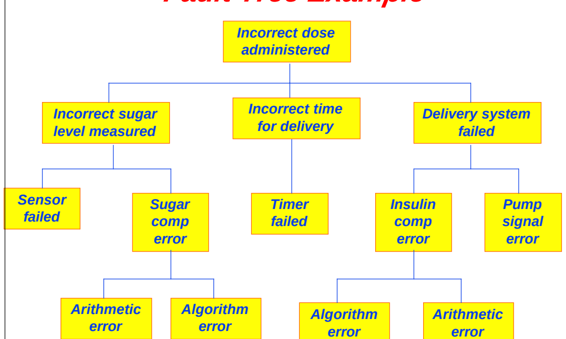
Here are some hazards for an insulin delivery machine which is attached to a patient and automatically delivers prescribed insulin doses to a patient:

Hazard	***Probability***	***Severity***	***Estimated risk***
Insulin overdose	medium	high	high
Insulin underdose	high	low	low
Power failure	high	low	low
Machine breaks off in patient	low	high	medium
Infection	medium	medium	medium
Allergic reaction	low	low	low

Types of Hazard Analysis

- ***Forward*** -- takes an initial event and traces it forward
  e.g., pipe breaks, pressure drops, pump breaks…
- ***Backward*** -- starts with a final outcome and determines the events that lead to it; e.g., insulin overdose <-- bad calculation <-- defective sugar-level sensor
- Problems with hazard analysis:
  - ✓Unrealistic assumptions, such as system built according to specs, operators are experienced and trained, testing is perfect, maintenance is perfect, key events are random and independent;
  - ✓Accident model doesn't match reality;
  - ✓Model oversimplifies reality.

Fault Tree Example



Fault Tree Evaluation

Pros

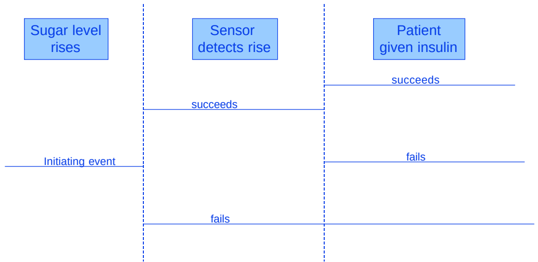
- Forces system-level analysis
- Offers an intuitive way of displaying relations between events
- Facilitates the detection of omissions

Cons

- Requires detailed understanding of the system that is being analyzed
- Some automation of the analysis is possible, but only for hardware
- Does not work for large complex systems because the fanout is huge
  e.g., can't do a fault tree with root node "airplane crashed"...

Event Tree Analysis

This is a forward search hazard analysis method
It starts with a description of event chains



Other Non-Functional Requirements

- ***Human factors*** -- building a user-friendly system requires expertise that most of us do not have; [Mantei88] describes cost-benefit tradeoffs of human factors.
- ***Testability*** (closely related to ***Understandability*** and ***Modifiability***)
  - ***cohesion*** -- measures how well the components of a module fit together
  - ***coupling*** -- measures the strength of interconnections between program units
- Requirements for testability can be given in terms of a minimum for cohesion for any one module and a minimally acceptable average for the whole system. Maximum coupling standards may also be set for any two modules or, a maximally acceptable standard might be set for the whole system

The Automated Money Machine (AMM)

- Consider the problem of building a software system which drives an Automated Money Machine (aka cash machine or bank machine.)
- The system takes as input a user transaction (e.g., deposit, withdraw, check balance,…) and sends the information to the central bank account system, receives acknowledgement that the transaction has been processed, and responds to the user (e.g., acknowledge deposit, dispense cash, give account balance,…)

■Example NFRs for the AMM

- Maintainability Requirements
  - The AMM System shall exhibit a Mean Time To Repair (MTTR) of not more than 2 hours. The MTTR is defined as the sum of the time required for fault isolation, correction, and restoration to service for each failure divided by the number of failures.
- Availability Requirements
  - The AMM System shall exhibit an availability of not less than 95 percent
- Reliability Requirements
  - The AMM System shall exhibit a system Mean Time Between Failure (MTBF) of not less than 96 hours. MTBF is defined as the quotient of the total number of operating hours divided by the total number of failures.
- Expandability Requirements

  The AMM System shall be designed in such a manner as to allow for future addition of 4 user buttons and 4 additional banking services.

■Security Requirements

- Access to account transactions shall be restricted to holders of valid banking cards and personal identification numbers.
- Cash withdrawals shall not exceed 500 dollars. Cash deposits shall not exceed $2,000.
- The AMM System shall shutdown upon detection of any device error or fatal software error.
- The AMM System shall shutdown upon loss of the link to the People's Bank Computer System.
- The AMM System shall record all transactions in its daily log.
- Developer will be responsible for ensuring the security of the physical cabinet and hardware devices.
- People's Bank will be responsible for the security of the account information contained on the People's Bank Computer System.

More Examples

- Restart Requirements

  The AMM System shall perform an automatic restart in the event of a fatal software error, to be completed within 5 minutes.

  The AMM System shall perform a cold start within 15 minutes. Cold start is defined as the process whereby the system is installed, configured, and started. Each site shall have specific configuration files which contain site specific parameters, such as site name and site address. The cold start procedure shall initialize the system from the site configuration file.
- Backup Requirements

  The AMM System has no backup requirements as the banker account information is stored on the People's Bank Computer System.
- Fallback Requirements

  The AMM System shall terminate the current transaction and shutdown in the event of a fatal device error, repeatable fatal software error, or network failure. The AMM System will not be operational again until the maintenance crew has investigated the failure.

Platform Requirements

The AMM System shall operate with not more than 4 MB RAM. 1 MB RAM shall be reserved for local data structures. 3 MB RAM shall be reserved for the operating system.

The AMM System shall operate with not more than 80 MB hard disk space. 3MB hard disk space is reserved for banking service files and configuration files.

The AMM System will execute under the Microsoft Windows Version 3.0 or later operating system. There are no Windows requirements for the human-machine interface.

The AMM System will operate on a 80386 processor or better.

Performance Requirements

The AMM System will be allocated 1.0 MB main memory to accommodate local data structures.

The AMM System will be allocated 3 MB hard disk space to accommodate any AMM banking files or configuration files.

The AMM System will respond to all banker requests in less than 10 seconds. This time shall be allocated as follows:

Banking Applications Subsystem:	0.5 seconds
Network Manager Subsystem:	0.5 seconds
People's Bank Computer System / Network:	9 seconds

Timing analysis will be performed through out the design and implementation of the subsystem to ensure that timing allocations are not being exceeded.

Additional Readings

- [Dasarathy85] Dasarathy, B., "Timing Constraints of Real-Time Systems: Constructs for Expressing Them, Methods for Evaluating Them", *IEEE Transactions on Software Engineering 11(1)*, January 1985.
- [Mantei88] Mantei, M. and Teorey, T., "Cost-Benefit for Incorporating Human Factors in the Software Lifecycle", *Communications of the ACM 31(4)*, 1988.
- [Musa87] Musa, J. et al *Software Reliability*, McGraw-Hill, 1987.
- [Thayer90] Thayer, R. and Dorfman, M., *System and Software Requirements Engineering*, IEEE Computer Society Press, 1990.
- [Roman85] Roman, G-C., "A Taxonomy of Current Issues in Requirements Engineering", *IEEE Computer*, April 1985.