XI. Non-Functional Requirements

Classification of NFRs

Criteria and Factors

Portability, Reliability, Performance

Example NFR for an Automated Money Machine

Presentation: N.C. Danh

Non-Functional Requirements -- NFRs (also Software Qualities)

- 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 wee classify them?

Types of NFRs

- ***Interface requirements*** -- how will the system interface with its environment, users and other systems; e.g., "user-friendliness”.
- ***Performance requirements***:
  - ✓ ***time/space bounds***, such as workloads, response time, throughput and available storage space, e.g., "system must handle 1,000 transactions/sec");
  - ✓ ***reliability*** -- availability of components and integrity of data, e.g., "less than 1hr downtime/yr"
  - ✓ ***security***, permissible access to data and operations;
  - ✓ ***survivability***, such as system will survive fire, natural catastrophes.
- ***Operating requirements*** -- include physical constraints (size, weight), personnel availability, skill levels...

Types of NFRs (cont'd)

- ***Lifecycle requirements*** -- can be classified under two sub-categories:
  - ✓Quality of the design, such as maintenability, 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.

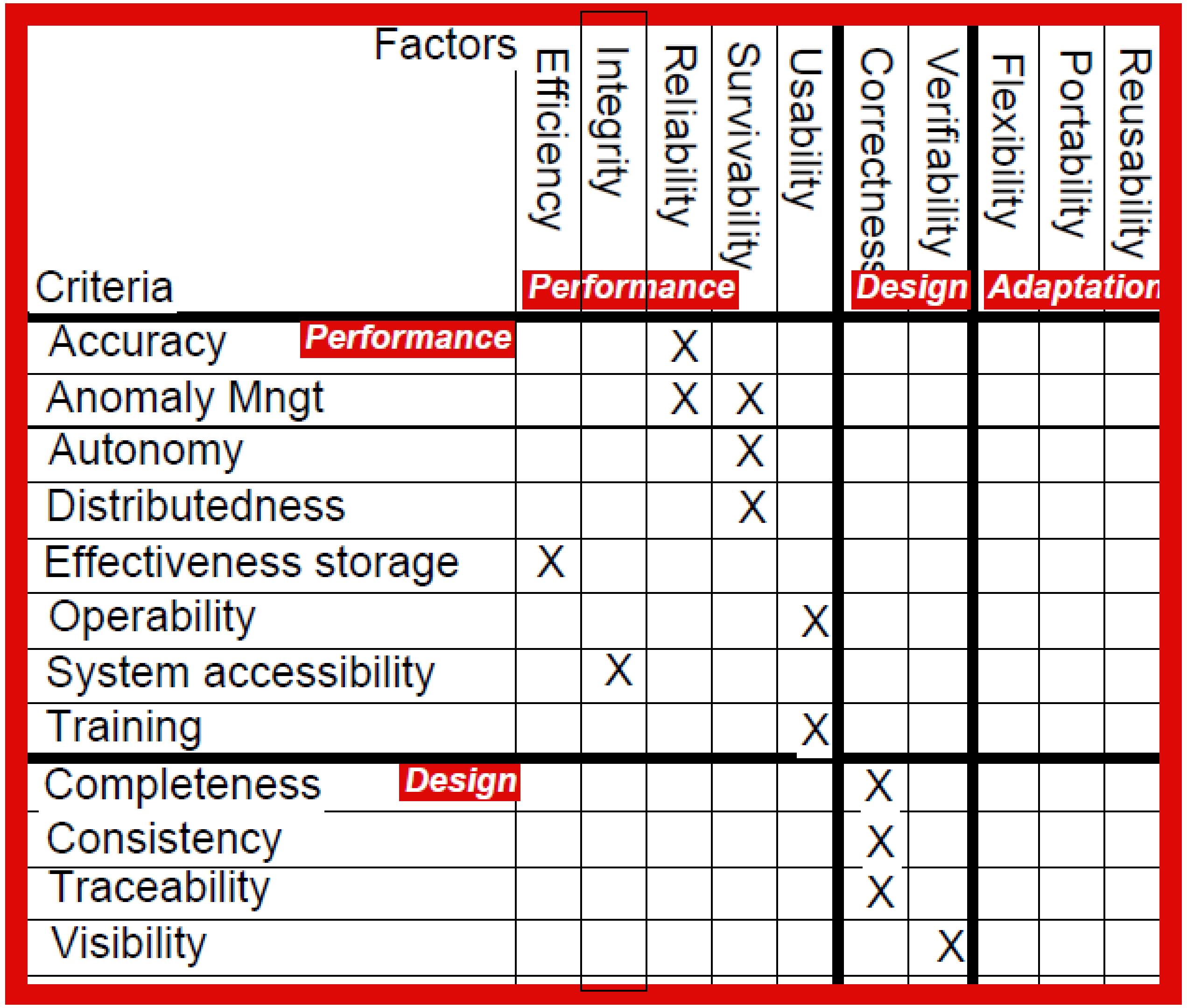
(Different) Classification of NFRs

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	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 associated criteria, e.g.,
  - ✓***Correctness*** depends on ***completeness***, ***consistency***, ***traceability***,...
  - ✓***Verifiability*** depends on ***modularity, self-description*** and ***simplicity***

Factors vs Criteria

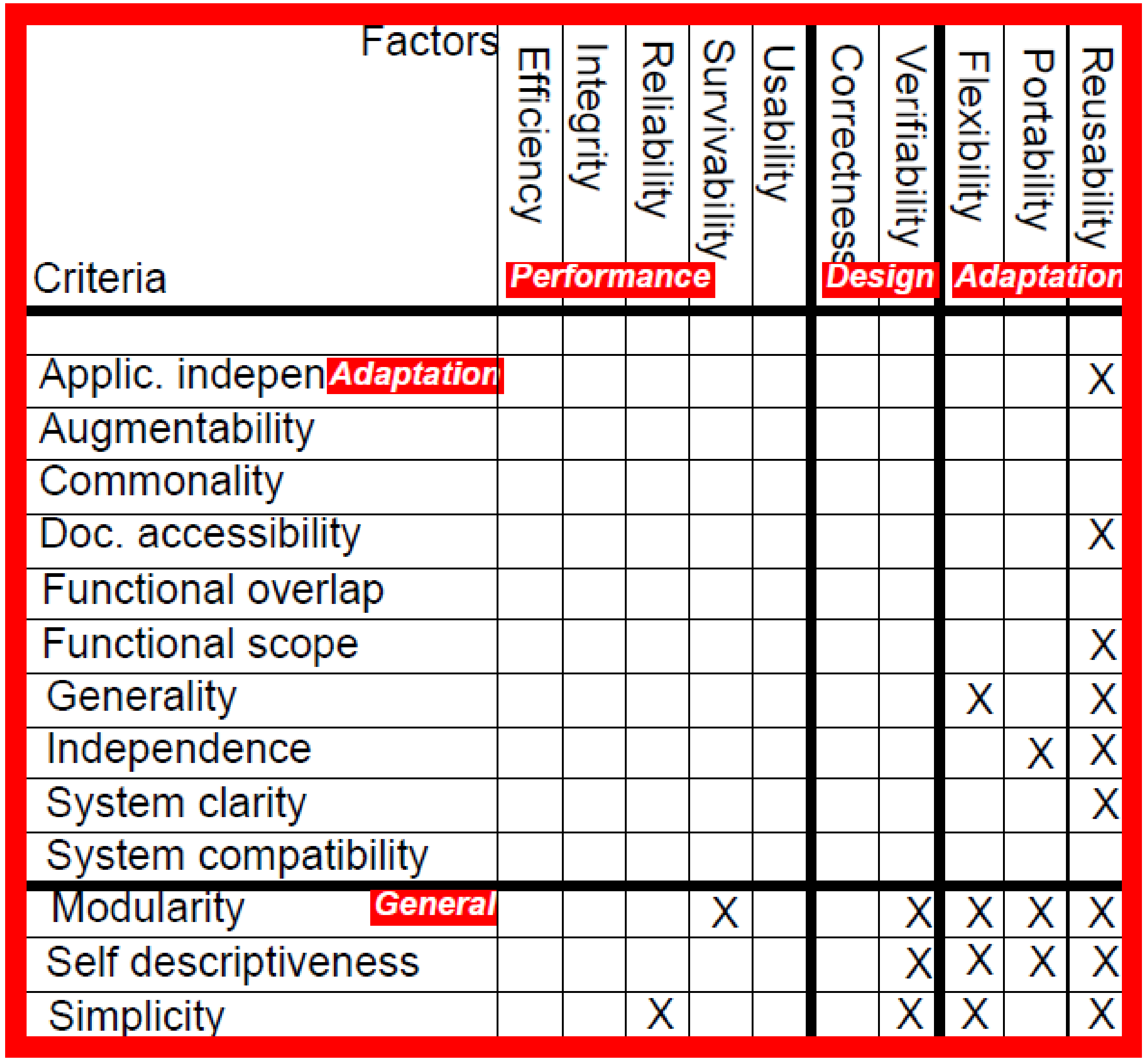


Criteria \ Factors	Efficiency	Integrity	Reliability	Survivability	Usability	Correctness	Verifiability	Flexibility	Portability	Reusability
	Performance					Design		Adaptation		
Accuracy **Performance**			X							
Anomaly Mngt			X	X						
Autonomy				X						
Distributedness				X						
Effectiveness storage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

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

(Partial table)

What Are Requirements?



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

(Partial table)

Quality Metrics

Quality	*Metric*
Speed	*transactions/sec* *response time*
Size	*KBytes* *number of RAM chips*
Ease of Use	*training time* *number of help frames*
Reliability	*mean-time-between-failures, 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.

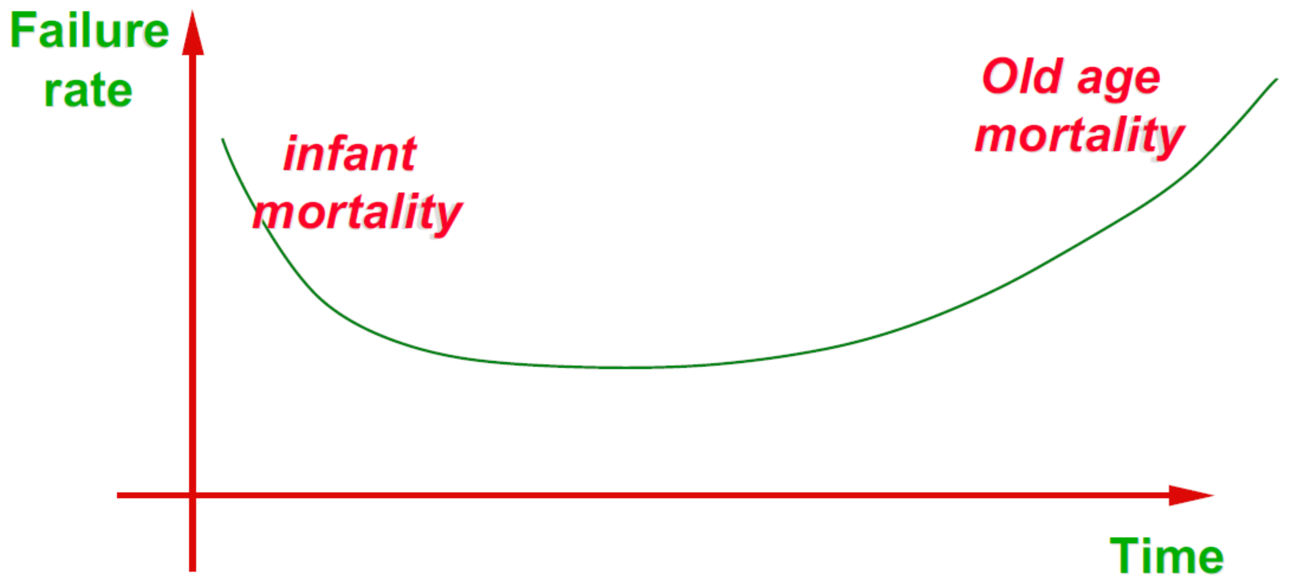
Reliability

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 an availability percentage (say, 99.999%):
  - ✓For a telephone device, it might mean that the telephone should break down, on average, <1week per year;
  - ✓For a patient monitoring system, it may mean that the system may fail <1hr/year;
  - ✓...But note the different risks involved.

Reliability: Adopting Techniques from Hardware

- Theory and practice of hardware reliability are well established, could be adopted for software.
- Most popular metric for hardware reliability is the ***mean-time-between-failures*** (MTBF).
- The "Bathtub" curve often applies:



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)

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

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

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

Reliability Metrics

- ***Probability of Failure on Demand***: measures likelihood that the system behaves in unexpected ways when some demand is made of it. Relevant to safety-critical systems.
- ***Rate of Failure Occurrence*** (ROCOF): measures frequency of unexpected behaviour; e.g., ROCOF=2/100 means that 2 failures can occur within 100 time units.
- ***Mean Time Between Failures*** (MTBF). Discussed earlier.
- ***Availability***. Measures the likelihood that the system will be available for use. Good measure for applications such as telecommunications, where the repair/restart time is significant and the loss of service is important, but not life-threatening.

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 operator intervention;
  - ✓Unrecoverable -- operator intervention needed for recovery;
  - ✓Non-corrupting -- failure doesn’t corrupt data;
  - ✓Corrupting -- failure corrupts data;

Failure Classes for an AMM

- For an Automated Money Machine (AMM) example,

Failure class	***Example***	***Fails/Ntrans***
Permanent	Can't read card mag strip	***1/100K***
Transient, non-corr	Can't read mag strip on one card	***1/10K***
Transient, corr	Cards issued by other 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 as follows:
  - ✓***Capacity*** -- maximum number of users/terminals/transactions/... the system can handle without performance degradation
  - ✓***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.

For Example

- ✓"...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...”
- ✓"...For more than 20 simultaneous terminals, the system will continue to operate with degraded services below what is defined in section X.Y.Z..."

Efficiency: Timing Requirements

- Let ***stimulus*** refer to a user-generated action, ***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...”;
  - ✓***Response-response*** -- e.g., "...system will commit an ATM transaction within 1min of completed..."
  - ✓***Stimulus-stimulus*** -- e.g., "...the user will type her password within 15secs from typing 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, Xray 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

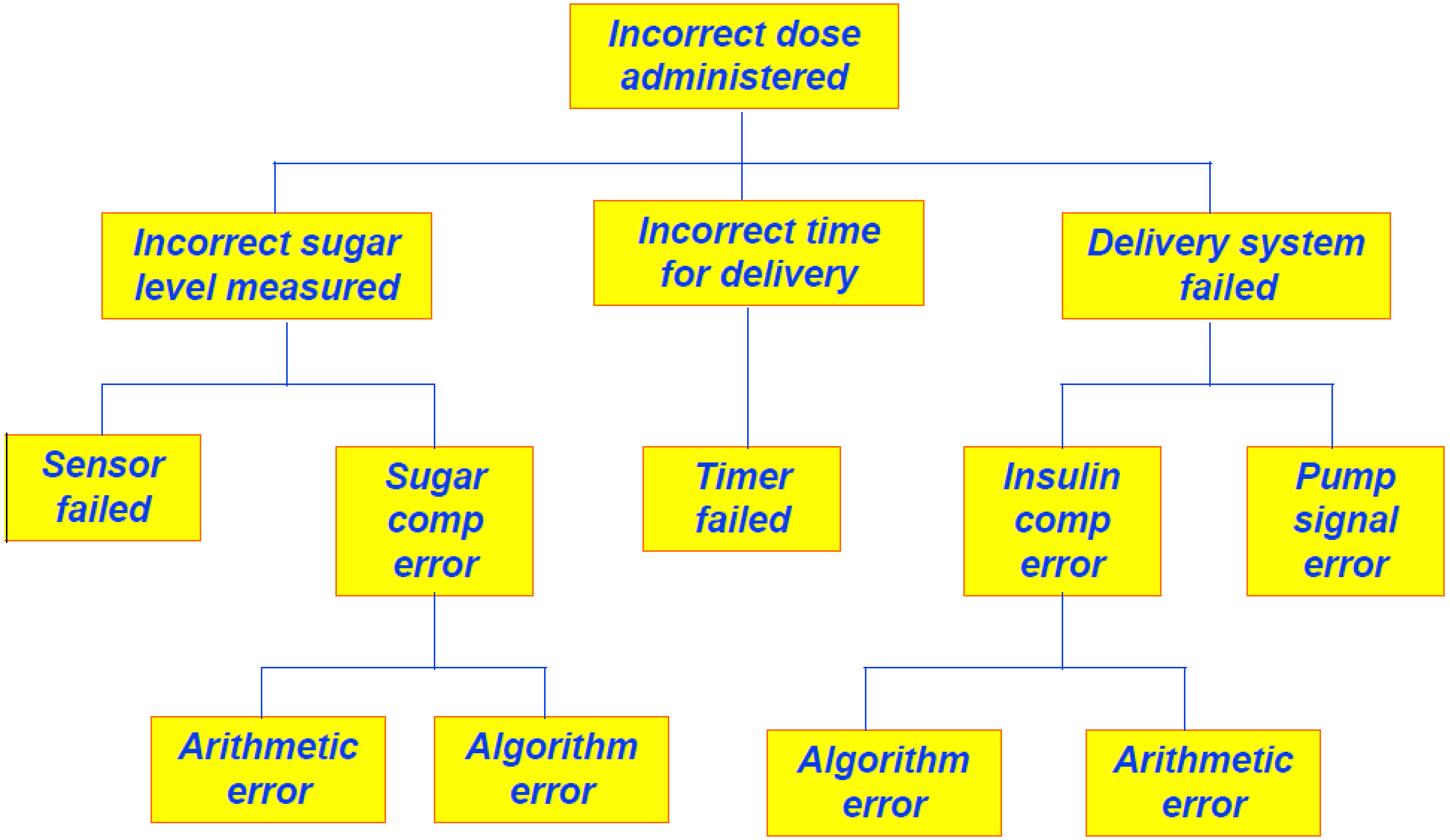
Here are some hazards for an insulin delivery machine which is attached to a patient and automatically delivers prescribed insulin doses:

Hazard	***Probability***	***Severity***	***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 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

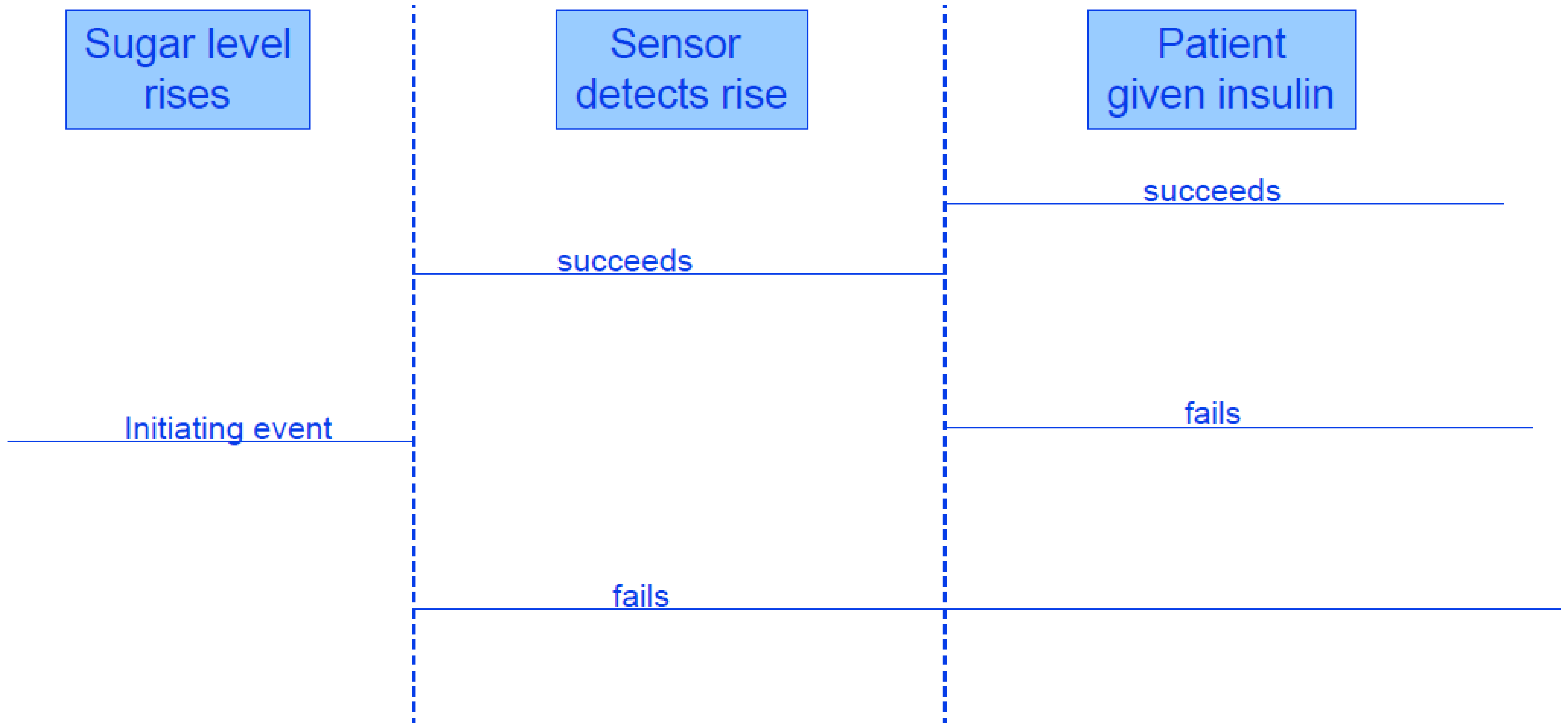
- Forces system-level analysis
- Offers intuitive way of displaying relations between events
- Facilitates the detection of omissions

Cons

- Requires detailed understanding of the system
- 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; starts with a description of event chains



Other Non-Functional Requirements

- ***Usability*** [Mantei88] -- how usability is the system-to-be.
- ***Testability*** (related to ***Understandability*** and ***Modifiability***) how easy is it to test the system; often measured in terms of:
  - ✓***cohesion*** -- how well do components within a unit fit together
  - ✓***coupling*** -- strength of interconnections between program units.
- Requirements for testability can set a minimum for cohesion for any one module and a minimally acceptable average for the whole system; Maximum coupling 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, account 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

- *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 and Expandability

- *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.
- System shall shutdown upon detection of device error, fatal software error, or upon loss of the link to DB.
- 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 is responsible for all account information contained on the Computer System.

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

More Examples

- *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 500MB RAM.
- The AMM System shall operate with not more than 80 GB hard disk space. 10GB 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 an Intel XXX processor or better.

Performance Requirements

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 sec
- Network Manager Subsystem: 0.5 sec
- People's Bank System / Network: 9 sec

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.