Reliability

- Reliability of a software system is the probability that the software system will function without failure for a “specified time” in a specified environment. [Musa,et al.], given that the system was functioning properly at the beginning of the time period.
- Reliability can also be measured/expressed as Failure Intensity and/or Failure Density.
  - **Failure Intensity** is the rate failures are happening and is typically the inverse of MTBF (Mean Time Between Failures)) e.g. 2 failures per 1000 transactions.
  - **Failure Density** is the failures per KLOC or per Function Point of developed code e.g. 2 failure per KLOC etc.
- Reliability Testing - Although faults could be removed using static code analysis, unit testing, white box testing, black box testing etc.; testing for reliability, in particular, involves creating operational profile of the software system.
- Growth models and fault injection are some of the techniques used to predict reliability.
- Thread safety, stateful data transfer and data validation are among common Reliability Assurance techniques.

Reliability Testing

- An **operational profile** is a complete set of operations with their probabilities of occurrence during the operational use of the software. Operational profiles underline how users will use the software system.
- Reliability testing thus involves repeatedly testing the **most probable use cases and inputs** to expose issues such as buffer overflow, memory leaks etc, and then measuring Failure Intensity (i.e. MTBF) etc., during this testing.
- Load testing is conducted to expose faults that wouldn’t appear if the system was lightly loaded e.g. issues with concurrency and big data.

Operational Profile – A PhotoGallery app

User role	**Probability**
Casual user	0.80
Travel Blogger	0.20

Modes	**Probability**
Not Connected	0.10
Connected to WiFi	0.30
Connected to 3G/4G	0.60

Tasks	**Probability**
Specifying the folder path in settings page	.01
Specifying the URL of the blog site	.03
Take a photo	.20
View photos in the photo gallery	.30
Upload photo to a remote site	.10
View enlarged picture	.30
Delete a picture in the gallery	.06

Reliability Models

- Reliability depends on how the software is used and therefore defining a model of usage when characterizing reliability is required.
- Reliability typically improves over time as certain bugs are fixed. Such models are referred to as reliability growth or trend models.
- Since failures may happen at random time, probabilistic models of failure could be established and used to predict reliability.
- The objective behind these models is to help determine how many bugs still stay in the software and how long will it take to identify and remove these bugs.

Reliability Models – Growth Model

Error #	Time Since Last Failure (hours)	Failure Intensity
1	4	.25
2	6	.166
3	8	.125
4	5	.20
5	6	.166
6	10	.1
7	12	.083
8	16	.0625
9	18	.055
10	25	.04

In the above example

$$\text{MTBF} = (4+6+8+5+6+10+12+16+18+25)/10 = 11 \text{ hour}$$

Given that the system has started correctly, the probability that the system is operational at time t is given as:

$$R(t) = e^{-t / \text{MTBF}}$$

Thus given that MTBF is 11hrs,

$$R(1) = e^{-1/11}.$$

The probability that the system is not operational 1 hour after the start is then

$$1 - R(1).$$

The constant ‘e’ is the base of natural log and has a value of 2.718.

Reliability Models: Fault Forecasting

- Fault forecasting [Mills 1972]: is achieved by injecting (seed) some faults in the program and thereafter estimating the remaining bugs based on how many seeded faults are detected.
- Assuming that the probability of detecting the seeded and non-seeded faults are the same:

$$N_d = N_s * (n_d/n_s)$$

where N_d, N_s, n_d, n_s are the total actual faults, total seeded faults, detected actual faults and detected seeded faults.

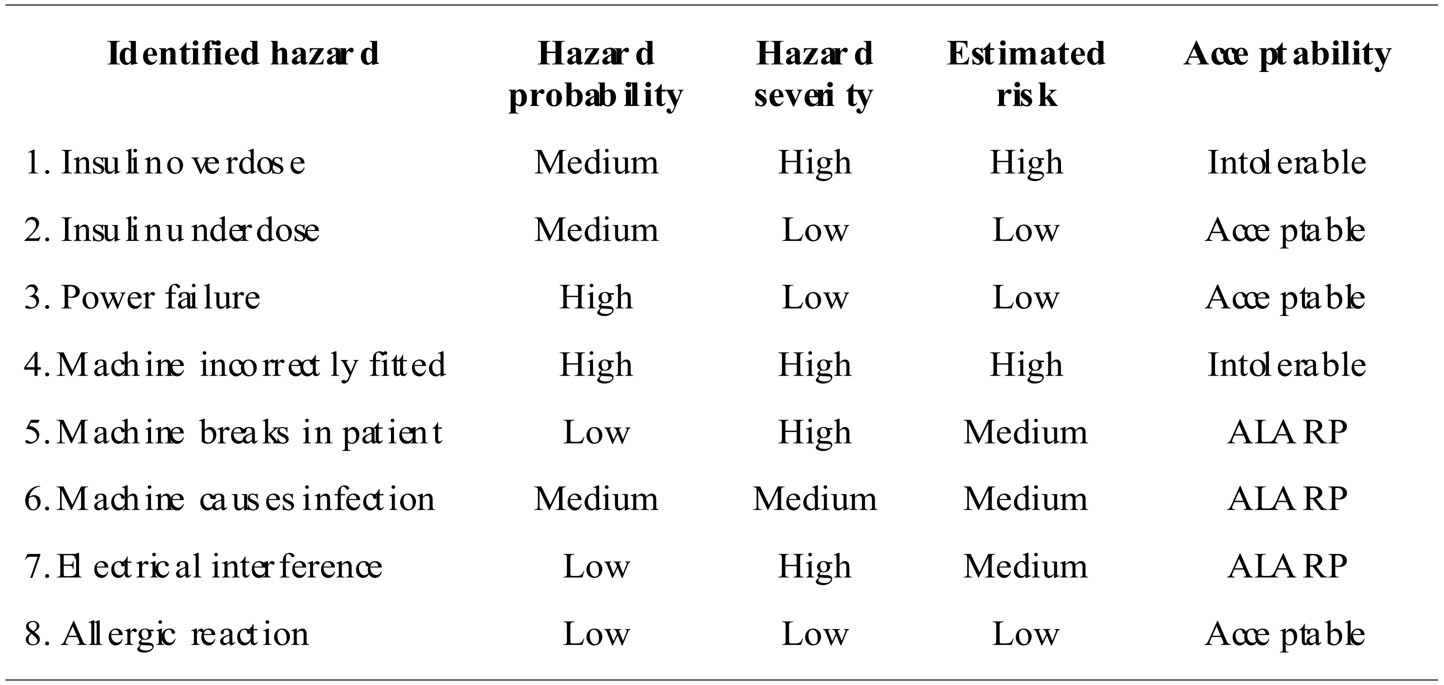
If $N_s = 20$; $n_s = 10$; and $n_d = 50$

Then $N_d = 100$.

Safety

- Safety is absence of catastrophic consequences on the users and the environment.
- Safety is an extension of reliability in the sense that it is reliability with respect to catastrophic failures.
- Analyzing safety requirements of a software system is a risk-driven process aimed at understanding the risks faced by the system as a consequence of errors and specifying requirements that reduce or alleviate these risks.
- This process typically results in a set of “excluding” requirements that define states and conditions that must not arise.
- FMEA (Failure Modes and Effects Analysis) and FTA (Fault Tree Analysis) are inductive and deductive techniques, respectively, commonly employed for analyzing safety critical systems.

FMEA (Insulin Pump)

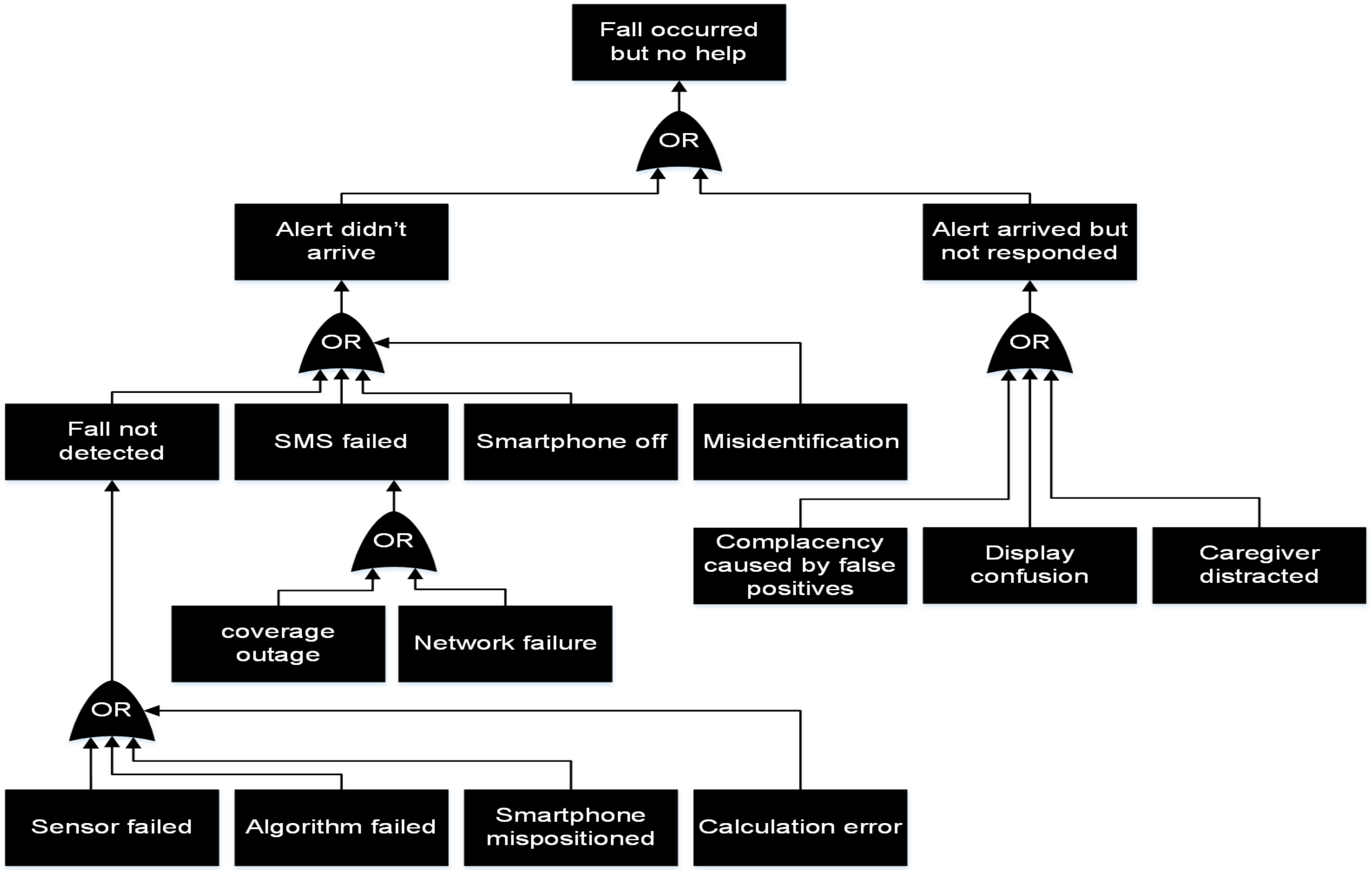


Identified hazard	Hazard probability	Hazard severity	Estimated risk	Acceptability
1. Insulin overdose	Medium	High	High	Intolerable
2. Insulin underdose	Medium	Low	Low	Acceptable
3. Power failure	High	Low	Low	Acceptable
4. Machine incorrectly fitted	High	High	High	Intolerable
5. Machine breaks in patient	Low	High	Medium	ALARP
6. Machine causes infection	Medium	Medium	Medium	ALARP
7. Electrical interference	Low	High	Medium	ALARP
8. Allergic reaction	Low	Low	Low	Acceptable

FMEA (Fall Detection & Emergency Alert)

Process Step	Failure Modes	Causes	Effects	O	S	
User fell but no help arrived	Misidentification	Emergency contact not correctly entered	Long term harm to the user	Low	High	H
		Wrong emergency contact entered		Low	High	H
		User ID not entered		Low	High	H
		Wrong User ID entered		Low	High	H
		Duplicate User IDs		Low	High	H
	Hardware, Software or Network Breakdown	Sensor data processing error	Long term harm to the user	Medium	High	M
		Algorithmic error		Medium	High	M
		Wireless coverage unavailable		Medium	High	L
		SMS software, protocol or network infrastructure failed		Low	High	L
		Network interface card failed		Low	High	L
		Sensor malfunction		Low	High	L
		Battery outage		High	High	H
		Smartphone crashed		Medium	High	L
	Usability	Smartphone wrongly placed or oriented	Long term harm to the user	High	High	M
		User forgot to carry smartphone		High	High	L
		Onboard SMS app confused the care giver		Medium	Medium	M
		Unclear User Identity		Medium	Medium	M
	Alert Ignored	Complacency resulting from oversensitivity causing false positives across the system or few select smartphones	Likely harm to the user	Medium	Medium	M

FTA


Fall occurred but no help
OR
Alert didn't arrive
Alert arrived but not responded
OR
OR
Fall not detected
SMS failed
Smartphone off
Misidentification
OR
coverage outage
Network failure
Complacency caused by false positives
Display confusion
Caregiver distracted
OR
Sensor failed
Algorithm failed
Smartphone mispositioned
Calculation error

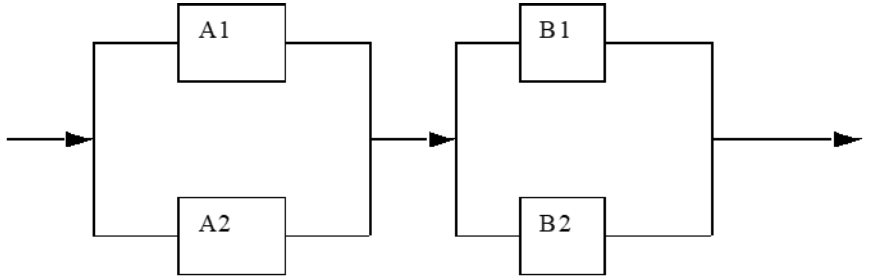
Availability

- Availability is the probability that a system is operational at any time.
- If reliability improves by eliminating faults or masking them effectively then availability improves by introducing redundancy and incorporating fault tolerance.
- Availability = MTBF / (MTTR + MTBF), where MTTR is the Mean Time To Recover and MTBF is Mean Time Between Failures.
- A system with N components connected in tandem will have overall availability of Π (A_n), where A_n, given by MTTR / (MTTR + MTBF), is the availability of the individual component. On the other hand, a system with M components, all connected in parallel, will render an overall availability of [1 - Π (1 - A_n)].
- Given that a component's availability is less than or equal to 1, these statistical availability models obviate that the availability of the overall system will improve if more redundancy is introduced i.e. more components are added in parallel and will deteriorate, if on the other hand, the number of components connected in tandem increases.

Consider a distributed systems architecture depicted below. Any incoming request is equally likely to traverse any of the four possible paths i.e. A1B1, A1B2, A2B1 and A2B2. The availability of the overall system in which A1 & A2 as well as B1 and B2 are pairs of replicated components introduced to induce redundancy is therefore [1 – ((1-A1)(1-A2))] x [1 – ((1-B1)(1-B2))]. Given that A1 = A2 = B1 = B2 = 0.90, the overall availability is 0.98.

[1 – ((1-0.9)x(1-0.9))] x [1 – (1-0.9)x(1-0.9)] = .99 x .99 = .98

0.9 x 0.9 = .81



Availability Testing

- Availability is verified and validated via stress testing that involves creating conditions e.g. resource starvation or forced shutdown of components causing the system to fail to determine how graceful, fast and correct the recovery from the failure was