Industrial Embedded Systems
- Design for Harsh Environment -

Dr. Alexander Walsch
alexander.walsch@ge.com

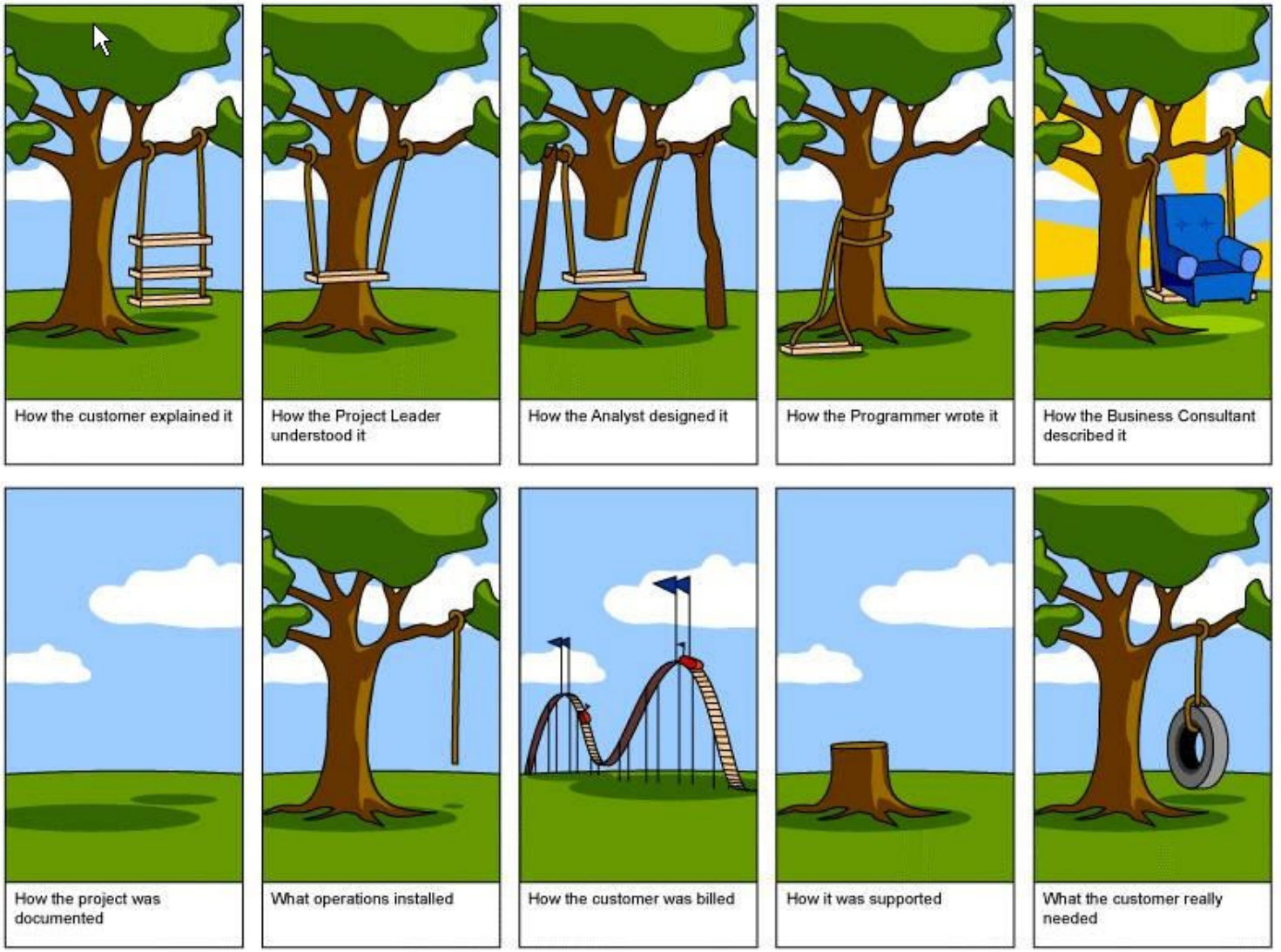
IN2244

Part II

WS 2013/14

Technische Universität München

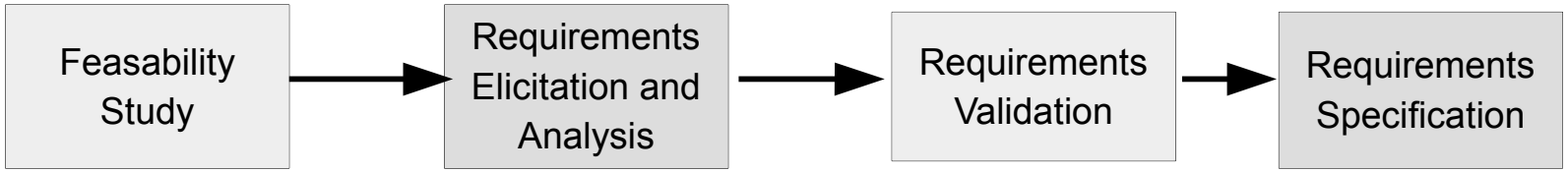
Motivation
- What the Customer really needed -



Source: http://www.bowdoin.edu/~disrael/what-the-customer-really-needed/what-the-customer-really-needed.jpg

Requirements Engineering

- Requirements are features of a system or system function used to fulfill the system purpose.



Requirements Engineering II

- The requirements elicitation and analysis phase of embedded system development is about:
  - Getting all system functions together
  - Showing scope, usage, and constraints (performance, environment, regulation, threats, etc.) of the proposed system
  - Get a good understanding on effort and system architecture (risk reduction)
- Wrong (e.g. missing, contradicting) information will make us fail at a very cost intensive level → validation
- Once all information are available and validated the requirements are translated into a requirements specification which is a technical document for further development (metrics and defined format on all requirements)

Requirements Elicitation and Analysis

How do we get all these requirements?

- Involves technical staff working with customers or users to find out about the application domain (field technicians), the services that the system should provide and the system's operational constraints.
- May involve end-users, our customers, managers, engineers involved in prior development and/or maintenance, domain experts, certification bodies, etc. These are called stakeholders.
- Also non-functional requirements can be discovered in a systematic way (QFD, FTA, RBD, PHA, ...)

Challenges in Requirements Analysis

- Stakeholders don't know what they really want.
- Stakeholders express requirements in their own terminology – maybe not precise.
- Different stakeholders may have conflicting requirements.
- Political factors may influence the system requirements (e.g. disasters).
- The requirements change during the analysis process.
- Some requirements might be common sense and not explicitly mentioned.

Requirements Validation

- Validity
  Does the system provide the functions which the customer expects?
- Consistency
  Are there any requirements conflicts?
- Completeness
  Are all functions required by the customer included? Are more functions included?
- Realism
  Can the requirements be implemented given available budget and technology -> feasibility?
- Verifiability
  Can the requirements be tested?

Traceability

Traceability

Traceability is concerned with the relationships between requirements, their sources and their design implications. Traceability can be a requirement itself.

- Source traceability
  - Links from requirements to stakeholders who proposed these requirements
- Requirements traceability
  - Links between dependent requirements
- Design traceability
  - Links from the requirements to the design

Requirements Specification Structure

Typical document layout:

Requirement Specification
1. Objective
2. System Description (boundary, interfaces, major components)
3. Functional Requirements
4. Non-functional Requirements
5. Mechanical Constraints
6. Environmental Constraints
7. RAMS (safety in a separate document)

All requirements get numbers which allow forward and backwards tracing.

Non-functional Requirements

- There are basically two kinds of requirements:
  - Non-functional (quality)
  - Functional (operations – IO)
- We will look into tools that help to gather requirements for
  - Safety: hazard analysis, fault trees (FTA), risk assessment (quality)
  - Reliability: (failure mode and effect) FMEA, FTA
- There are more non-functional requirements which will not be covered.

Non-Functional Requirements

Non-Functional Requirement

Constraints on implementation – How should the system be?

Includes

- Global constraints that influence system as a whole (shock, vibration, temperature, cost…)
- Function performance (response time, repeatability, utilization, accuracy)
- The “-ilities” (reliability, availability, safety, security, maintainability, testability, …)
- Other quality (ease of configuration and installation, …)

Non-Functional Requirements Capture

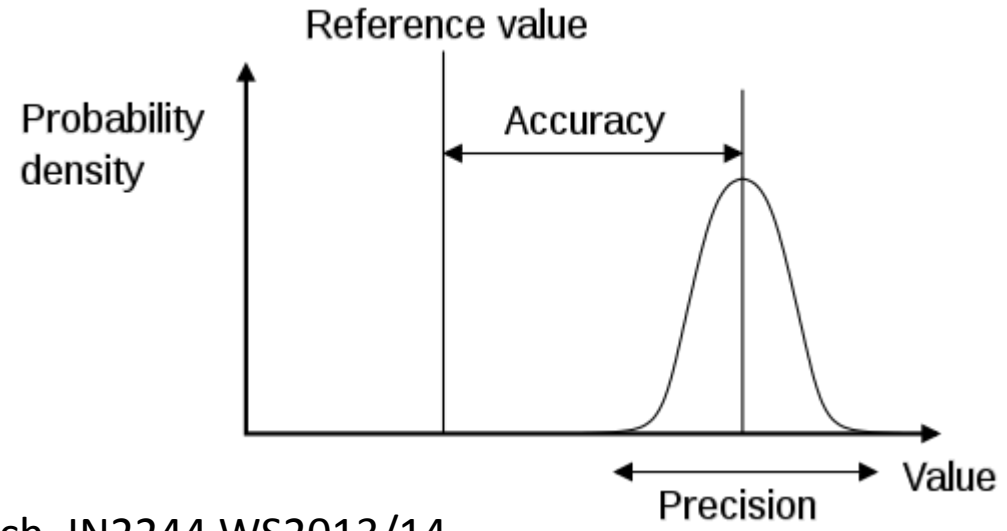
Look at system as black-box and concentrate on a specific use case

- Look at real-time aspects
  response time, sampling rate
- Data quality
  accuracy, precision
- Refine functional requirements – make more specific and testable
- Look at comparable systems (prior art, competitors)
- Safety (new laws or regulation) and reliability
- Hardware constraints (memory, CPU, IO)

Non-Functional Requirements
- Textual Examples -

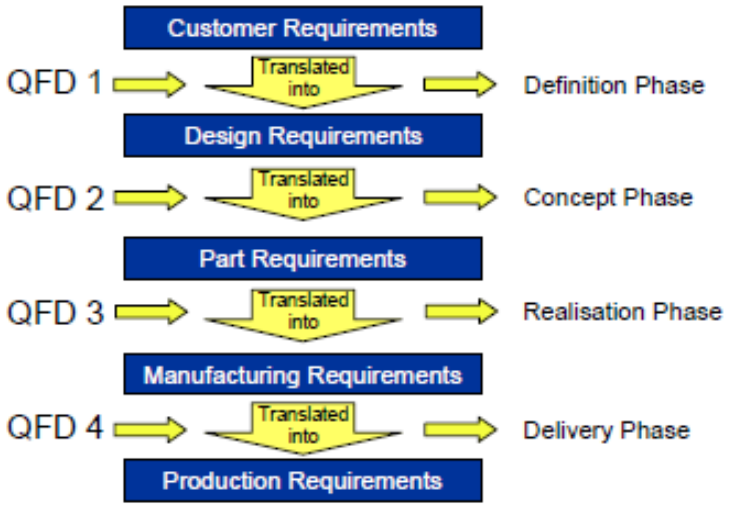
“Pressure samples shall be taken every 1s.”
“The response time for pressure measurement shall be less than 10ms.”
“Reliability: 1000 FIT”
“The measurement shall have an accuracy of 2%.”
“The measurement shall be repeatable with a precision not less than 0.5%.”
“The system shall meet the safety criteria according to [std.].”

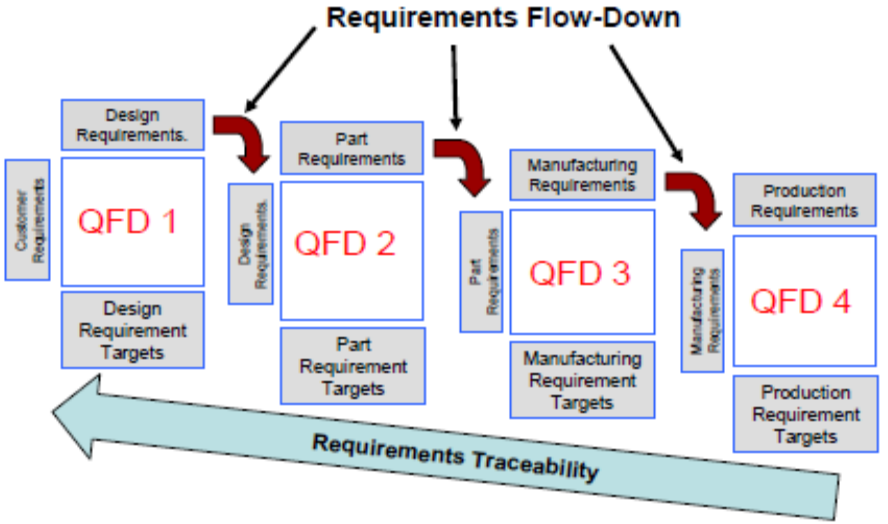
Source: wikipedia



Quality Function Deployment (QFD)

- QFD is a systematic way to correlate the relationship between customer requirements and technical requirements.
- QFD is based on a sequence of matrix charts.
- QFD has been introduced for quality planning in manufacturing (Akao 1960s) but is a general methodology that can also be applied to computer system design.





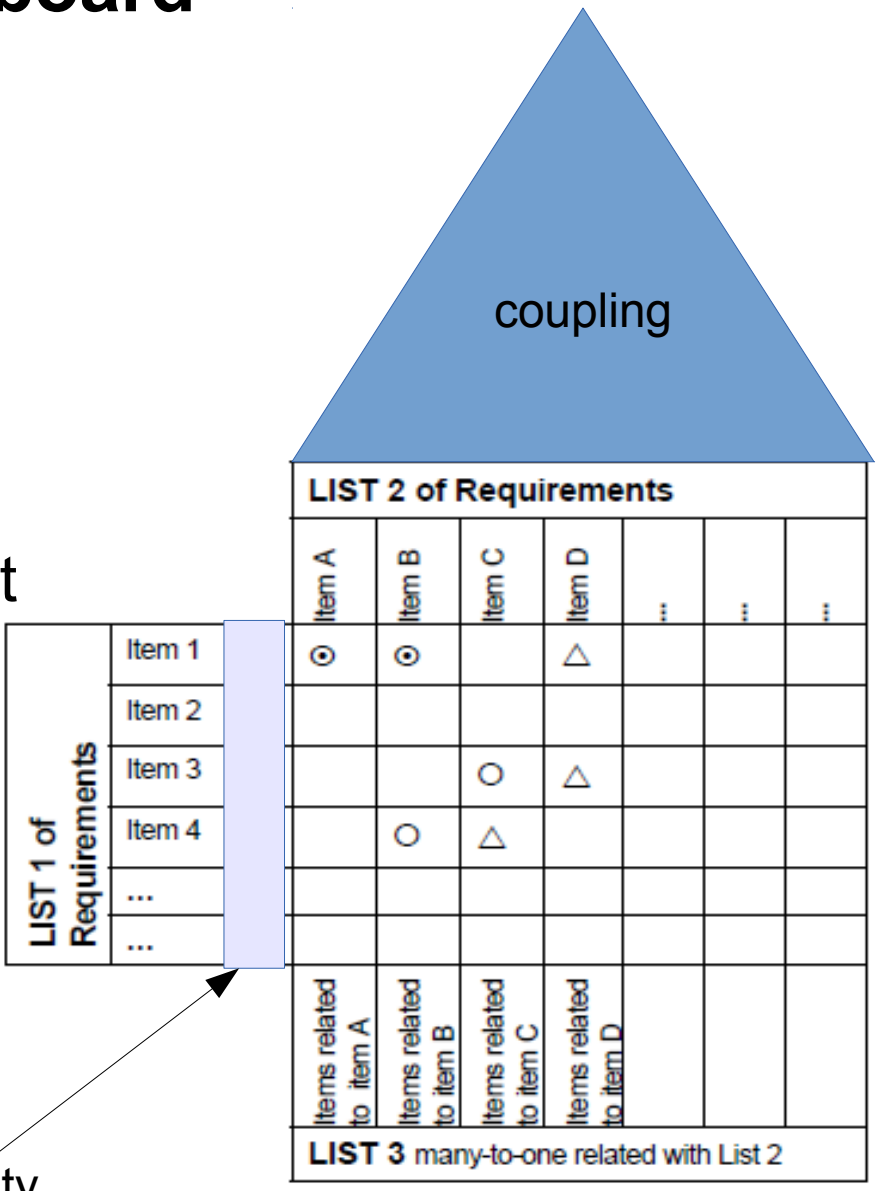
Source: Borge, A functional approach to QFD

QFD for Software

- QFD needs to be adjusted to reflect embedded software development.
- Customer requirements – requirements as received from customer
- Design requirements → Software technical requirements (functions)
- Part requirements → Architectural requirements (coarse design, larger entities, look at cohesion and coupling)
- Manufacturing requirements → Detailed Design (functions, classes, algorithm, data)
- Production requirements → Implementation (coding details, code quality)

QFD Example
- See Whiteboard -

- Development of a smart meter
- Customer requirements:
  - Inexpensive
  - Secure and reliable
  - Measures voltage and current
  - Wireless comms
  - Powerline comms
  - Display



⊙ indicates a strong relationship
○ indicates a medium
△ indicates a weak relationship

Source: Borge, A functional approach to QFD

Fault, Error, Failure

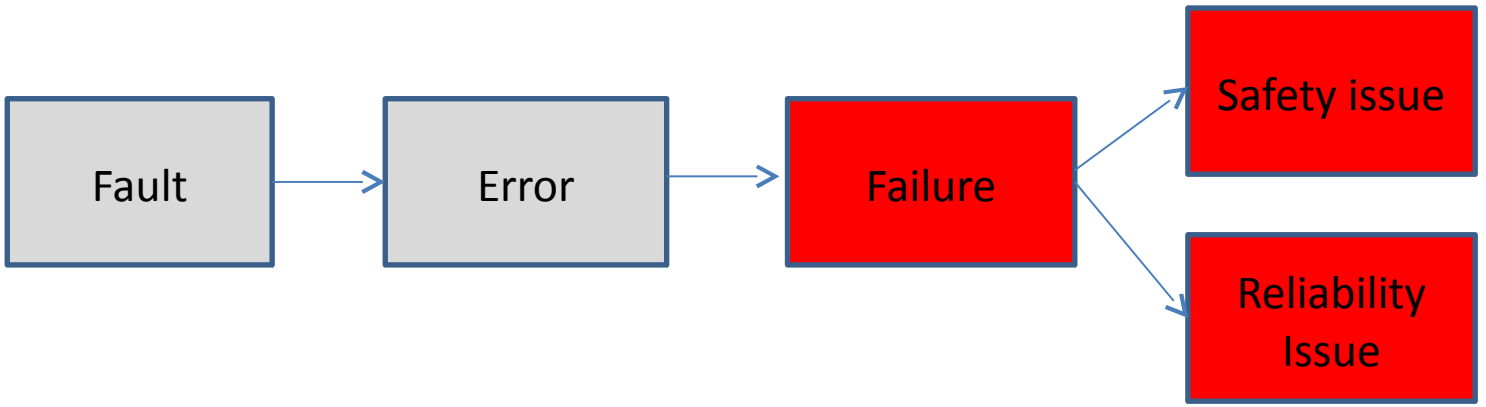
Fault (HW), Defect, Bug (SW)
abnormal condition that may cause a reduction in, or loss of, the capability of a functional unit to perform a required function

Error (revealed fault)
a deviation from the correct value or state

Failure
Failure is defined as deviation from the specification. The designed function can not be executed anymore as specified.

Failure Mode
A function can fail in various ways. In our analysis we pick the failure mode that leads to the failure we investigate.

Fault, Error, Failure II



- Hardware faults can be random or systematic. Software defects are systematic
- Hardware faults can be thought of as physical faults, e.g. a bit flips, a wire breaks. Software defects are mistakes during development
- Faults and defects are dormant until the resource is used (think of a software task that executes specific code for the first time)
- Once it is used it may cause an error which is a deviation from the expected
- The error may make the system deviate from its specification. It is running outside its intended use

Failure Modes

Function:
A process variable is measured (input) and the temperature compensated reading transmitted using a 4 – 20 mA data communication interface (output).

The following failure modes and occurrences are known. What failure modes do influence our design most?

Failure Mode	Failure occurance
4 – 20 mA current signal stuck fail (output)	Low
4 – 20 mA current signal low fail (output)	Low
Sensor head fail (input)	Medium
Power failure	High
Other	low

Failure Modes and Effect Analysis (FMEA)

- System FMEA in requirements analysis (proposed system)
  - Also: Design FMEA (existing system)
- What are the failure modes and what is the effect:
  - System failure (e.g. power, communication, timeliness, erroneous) mode assessment
  - Plan how to prevent the failures
- How does it work?
  - Identify potential failure modes and rate the severity (team activity)
  - Evaluate objectively the probability of occurrence of causes and the ability to detect the cause when it occurs
  - Rank failure modes and isolate the most critical ones

FMEA II

- FMEA tools
  - Spreadsheet, proprietary (e.g. Reliasoft Xfmea)
- Risk ratings: 1 (best) to 10 (worst)
  - Severity (SEV) – how significant is the impact
  - Occurance (OCC) – likelihood of occurance
  - Detection (DET) – how likely will the current system detect the failure mode
- Risk Priority Number (RPN)
  - A numerical calculation of the relative risk of a particular failure mode
  - RPN = SEV x OCC x DET
  - Used to isolate the most risky functions and their failure modes
  - Qualtitative approach (risk ratings are relative numbers)

FMEA III

- Function – What is the system going to do?
- Failure – How could the function fail?
- Effect – What could be the outcome of the failure?
- Cause – What could be the cause of the failure?

Function	Failure	Effect	Si	Cause	Oi	Control	Control Type	Di	RPNi
Function 1	Failure mode 1	Effect 1	2	Cause 1	9	Detection 1	Detection	6	108
	Failure mode 2	Effect 2	8	Cause 2	2	Detection 2	Detection	6	96
	Failure mode 3	Effect 3	1	Cause 3	3	Detection 3	Detection	6	18
Function2	Failure mode 1	Effect 1	6	Cause 1	7	Detection 1	Detection	6	252
	Failure mode 2	Effect 2	1	Cause 2	2	Detection 2	Detection	6	12

FMEA Example
- See Whiteboard -

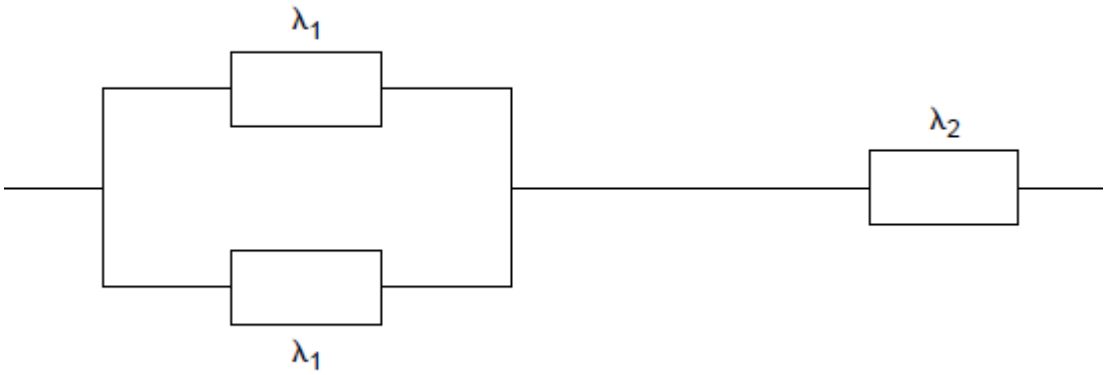
- We will take the software technical specification from QFD and derive possible failures, causes and detection mechanisms.
- The intent here is to specify additional non-functional software requirements.
- When thinking about software failures consider this:

Quality	Description of Quality
Accuracy	The term *accuracy* denotes the degree of freedom from error of sensor and operator input, the degree of exactness possessed by an approximation or measurement, and the degree of freedom of actuator output from error.
Capacity	The terms *capacity* denotes the ability of the software system to achieve its objectives within the hardware constraints imposed by the computing system being used. The main factors of capacity are Execution Capacity (timing) and Storage Capacity (sizing). These refer, respectively, to the availability of sufficient processing time and memory resources to satisfy the software requirements.
Functionality	The term *functionality* denotes the operations which must be carried out by the software. Functions generally transform input information into output information in order to affect the reactor operation. Inputs may be obtained from sensors, operators, other equipment or other software as appropriate. Outputs may be directed to actuators, operators, other equipment or other software as appropriate.
Reliability	The term *reliability* denotes the degree to which a software system or component operates without failure. This definition does not consider the consequences of failure, only the existence of failure. Reliability requirements may be derived from the general system reliability requirements by imposing reliability requirements on the software components of the application system which are sufficient to meet the overall system reliability requirements.
Robustness	The term *robustness* denotes the ability of a software system or component to function correctly in the presence of invalid inputs or stressful environmental conditions. This includes the ability to function correctly despite some violation of the assumptions in its specification.
Safety	The term *safety* is used here to denote those properties and characteristics of the software system that directly affect or interact with system safety considerations. The other qualities discussed in this table are important contributors to the overall safety of the software-controlled protection system, but are primarily concerned with the internal operation of the software. This quality is primarily concerned with the affect of the software on system hazards and the measures taken to control those hazards.
Security	The term *security* denotes the ability to prevent unauthorized, undesired and unsafe intrusions. Security is a safety concern in so far as such intrusions can affect the safety-related functions of the software.

Source:
Software Safety Hazard Analysis, J. Lawrence, LBLL

Reliability Block Diagram (RBD)

- We need two things to compare different architectures (in EE):
  - A probabilistic model – probability law
  - A notation – Reliability Block Diagram (RBD) which assume probabilistic independent blocks
  - Each block has a defined function, a failure mode with a failure rate
  - A system function can be spread across different blocks (think of blocks as components)



Source:
Smith: Reliability, Maintainability and Risk

RBD Example
- See Whiteboard -

Fault Tree Analysis (FTA)

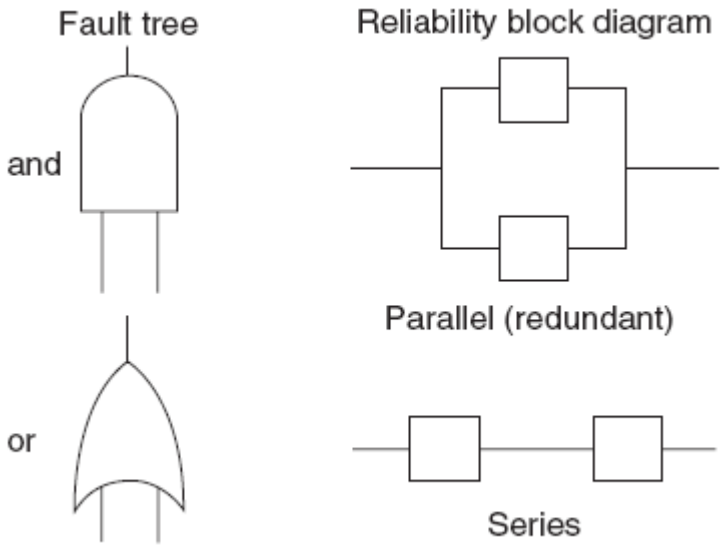
- Top event is failure mode (system or function)
- Devide system functions into sub-functions (functional decomposition) or system into components (component decomposition)
- Look into combinations of faults (strength of FTA)
- Tree like structure using combinatorical logic
- Paths of Failure

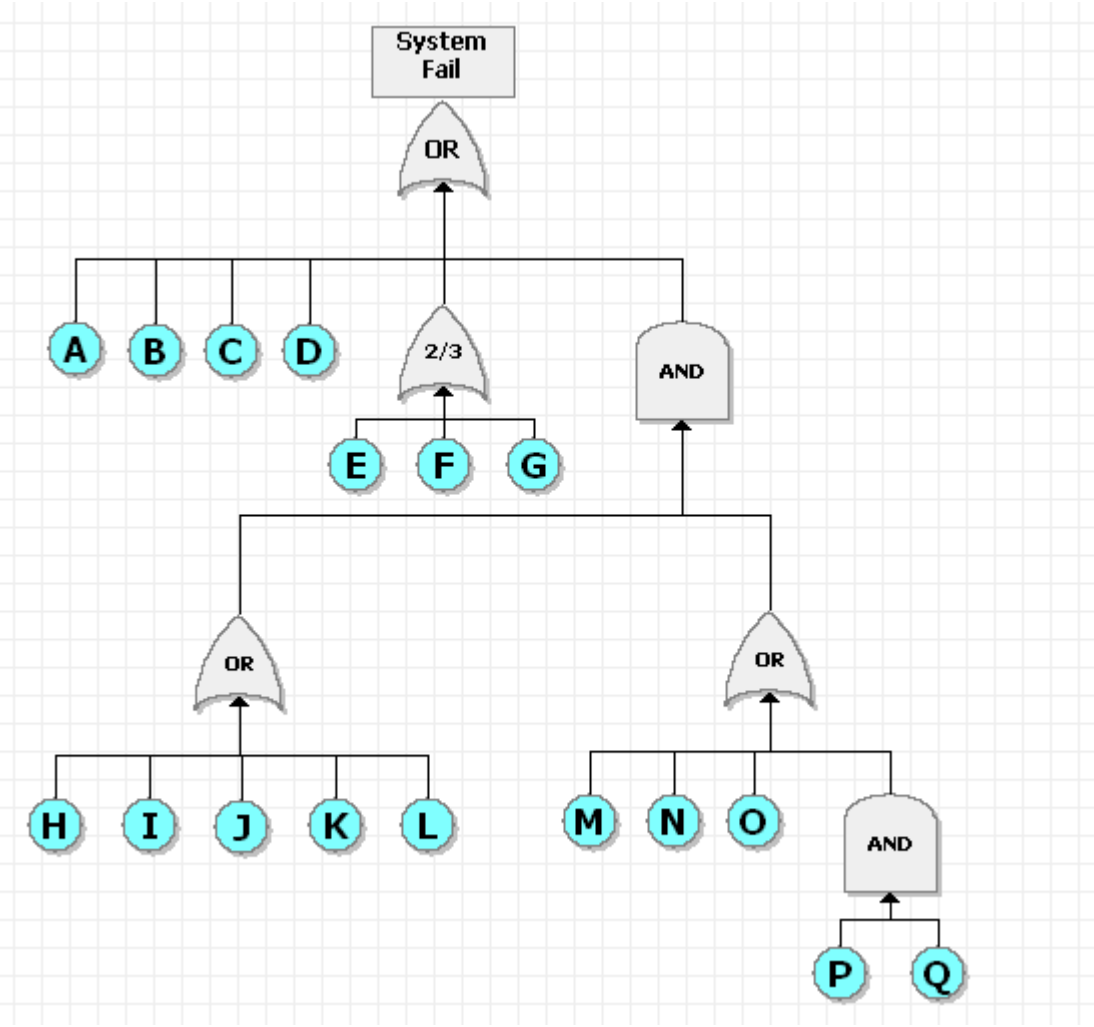
Outcome:

- Root cause event (external, internal) that (in combination) will lead to top event → failure modes of sub-functions or components
- Good system understanding – very useful if applied to existing systems to isolate reliability issues

FTA II

- FTA is semantically equivalent to Reliability Block Diagram (RBD)





Source:
Smith, Functional Safety

FTA Example
- See Whiteboard -

Where are we?

- We know that there are critical requirements that influence our proposed system
- Criticalilty can be derived from FMEAs and FTAs (there are other methods as well)
- Criticality can be quantified such that
  - Architecturural or technology decisions can be made.
- Concepts are derived from EE hardware engineering. A discipline more mature than software engineering.

Failure Rate (Hardware)

Failure Rate

A time dependent measure of #failures/time. Commonly only random failures are considered. The symbol for failure rate is λ(t). A failure rate is tied to a failure mode. This is a hardware related metric.

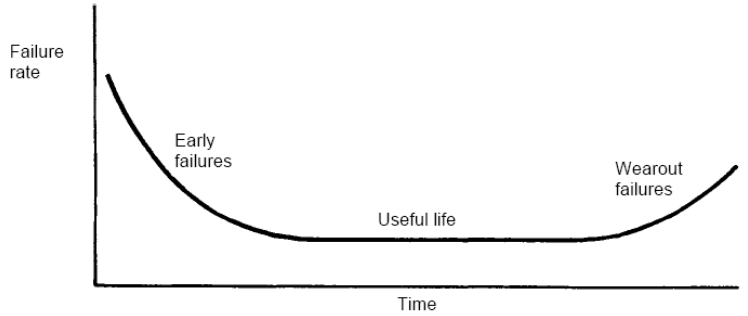
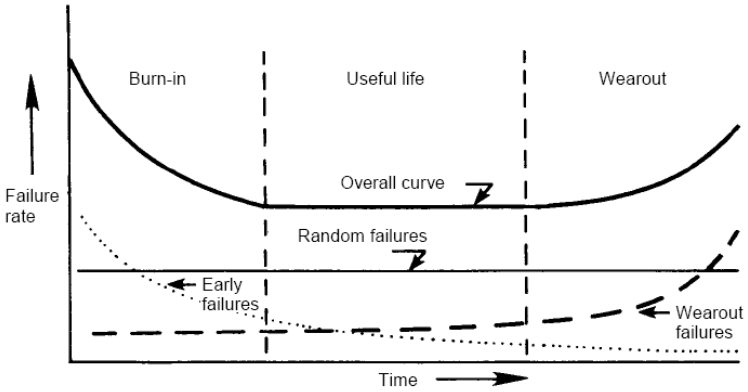


Figure 2.4

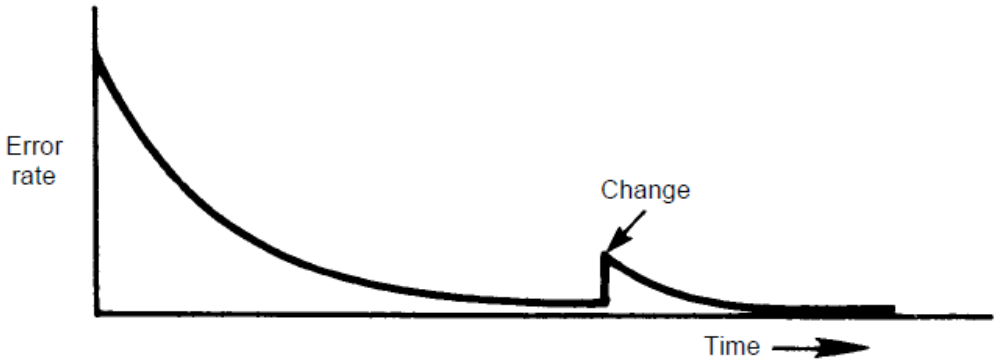


Source:

Smith: Reliability, Maintainability and Risk

Failure Rate (Software)

- A failure has been defined as deviation from the specification. This deviation can happen in two ways
  - Random (Hardware only) – happen randomly in time. The rate is predictable (statistical quantification). Previous slide.
  - Systematic (Hardware and Software) – linked to a certain cause (fault, defect, bug) which is present at time of commissioning
    They are not predictable. A rigorous design and qualification process must be applied.
  - On change (e.g. software update the error rate may increase)



Source:
Smith: Reliability, Maintainability and Risk

Reliability

Reliability

Reliability of a system or component is defined to be the probability that a given system or component will perform a required function under specified conditions for a specified period of time.

- "probability of non-failure (survival) in a given period"
- Reliability of a system function is modeled as: $R(t) = e^{-\lambda t}$ if the failure rate is constant.
- λ is often expressed as failures per 10^6 hours or FIT (failures per 10^9 hours).
- If "λt" small then R(t) = 1 - λt

Mean Time Between Failure (MTBF)

MTBF

Mean Time Between Failures (MTBF) is the average time a system will run between failures. The MTBF is usually expressed in hours.

Let us consider N items with k having failed at time t, T being the cumulative time.

$$N_s(t) = N - k \text{ ; number surviving at time } t$$

$$R(t) = \frac{N_s(t)}{N}$$

$$T_{total} = \int_0^\infty N_s(t)\,dt$$

$$MTBF : \Theta = \int_0^\infty \frac{N_s(t)}{N}\,dt = \int_0^\infty R(t)\,dt = \int_0^\infty e^{-\lambda t}\,dt$$

$$\Theta = \lambda^{-1},\ \lambda = const.$$

MTBF II

The observed MTBF (not all items have failed but k):

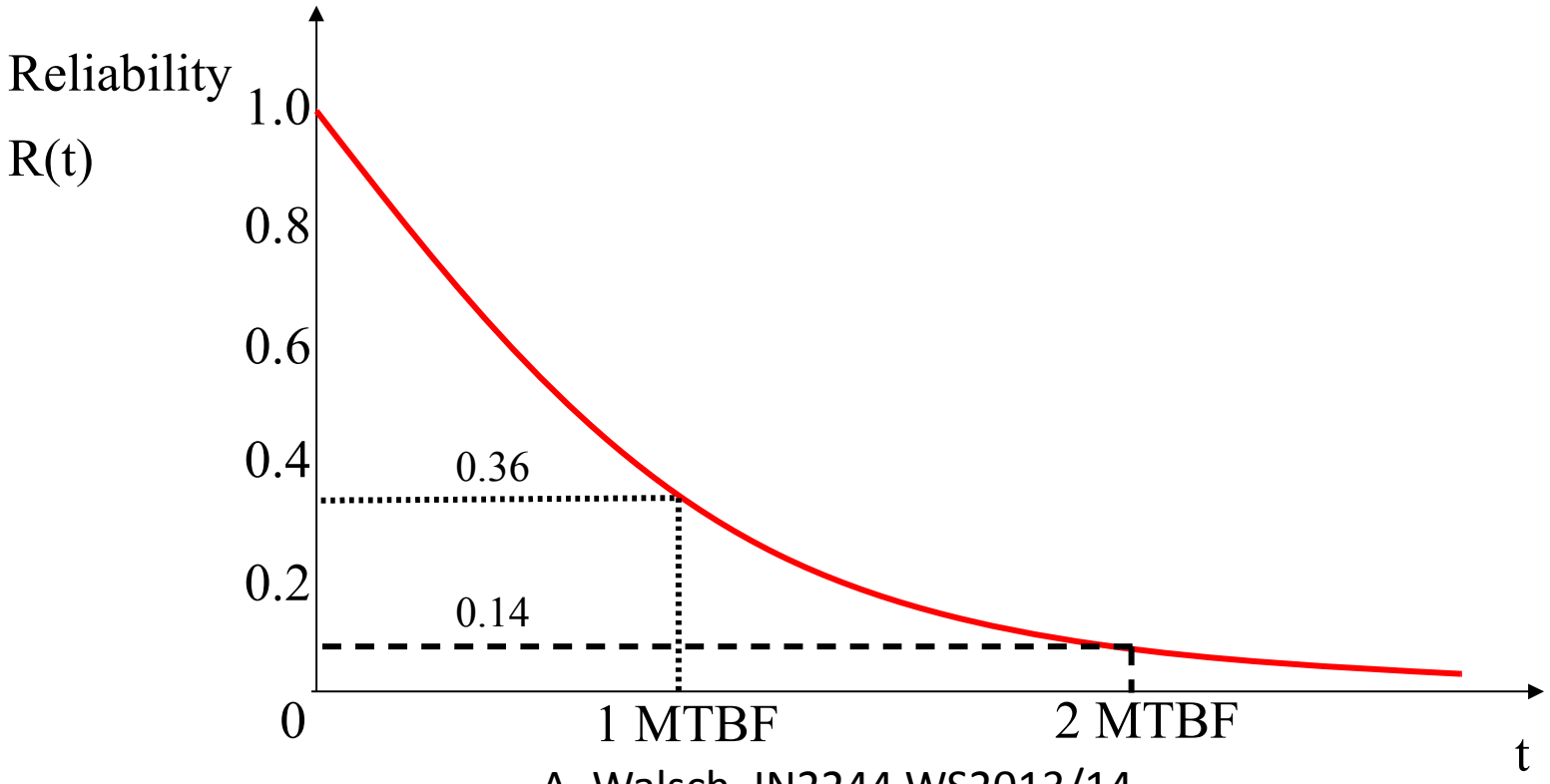
$$\hat{\Theta} = \frac{T}{k}$$ T = total time, k = failed items (total N)

Relation between Reliability and MTBF

$$R(t) = e^{-\lambda t} = e^{-\frac{t}{\Theta}}$$

$$t = \Theta \Rightarrow R = e^{-1} \approx 0.37$$

$$t = 2\Theta \Rightarrow R = e^{-2} \approx 0.14$$



Failure Rate Example

A system (S) has 10 components. Each component does have a failure rate of 5 per 10^6 hours (5000 FIT). Calculate the failure rate and MTBF of a function. Consider two cases:

- All components are required to perform the function (single point of failure).

- Each component performs a different function. Calculate the metrics for any of the functions.

We assume that there is only one failure mode for the component.

Failure Rate Example II

A) All components are required to perform the function (single point of failure)

λ_C = 5000 FIT

$\lambda_{function}$ = 10 * 5000 FIT = 50000 FIT ($5 * 10^{-5}$ failures/hour)

MTBF = 20000h

B) Each component performs a different function

$\lambda_C = \lambda_{function}$ = 5000 FIT = $5 * 10^{-6}$ failures/hour; MTBF = 200000h

MCU Example

Reliability Report - 2nd Quarter 2013 (CY) — **Publish Date: 12 Sep, 2013**

Reliability Die Monitor :

Search by Device : [ ] GO »

OR

Search by Device Family : dsPIC33F

OR

Search by Process : ---Select a Process---

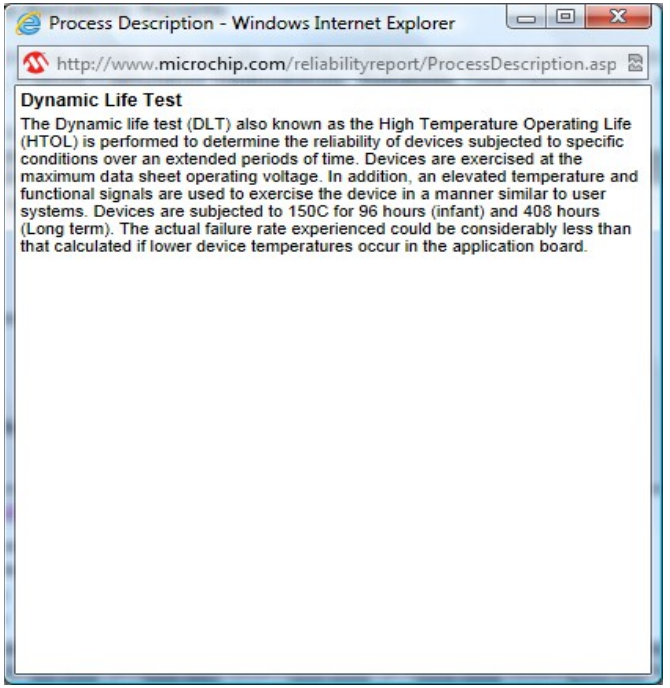
Reliability Package Monitor :

Search by Package : QFN

Dynamic Life Testing

Stress Temperature : 150 degrees C
Derated Temperature : 55 degrees C
Activation Energy : 0.7 eV
Acceleration Rate : 259

Device	Report Period	96 Hours Fails	96 Hours Sample	408 Hours Fails	408 Hours Sample	Device Hours	Total Life FIT Rate 60% Confidence	MTTF (Years)
dsPIC33F	YTD-13	0	3,080	0	3,075	1,255,080	3	40,509
	Rolling Yr - 12/13	1	6,158	0	6,141	2,507,160	3	36,665
	CUM 09-13	0	26,163	4	24,947	10,295,112	2	58,143
	QRT-13	0	1,680	0	1,675	683,880	5	22,073



Mean Down Time (MDT)

MDT

Mean Down Time (MDT) is the average time a system is in a failed state and can not execute its function.
MTBF can be understood as the mean up time.

MTTR

Mean Time to Repair (MTTR) is overlapping with MDT. Used for maintenance calculations. It can be visualized as the average time it takes (a technician) to repair the system such that it is up again.
We will not use MTTR in this lecture anymore.

For software the equivalent would be the time it takes to make a modification (e.g. bug fix, update) and install the new software function.

Availability

Availability

Availability is the probability that a system is functioning at any time during its scheduled working period (in percent).

$$A = \frac{up\,time}{total\,time} = \frac{up\,time}{up\,time + down\,time} = \frac{MTBF}{MTBF + MDT}$$

Reliability vs. Availability:

Reliability is inherent to a function given its specified conditions (internal properties). Availability takes failure and repair into account (internal and external properties).

Unavailability Example

$\lambda = 10^{-6}$ failures/hour ; MDT = 10h

Unavailability = ?

Unavailability Example

$\lambda = 10^{-6}$ failures/hour ; MDT = 10h

Unavailability = ?

$$U = \frac{down\,time}{total\,time} = \frac{MDT}{MTBF + MDT} \approx \lambda * MDT$$

$$\Rightarrow U = 10^{-5}$$

The Bernoulli Experiment applied to Reliability

We have a total number of n identical components. For each component only two states are defined: “functioning” or “has failed”. Both states have a certain probability assigned.
The Bernoulli experiment gives us the probability of finding k (out of n) components in a functioning state.

We state:

P(functioning) + P (failed) = 1 ;
P(functioning) = p; P (failed) = q

The Bernoulli Experiment II

The probability of k functioning components out of n total is

$$P(n, p, k) = \binom{n}{k} p^k q^{n-k}$$

Now we need the probability that a system function (spread across k components or sub-functions) is working -> reliability ("probability of survival")

$$P(n, p, k) = \binom{n}{k} R^k (1-R)^{n-k}$$

is the probability of having k functioning components in an assembly of n total.

Series Reliability Calculation

All n components above need to work such that the series assembly (system) is functioning.

The probability of having n functioning blocks out of n total is

$$R_S = P(n, n, k) = \binom{n}{n} R^n (1-R)^{n-n} = R^n$$

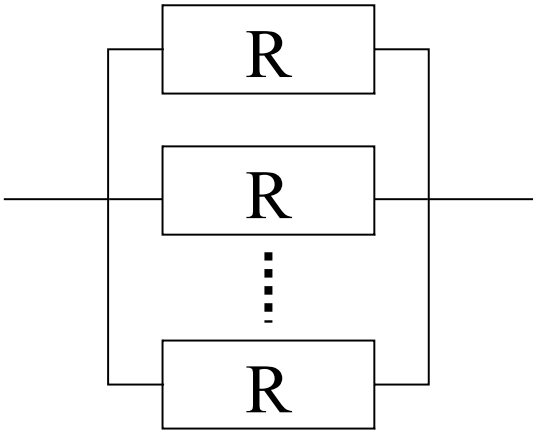
when using a Bernoulli experiment.

$$R_S = R * R * \ldots * R = R^n$$

when using the probability law for independent events.

Parallel Reliability Calculation
- Full active Redundancy -

At least 1 component needs to be functioning in full active redundancy configuration.



Therefore, the assembly is working if n or (n-1) or ... or 1 component work.

n=2: 2 or 1 component must be functioning.

$$R_S = \binom{2}{2} R^2 (1-R)^0 + \binom{2}{1} R^1 (1-R)^1 = 2\mathrm{R} - R^2 = R(2-R) > R$$

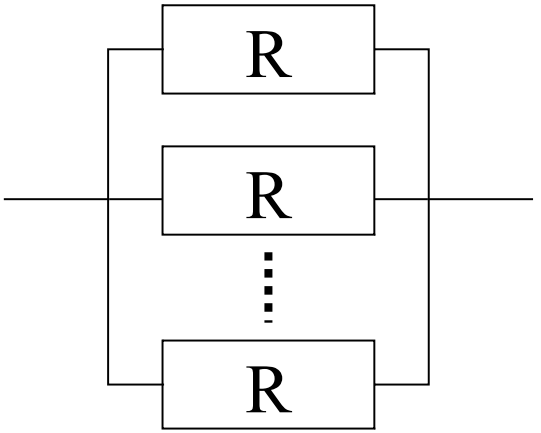
n=n:

$$R_S = \binom{n}{n} R^n (1-R)^0 + \ldots + \binom{n}{1} R^1 (1-R)^{n-1} = 1 - (1-R)^n$$

Parallel Reliability Calculation
- Partial active Redundancy -

At least m components need to be functioning in partial active redundancy configuration.

Therefore, the assembly is working if n or (n-1) or ... or m components work.



n=3: m = 2 (2oo3, spoken “two out of three”)

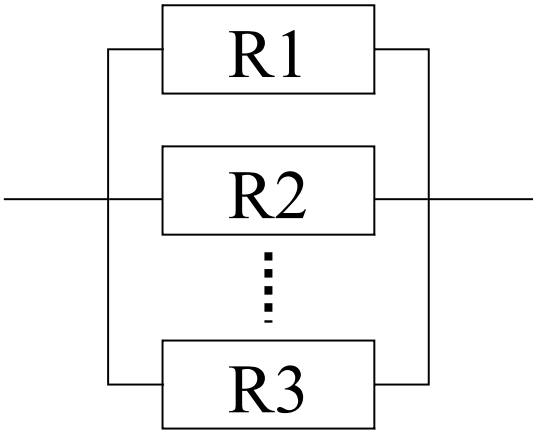
$$R_S = \binom{3}{3} R^3 (1-R)^0 + \binom{3}{2} R^1 (1-R)^1 = 3R^2 - 2R^3$$

n=N, m = M: (MooN, spoken “M out of N”)

$$R_S = \binom{n}{n} R^n (1-R)^0 + \ldots + \binom{n}{m} R^m (1-R)^{n-m}$$

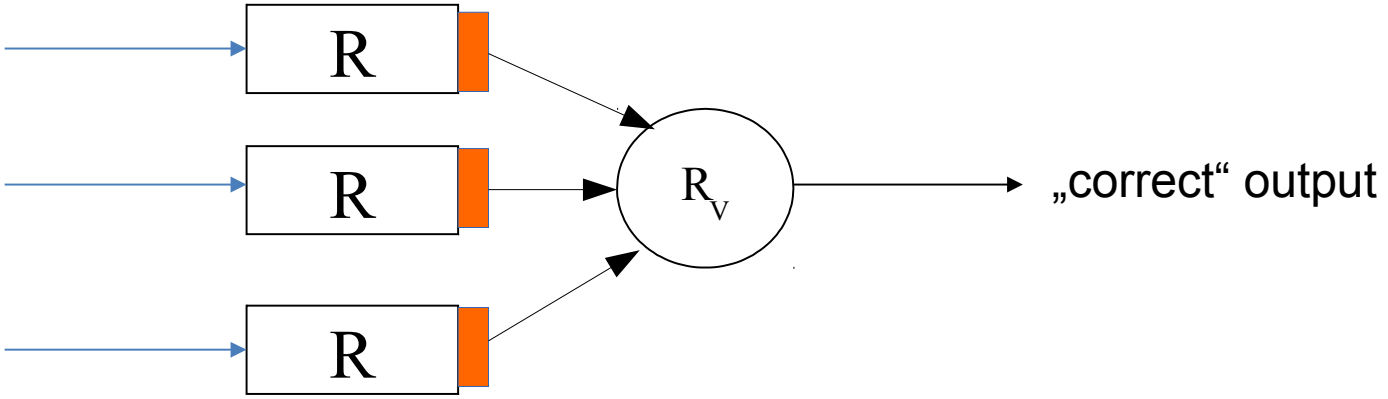
Replication and Diversity
- Avoidance of Common Cause Faults -

- Replication:
  identical copy of the original function
  (identical in specification for all phases
  of development and in implementation)



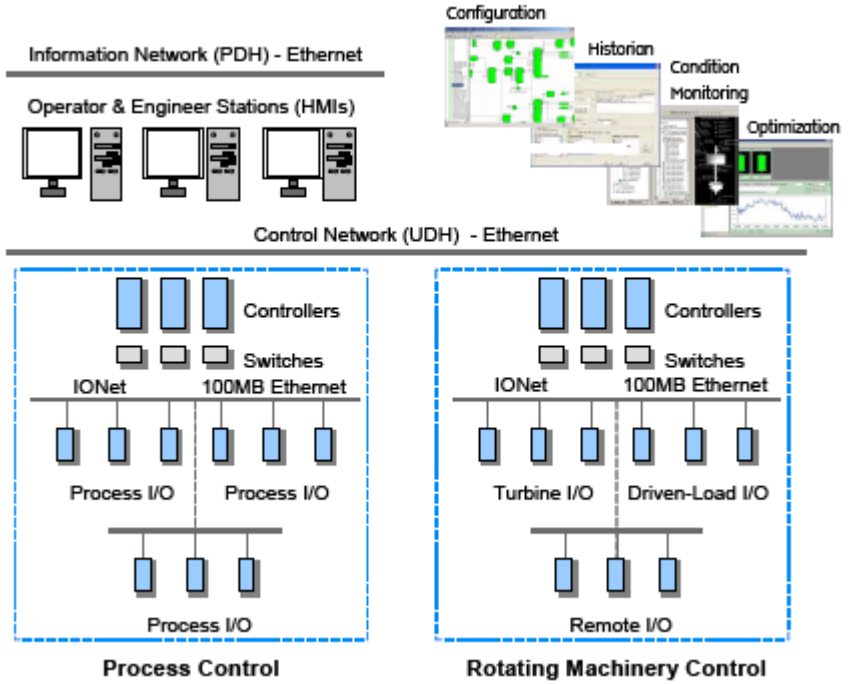
- Diversity:
  different copy of the original function
  (differences in specification and
  implementation – same interface to caller, same functional
  semantics – different behavioral semantics)
- From DO-178B (multiple dissimilar software, n-version programming): different programming languages, different compilers, dissimilar processor, different teams, different linkers and loaders, different design standards)

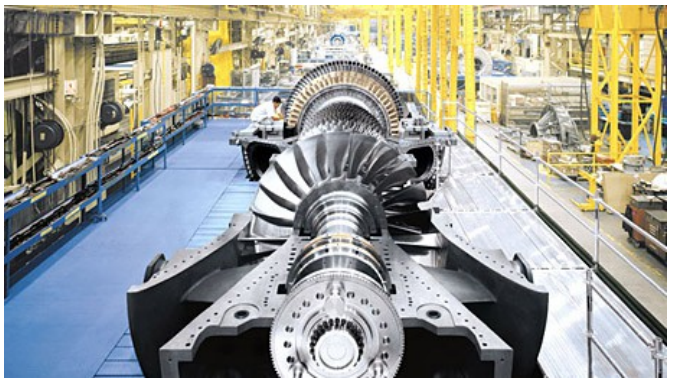
Partial Active Redundancy Example
- 2oo3 Majority Voter -



- Three inputs, one output: Triple Modular Redundancy (TMR)
- Input stages have reliability R, Voter and output stage have reliability R_V
- One unit may fail but no more (partial redundancy)
- Reliability: $R_S = 3R^2 - 2R^3 = R(3R - 2R^2) > R\,?$
- Adjudication method: majority, median, consensus

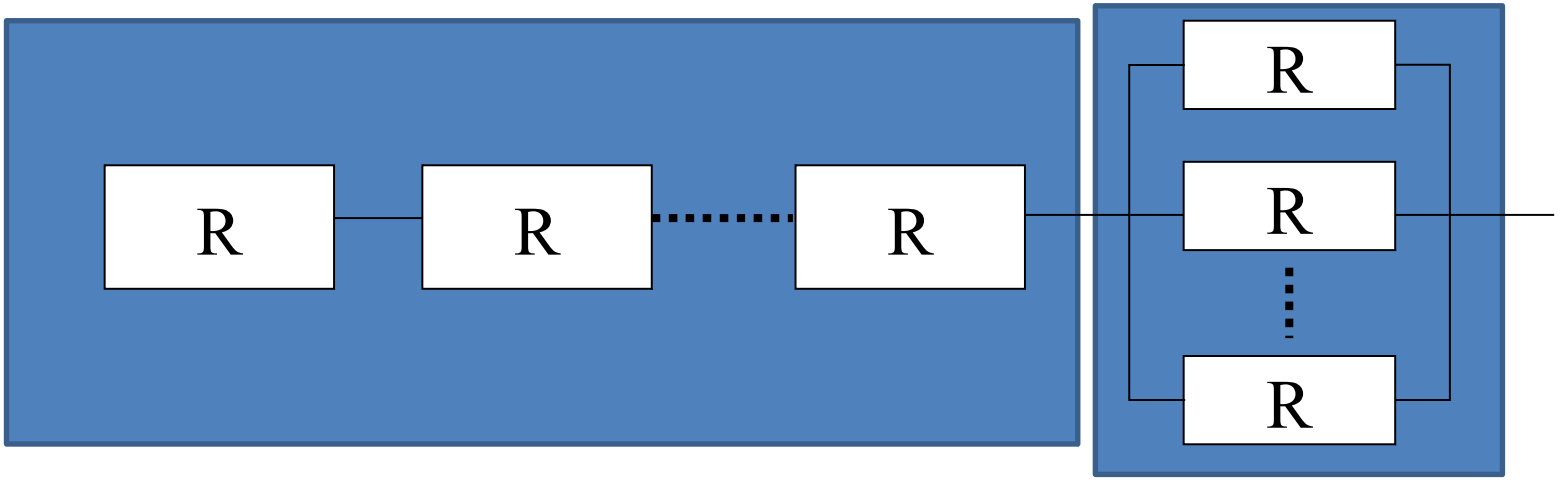
Partial Active Redundancy Example
- 2oo3 Majority Voter -



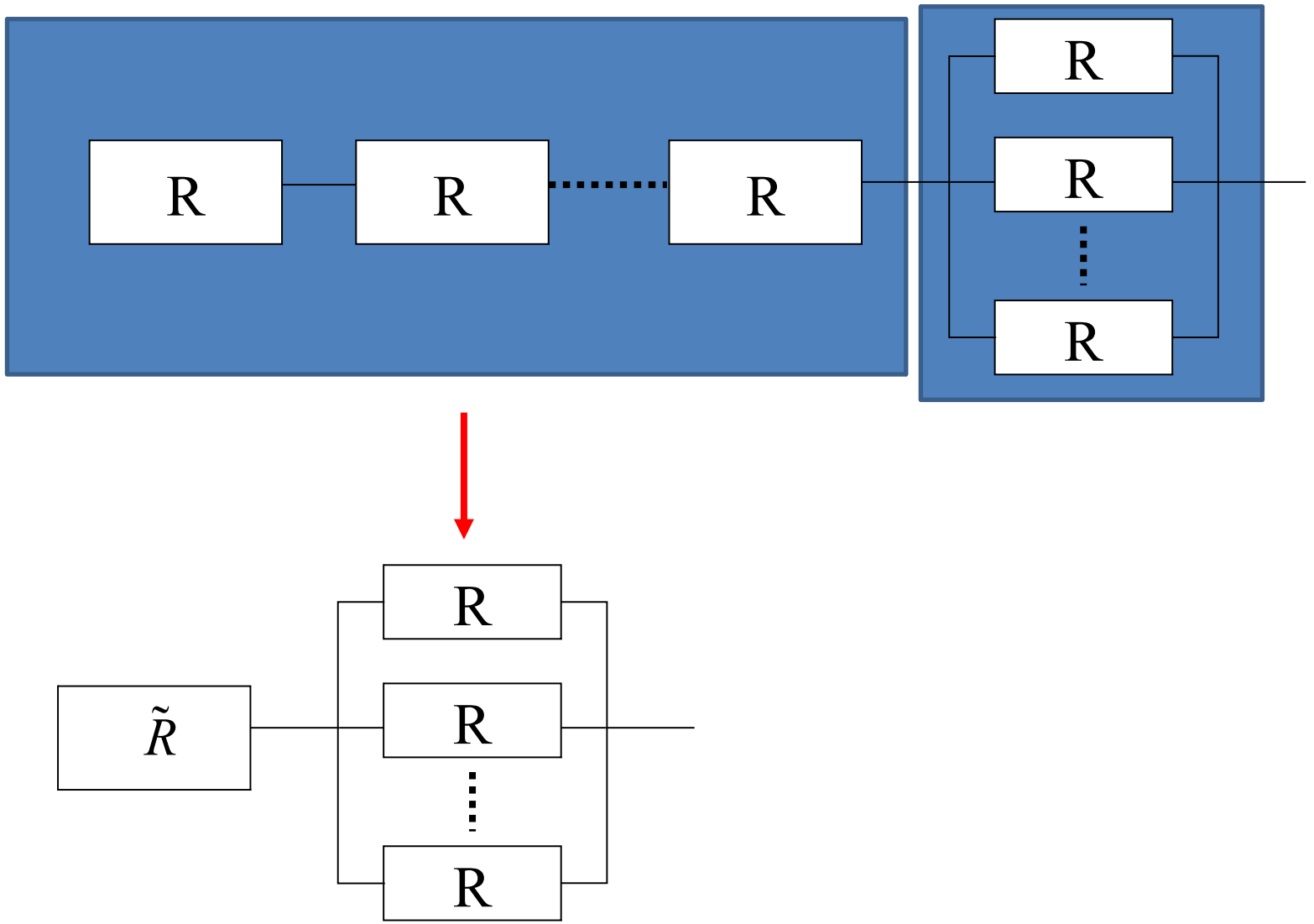


Source:
GE Energy

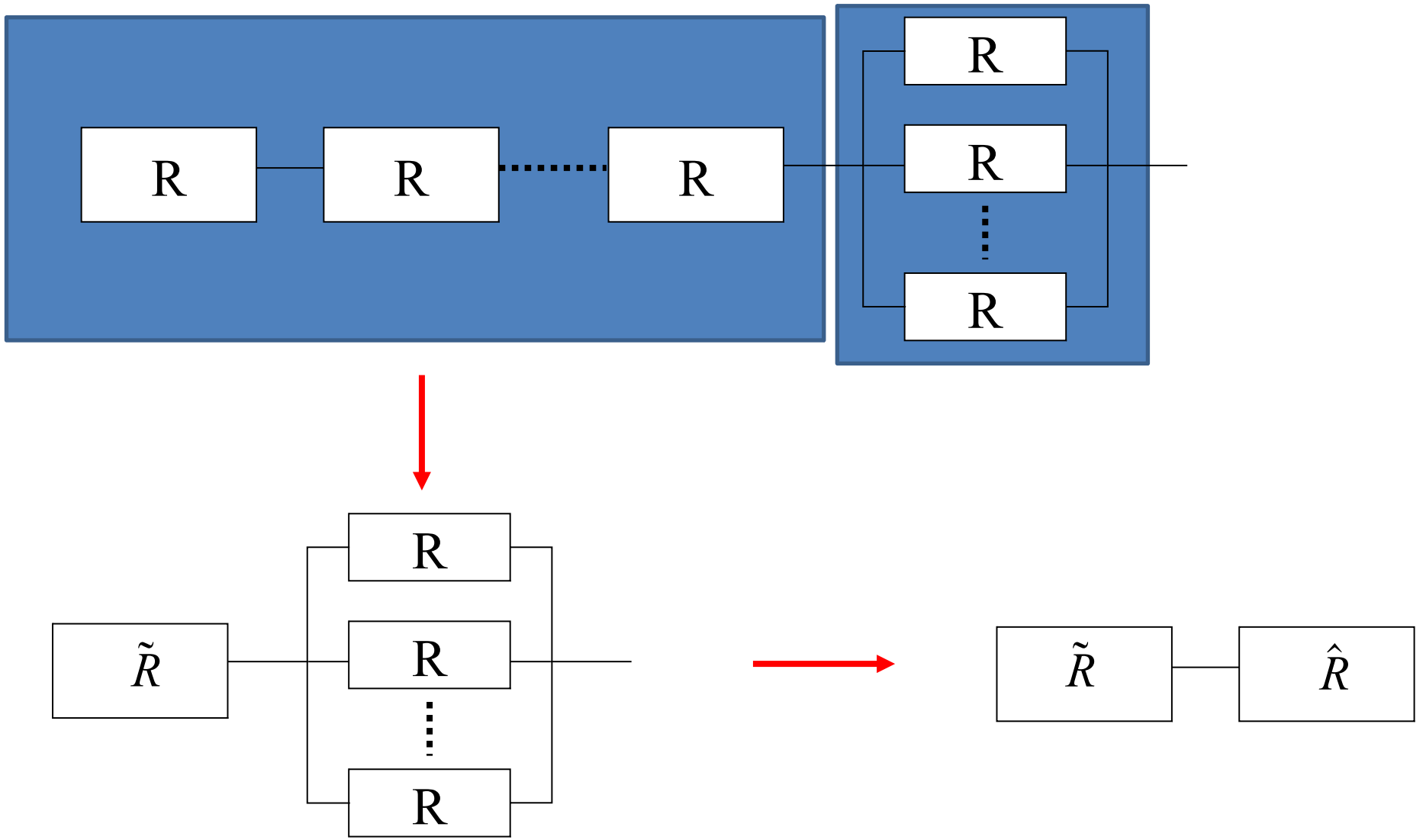
Complex Configurations



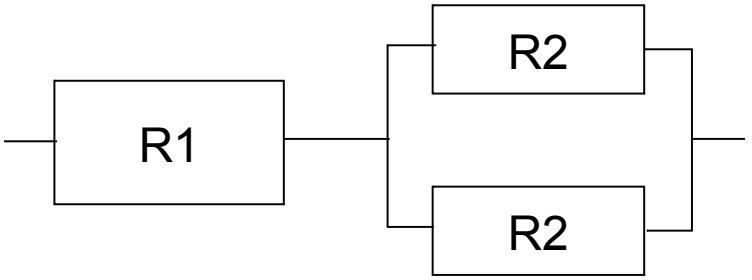
Complex Configurations



Complex Configurations



Complex Configuration Example

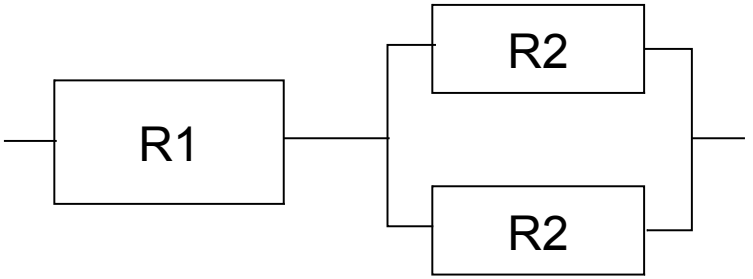


Calculate the MTBF of this system (S) made of identical components (R1=R2=R3=R). $\lambda = const.$

$$R_S = R\left(2\mathrm{R} - R^2\right) = 2e^{-2\lambda t} - e^{-3\lambda t}$$

$$\Theta = \int_0^\infty 2e^{-2\lambda t} - e^{-3\lambda t}\, dt = \ldots = \frac{2}{3\lambda}$$

Software Considerations



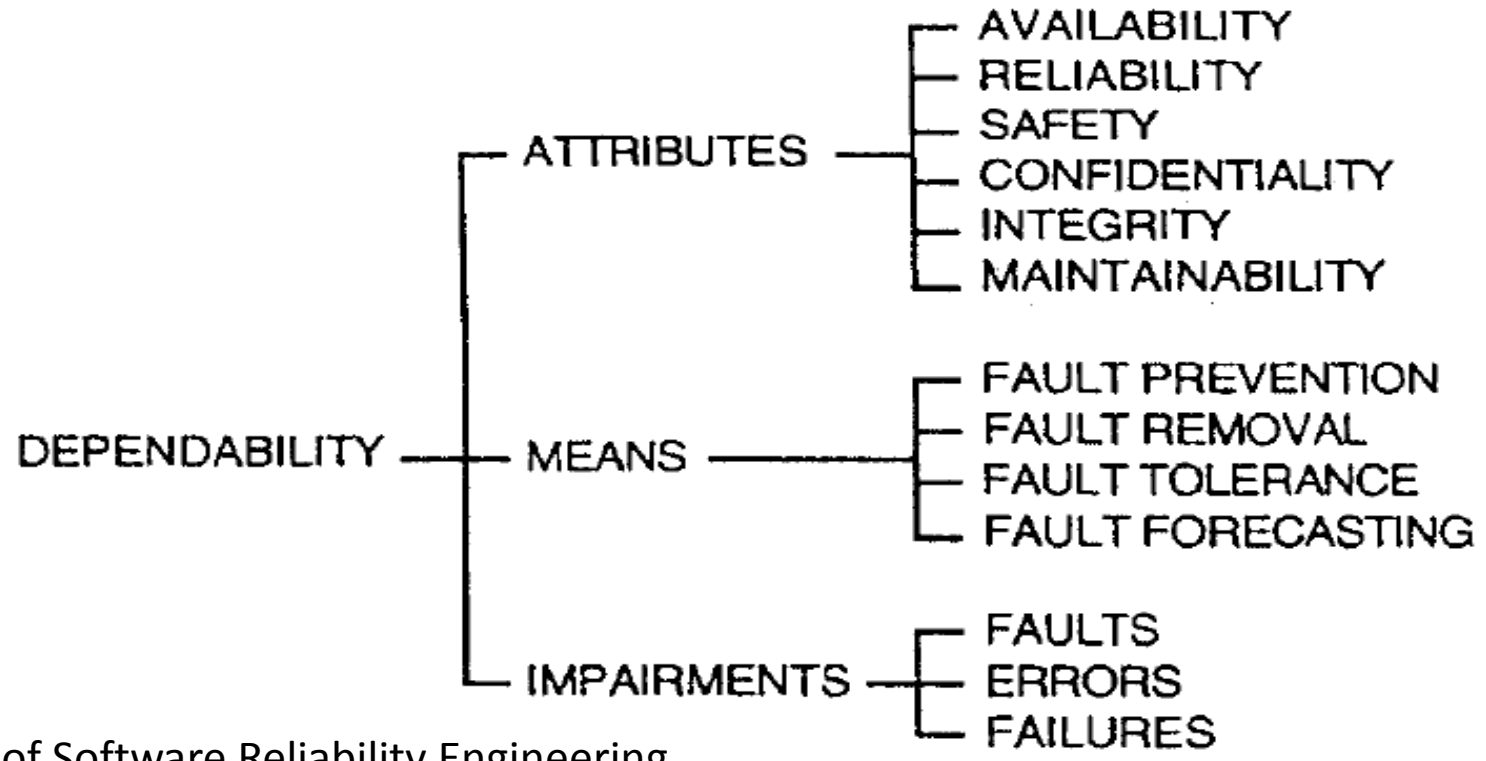
Definition:
Probability of failure-free software operation for a specified period of time in a specified environment (from „Standard Glossary of Software Engineering Terminology" STD-729-1991, ANSI/IEEE 1991)

Four Methods:

- Fault Prevention: avoid by construction (development: left wing of V-model)
- Fault Removal: detect by verification and validation (development: right wing of V-model)
- Fault Tolerance: provide service despite fault (operation)
- Fault Forecasting: estimate faults/failures by evaluation (future)

Embedded System Development

Main Drivers:
Cost, Function, Performance, Dependability (trustworthiness)



Source:
Handbook of Software Reliability Engineering
IEEE 1996, Michael R. Lyu

Backup

Feasibility Study

Feasibility Study

A feasibility study decides whether or not the proposed system or component is worthwhile. Usually a study on the most risky elements of a new development.

A short focused study (simulation or setup) that checks

- If the proposed system can be engineered using current technology and within budget (technical and economic feasibility);
- If the proposed system can be integrated with other systems that are used (interoperability).
- If the proposed system can meet the requirements (especially non-functional like reliability, e.g.)

Reliability in Product Descriptions

Maximize Efficiency.
Improve Quality. Reduce Costs.
Enhance Safety.

Better measurement means a stronger bottom line. Rosemount 3051 Pressure Transmitters deliver proven reliability, performance and unparalleled safety to increase your plant profitability. With over 3.5 million installations, the Rosemount 3051 is more than field proven – it is the industry standard.

Since introduction, you have experienced a seamless evolution of Coplanar™ platform enhancements. Our investment legacy gives you the means to achieve the business results you demand – without the risk of changing platforms. Rosemount 3051 Pressure Transmitters are your pathway to better measurement.

Source:
Rosemount

What is Rosemount marketing advertising with in this example?

Functional Requirements

Functional Requirement

Core system function used to fulfill the system purpose – we ask what must the system do?

- Inputs and associated outputs (valid inputs, invalid inputs, warnings, errors)
- Formats for I/O
- User Interfaces and different roles (technician, customer, …)
- States of the system (operational, error)
- Failure modes

Functional Requirements Capture

Look at system as black-box

- Look at what it interacts with
  Other systems, devices, users (identified as user-roles)
  UML: use case diagram
- Look at how it interacts
  Data flow, control
  UML: sequence diagram
- Traditional, basic form: textual, according to some template or standard form (text document, unique ID)
- Model-based form: use case and sequence diagrams (UML) + textual description

Functional Requirements
- Textual Examples -

“The system shall connect to a pressure sensor with 4 – 20 mA interface.”

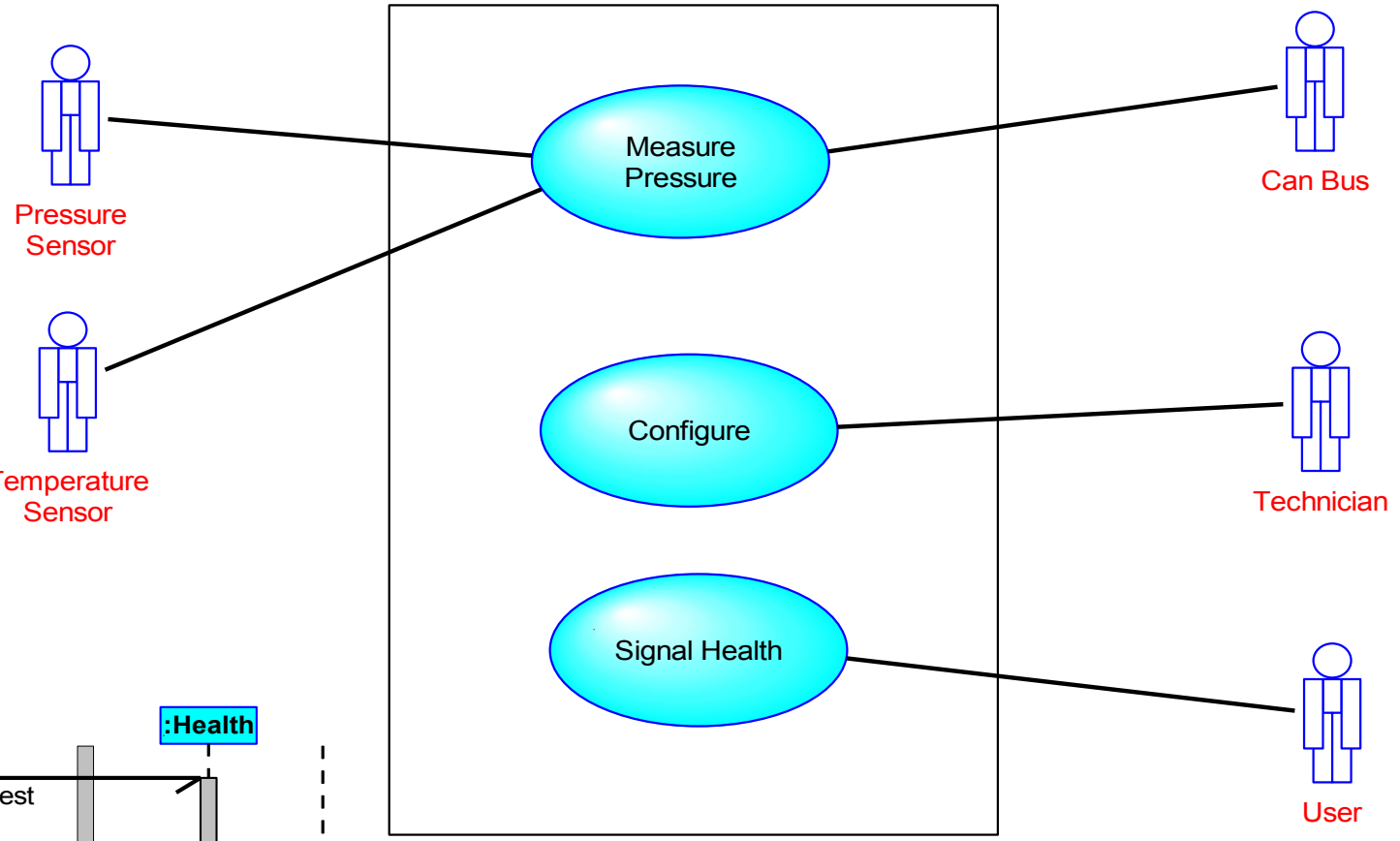
“The system shall not supply power to the pressure sensor.”

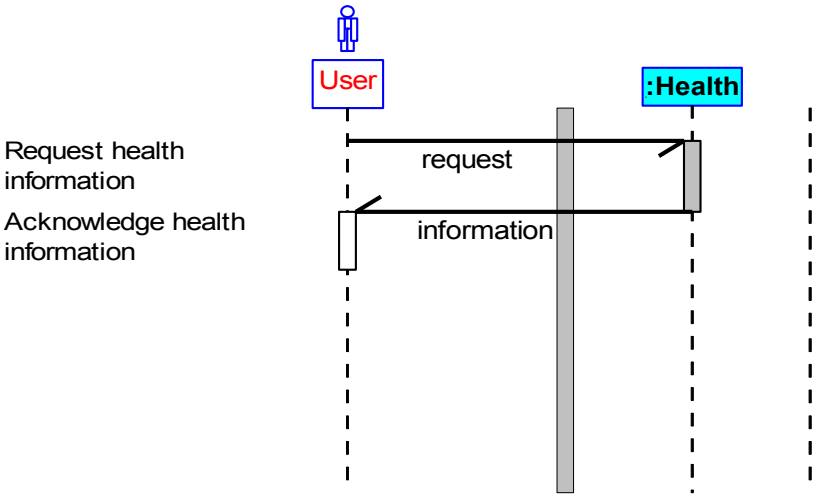
“The system shall indicate a violation of input range by an “out of range” error message according to [*std. xyz.*] if the current input is less than 5 mA or more than 19 mA.*”*

*“*All pressure readings shall be communicated via the CAN bus.”

“All pressure readings shall be communicated according to [*std. xyz*]”

Functional Requirements
- Model Driven Development (MDD) Example -





- Functional View
- Actors = external users or devices
- Use cases = functions

MDD Example II