RELIABILITY

EL-GY 6483 Real Time Embedded Systems

CASE STUDIES

THERAC 25

- Radiation therapy machine used in 1980s
- Had several software bugs, including a race condition that allowed a high-power electron beam to be activated without an attenuator in place
- In at least 6 accidents, patients were given massive overdoses, resulting in death or serious injury

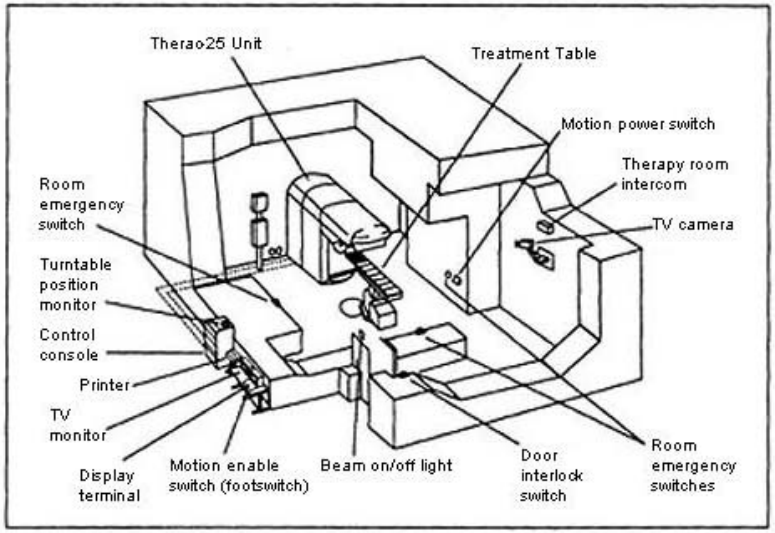
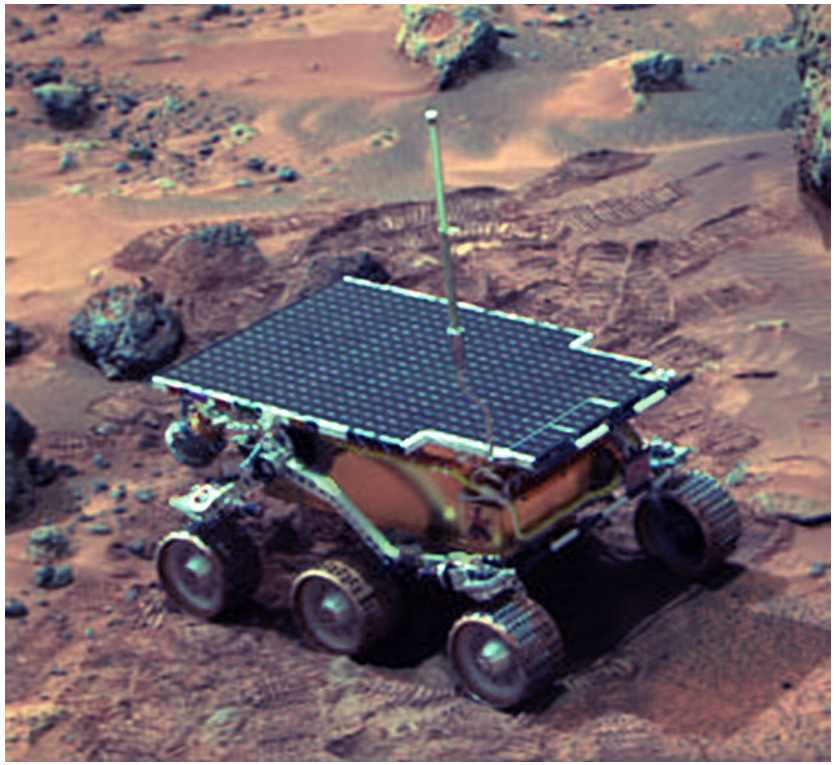


Figure 1. Typical Therac-25 facility

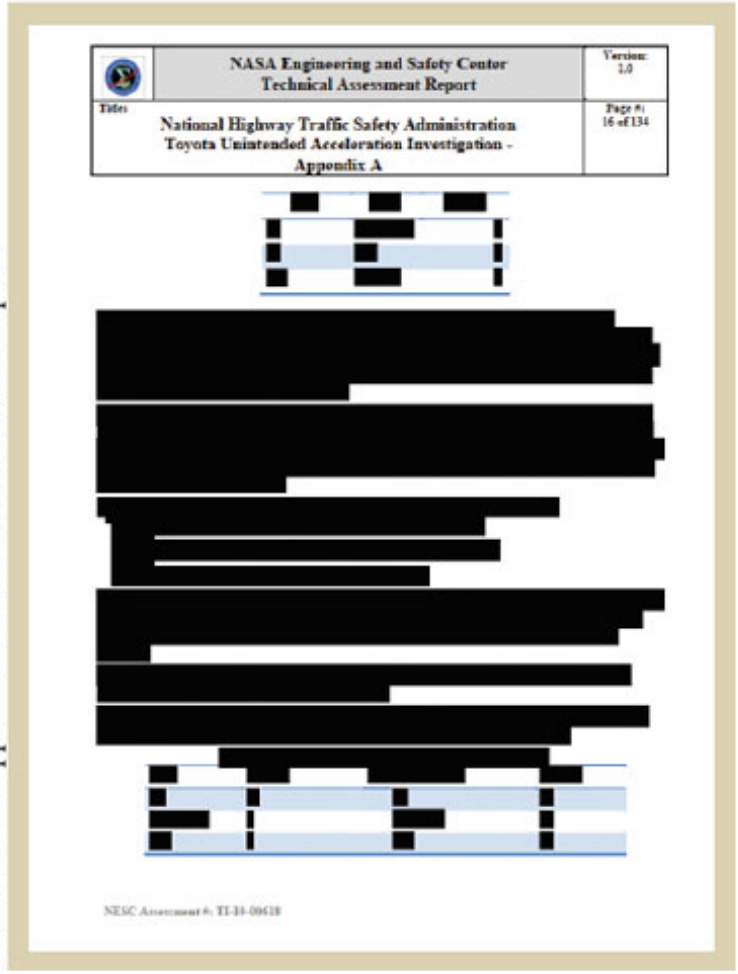
MARS PATHFINDER

- Classic priority inversion bug, system continuously reset itself
- Engineers were able to enable priority inheritance in OS, upload new firmware remotely and repair system



TOYOTA UNINTENDED ACCELERATION

- Multiple reports of unintended acceleration in certain Toyota vehicles
- Specific proximate cause of known incidents not definitively (publicly) identified, some software bugs have been highlighted as potential candidates


NASA Engineering and Safety Center
Technical Assessment Report

Version: 1.0

Title: National Highway Traffic Safety Administration Toyota Unintended Acceleration Investigation - Appendix A

Page #: 16 of 134

Redactions in Appendix A of NHTSA and NASA's report.[1]

RELIABILITY IN EMBEDDED SYSTEMS

SAFETY-CRITICAL SYSTEMS

Reliability concerns are not just limited to multi-billion dollar projects and medical devices:

- A cell phone that doesn't work when the owner needs to call 911.
- A home hot water heater that malfunctions and burns a child.
- An automobile that unintendedly accelerates (see Toyota case study).

LAWS OF ENGINEERING

- Murphy's law: If something can go wrong, it will go wrong.
- Green's law: If a system is designed to be tolerant to a set of faults, there will always exist someone so skilled to cause a nontolerated fault.
- Sodd's second law: Sooner or later, the worst possible set of circumstances is bound to occur.

COROLLARIES TO THE LAWS

- Corollary to Murphy's law: Prepare for faults and errors to occur.
- Corollary to Green's law: Whatever safeguards you put in place to protect your end users from themselves, some users will manage to get around them.
- Corollary to Sodd's second law: Design your system for acceptable performance under worst-case conditions (and best-case conditions - see Mars Pathfinder rover).

DEFINITIONS

- Failure: visible deviation from specification
- Reliability: probability of non-failure during a specific time interval
- Availability: fraction of time that system meets specs
- Dependability: system generally does the right thing at the right time

TYPICAL SOURCES OF FAILURE

- Hardware
- Software
- Communications
- People/UI

HARDWARE RELIABILITY

RELIABILITY

Reliability of hardware components amenable to statistical analysis:

$$R(t) = P(t_{init} \leq t < t_f)$$

We often model the lifetime of a system with exponential distribution, then

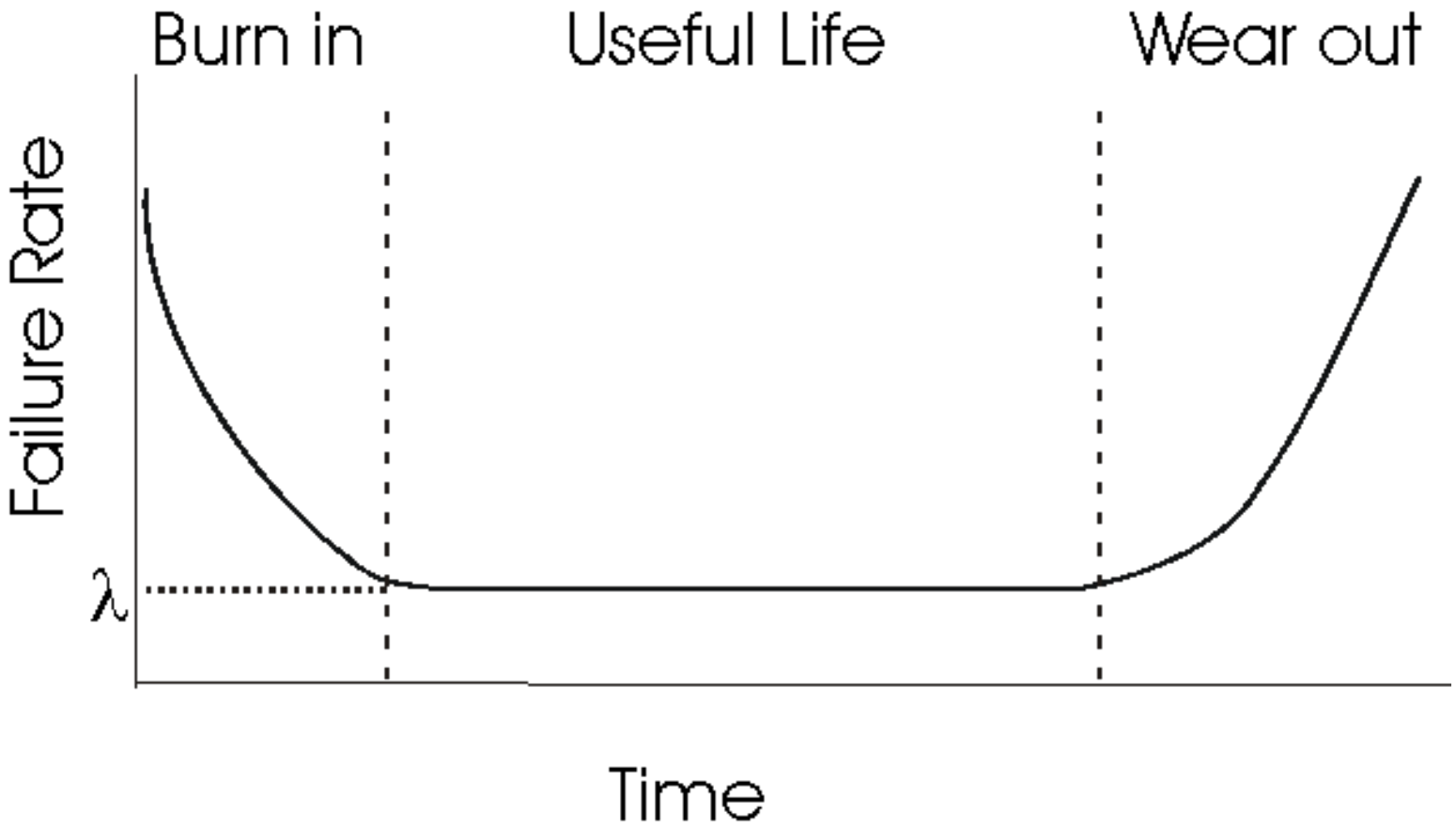
$$R(t) = e^{-\lambda t}$$

where λ is failure rate of components. Then the mean time to first failure is expected value of t_f, .

$$\frac{1}{\lambda}$$

FAILURE RATE

Failure rate is not really constant in time; more of a bathtub curve.



SERIAL COMPONENT RELIABILITY

For n components connected serially,

$$R_s(t) = \prod_{i=1}^{n} R_i(t)$$

and system failure rate is

$$\lambda_s = \sum_{i=1}^{n} \lambda_i$$

PARALLEL COMPONENT RELIABILITY

For n components connected in parallel, with statistically independent failures, failure probability is

$$Q_p(t) = \prod_{i=1}^{n} Q_i(t) = \prod_{i=1}^{n} 1 - R_p(t)$$

and so reliability is

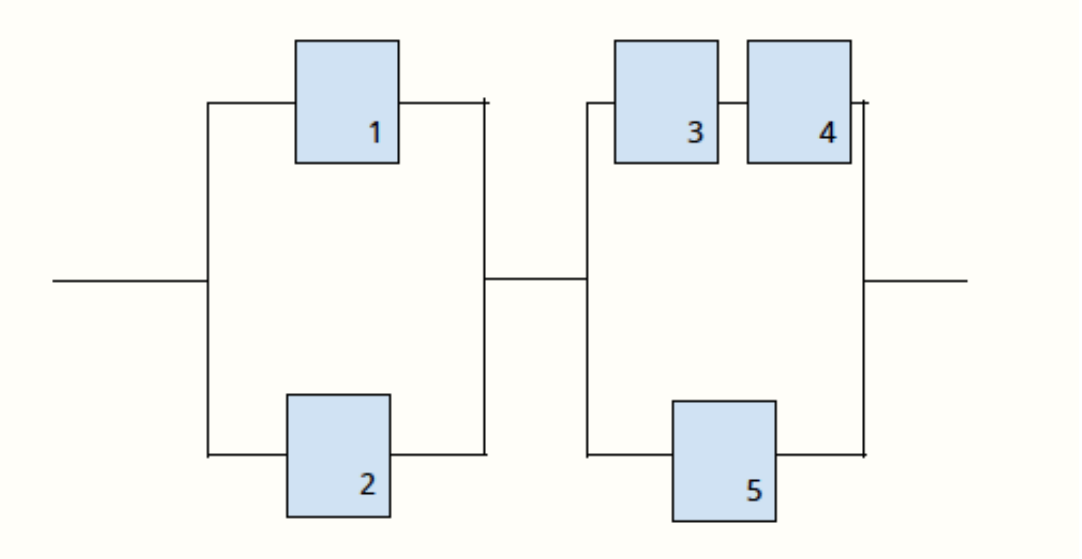
$$R_p(t) = 1 - \prod_{i=1}^{n} (1 - R_i(t))$$

(benefits of redundancy!)

EXAMPLE

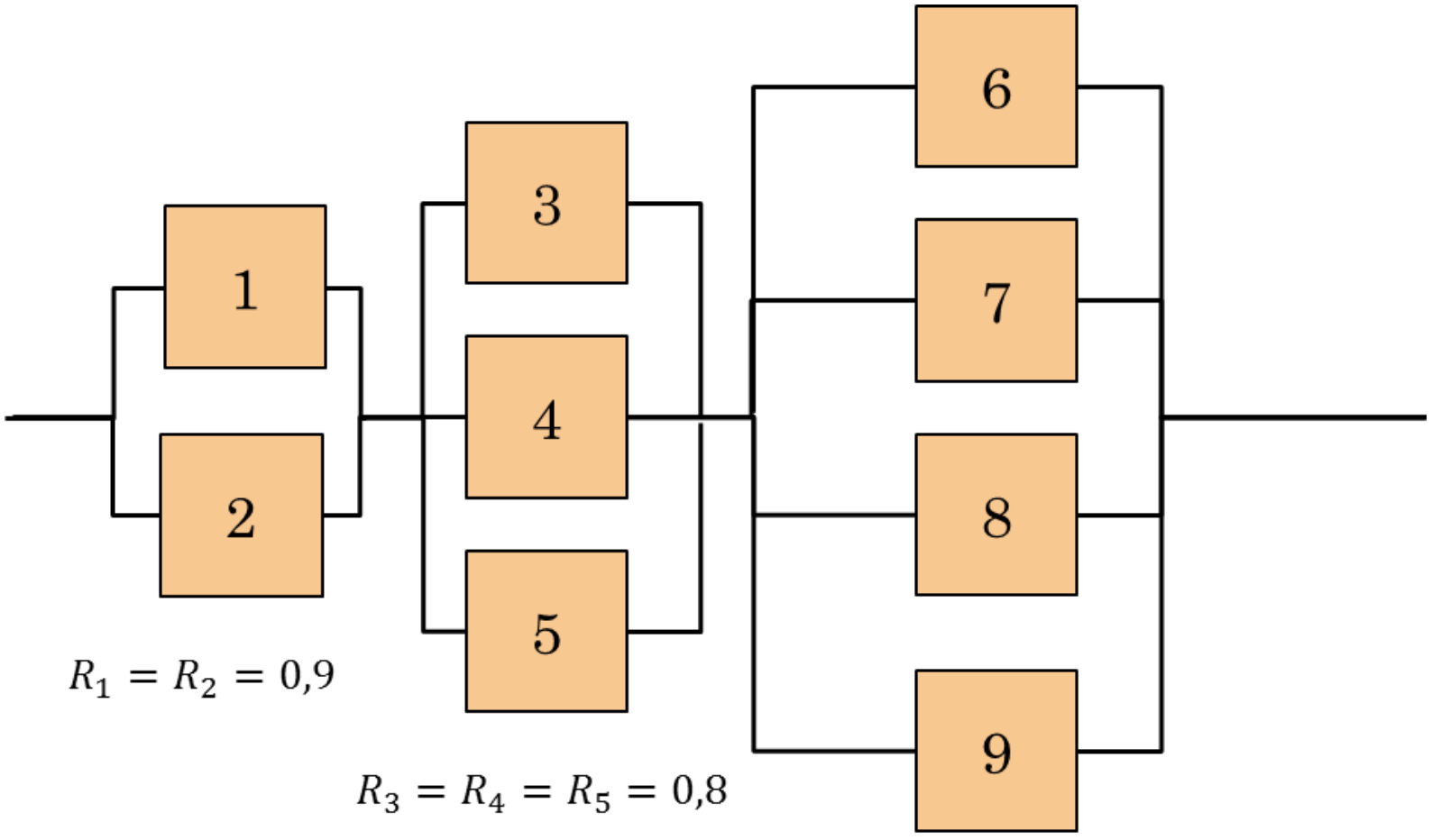
Given R1 = 0.9, R2 = 0.9, R3 = 0.99, R4 = 0.99, R5 = 0.87,

R = [1 - (1 – 0.9)(1 – 0.9)][1 - (1 – 0.87)(1 - (0.99 * 0.99))] = 0.987



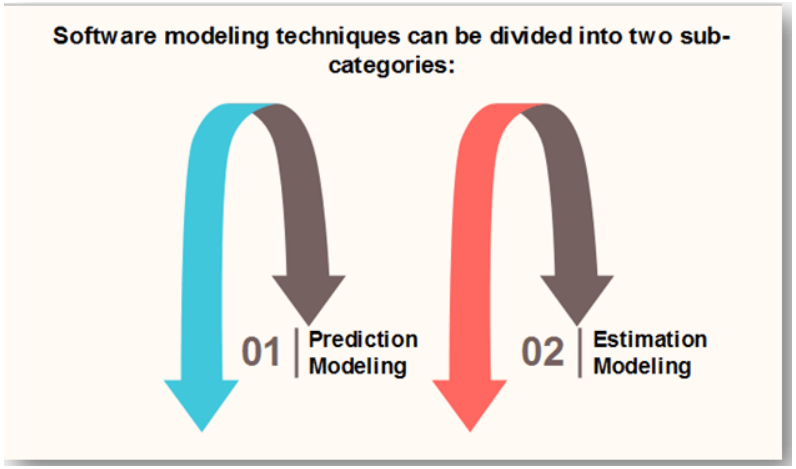
Minimal path set: (2,3,4), (1,3,4), (1,5), (2,5)

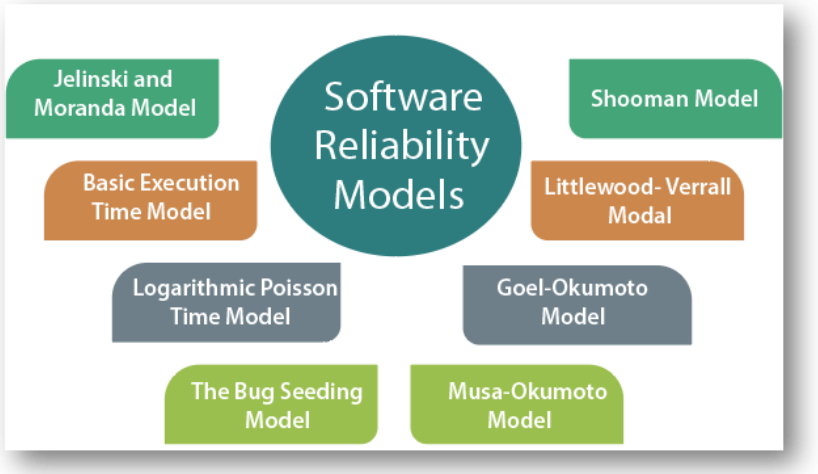
EXAMPLE



SOFTWARE RELIABILITY

MODELING SOFTWARE RELIABILITY



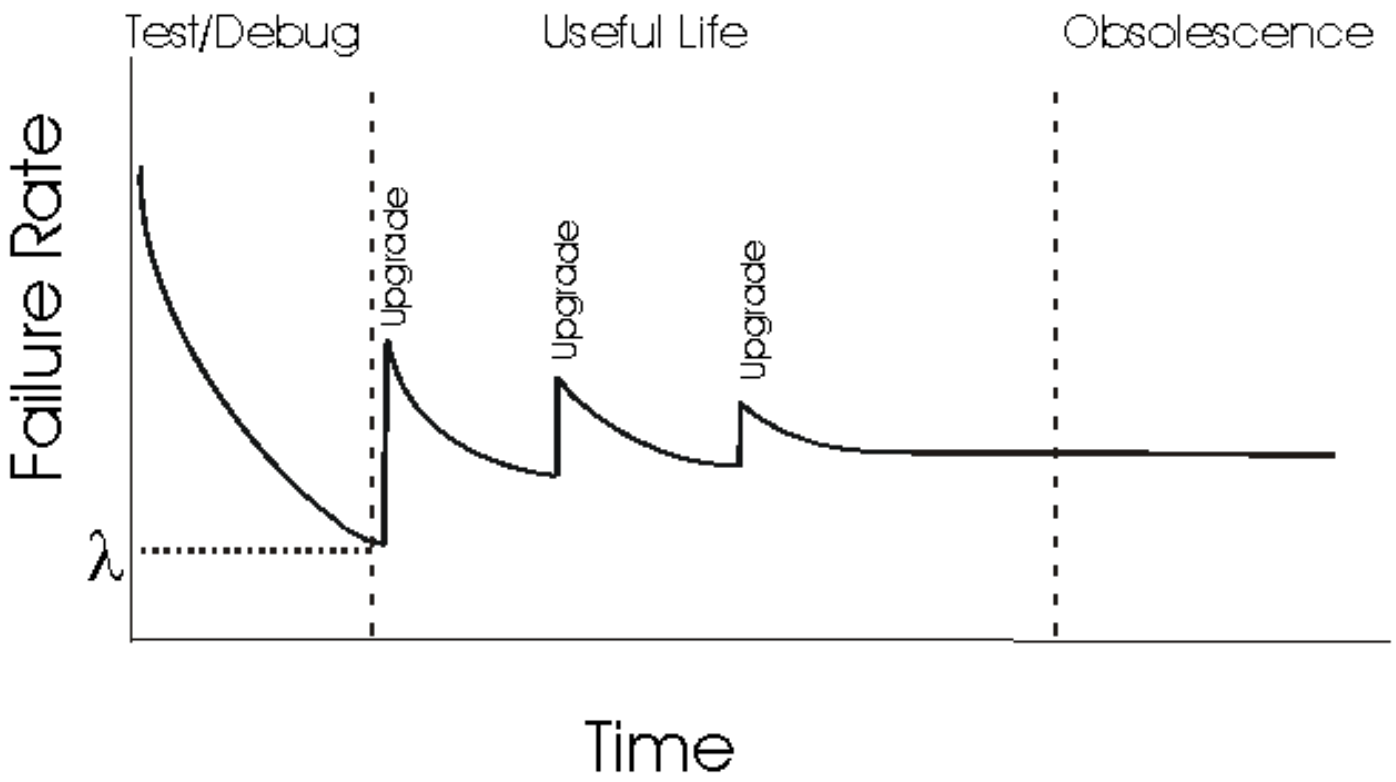


Basics	Prediction Models	Estimation Models
Data Reference	Uses historical information	Uses data from the current software development effort.
When used in development cycle	Usually made before development or test phases; can be used as early as concept phase.	Usually made later in the life cycle (after some data have been collected); not typically used in concept or development phases.
Time Frame	Predict reliability at some future time.	Estimate reliability at either present or some next time.

Reference: https://www.javatpoint.com/software-engineering-software-reliability-models

FAILURE RATE

The bathtub curve for software failure looks like this:



TESTING

THE MYTH OF "EXHAUSTIVE" TESTING

- Therac 25 bug was only triggered by highly skilled operators who typed very quickly
- Mars rovers aren't testing in Mars gravity or real Earth-to-Mars communication conditions
- Toyota vehicles are impossible to test completely at the device level: too many possible operating conditions, timing sequences, faults (which might be intermittent)
    - Toyota tested about 35 million miles at system level, 11 million hours module level software testing
    - In 2010, Toyota sold 2.1 million vehicles - will see many more hours of field use

THE MYTH OF “EXHAUSTIVE” TESTING

Even under ideal conditions: testing can only prove the existence of bugs, not the absence of bugs.

TESTING AT MULTIPLE LEVELS

- Unit testing
- Subsystem testing
- Integration testing
- Regression testing

STATIC VS DYNAMIC TESTING

- Static analysis: test source code while it is not running. Identifies potential faults of certain classes.
- Dynamic analysis: test source code during execution. Identifies realized faults that happened to occur during testing

FUNCTIONAL VS COVERAGE TESTING

- Functional testing (a.k.a. black-box testing) uses tests that check how well implementation meets specs.
    - Stress tests
    - Boundary (input) value tests
    - Failure mode tests
    - Experience-based tests
    - Random tests
    - Performance tests
- Coverage testing (a.k.a. white-box testing) uses tests that execute certain portions of code.

RECOVERY FROM FAILURE

MARS PATHFINDER

"We did see the problem before landing but could not get it to repeat when we tried to track it down. It was not forgotten nor was it deemed unimportant... we ran out of time to get the lower priority issues completed.

We did have one other thing on our side; we knew how robust our system was because that is the way we designed it.

We knew that if this problem occurred we would reset."

WATCHDOG TIMER

- “Kick” the watchdog at regular intervals if system is functional
- Kicking the watchdog resets timer
- If timer elapses (watchdog wasn’t kicked), reset MCU

WATCHDOG TIMER ON STM32F4 DISCOVERY

STM32F4 Discovery board has two watchdogs:

- Independent watchdog has its own clock, runs even if system clock fails, keeps going in standby and stop low-power modes
- Window watchdog runs from APB1 clock

POST-MORTEM

ROOT CAUSE

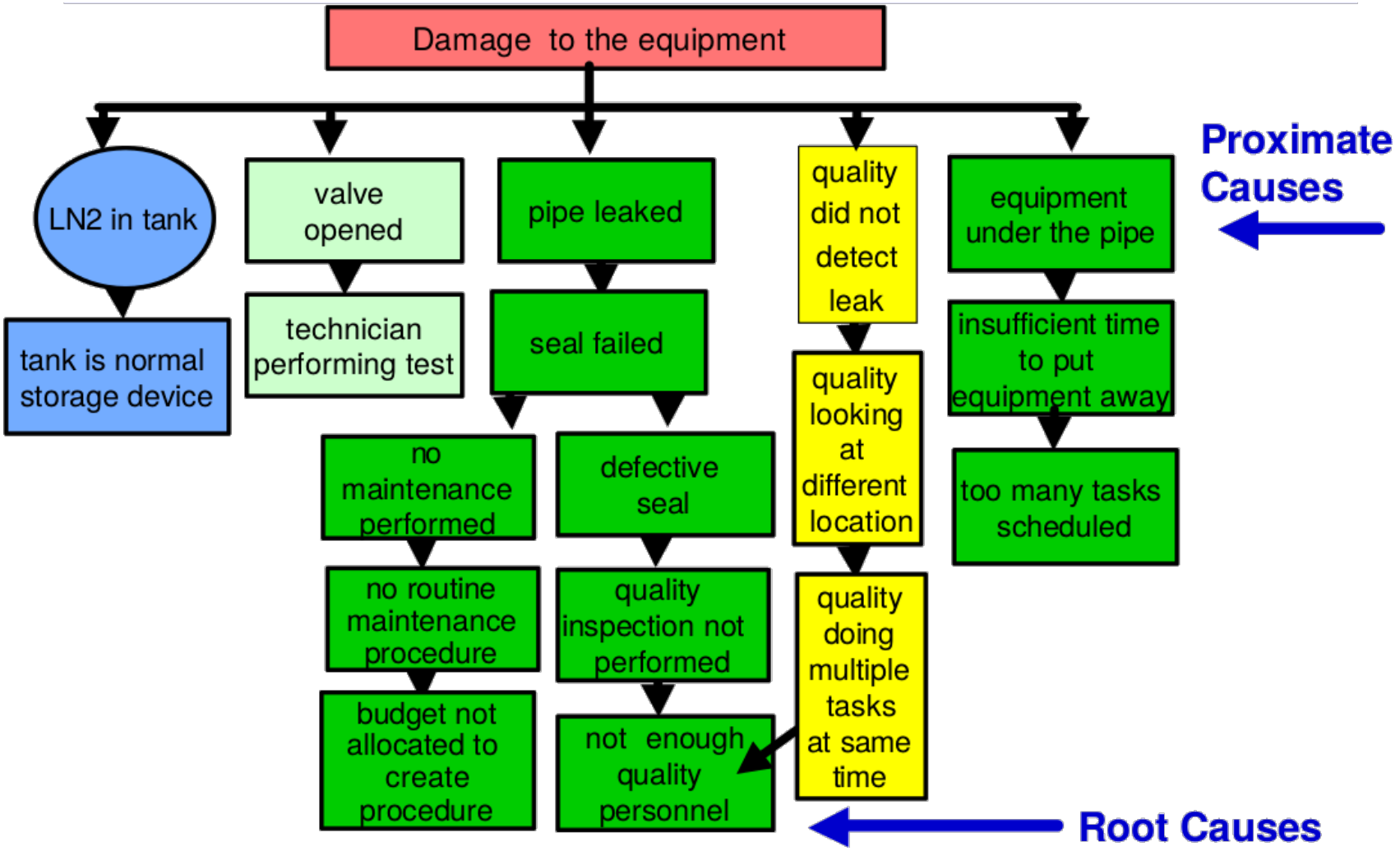
In safety-critical context, when a bug is found, we often use root-cause analysis to implement process improvement when bugs are found.

An undesirable outcome is usually due to some combination of

- Immediately visible cause (proximate cause)
- Underlying organizational causes (root cause)

Common technique: create an event and causal factor tree.

EXAMPLE


Damage to the equipment
LN2 in tank
tank is normal storage device
valve opened
technician performing test
pipe leaked
seal failed
no maintenance performed
no routine maintenance procedure
budget not allocated to create procedure
defective seal
quality inspection not performed
not enough quality personnel
quality did not detect leak
quality looking at different location
quality doing multiple tasks at same time
equipment under the pipe
insufficient time to put equipment away
too many tasks scheduled
Proximate Causes
Root Causes