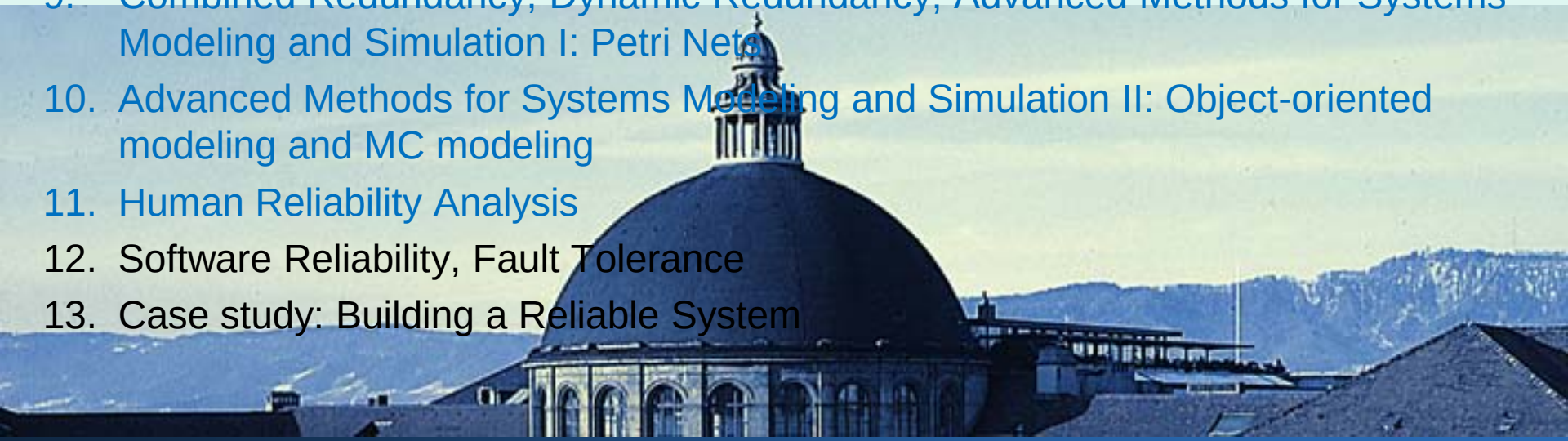
Reliability of Technical Systems

Main Topics

1. Short Introduction, Reliability Parameters: Failure Rate, Failure Probability, etc.
2. Some Important Reliability Distributions
3. Component Reliability
4. Introduction, Key Terms, Framing the Problem
5. System Reliability I: Reliability Block Diagram, Structure Analysis (Fault Trees), State Model.
6. System Reliability II: State Analysis (Markovian chains)
7. System Reliability III: Dependent Failure Analysis
8. Data Collection, Bayes Theorem, Static Redundancy
9. Combined Redundancy, Dynamic Redundancy; Advanced Methods for Systems Modeling and Simulation I: Petri Nets
10. Advanced Methods for Systems Modeling and Simulation II: Object-oriented modeling and MC modeling
11. Human Reliability Analysis
12. Software Reliability, Fault Tolerance
13. Case study: Building a Reliable System



Hardware and Software Faults

Mechanical / Hardware faults are design, manufacturing or operation faults.

They caused by disturbance or wear-out, where the physical rules are well-known. Therefore these faults and their impacts on the failure rate are better understood than software faults.

Software faults are (nearly always) design faults, which exist as latent faults during system operation.

Well tested programs may still contain a high number of design faults, but exhibit a low failure rate.

Countermeasures Against Software Faults

1.Removal of the failure:

- system reset (depends on environment)
  Failure can, but need not occur again.

2.Countermeasures against future failures:

- error by-passing: questionable input data and/or commands are no longer used and substituted as far as possible
- design fault tolerance by diverse design

3.Removal of the design fault:

- program correction (i. e. software repair): fault localization in the program, change of the program to remove the design fault and not to insert a new one

Reliability

Software is not destroyed by design fault and thus "repairable", in principle (program correction).

Reliability measure to be quantified:

- Failure rate z(t)

Reliability measures derived thereof:

- Availability V $V = \frac{\mu}{\lambda+\mu}$ where the failure rate z(t) = λ and the repair rate μ are constant (at least approximately for a limited period of operation)

- Reliability R(t)

  for time t from process start or restart, respectively: $R(t) = e^{-\int_0^t z(x)dx}$

  If failure rate z(t) = λ is constant: $R(t) = e^{-\lambda t}$

Fault Locations and their Correction (1/2)

number of fault locations in a program: s(t)

number of fault locations where a correction or improvement has been attempted (successful or not) b(t)

s(t) + b(t) ≥ s(0) holds where s(0) is the number of fault locations at the beginning of an operation
">" if some attempts have not been successful

improvement portion a

$$a = \frac{\text{number of improvement attempts}}{\text{number of failure events}} \leq 1$$

If an improvement is tried after each software failure a = 1 holds.

a > 1 is impossible, since we assume that faults can only be improved after they have been detected through a failure.

Fault Locations and their Correction (2/2)

It should be noticed that the number of fault locations in a program cannot always be quantified without ambiguity. Fault locations may overlap or spread over a collection of statements. Semantical faults can not be uniquely to a syntactical structure.

The improvement portion a is determined by the organization of software maintenance. During the test phase nearly all failures are analyzed and improved ($a = 1$). In a later operation phase, the user and the maintenance personnel are separated from each other. Therefor, the maintenance personnel usually collects a number of (possibly identical or similar) failure reports, before an improvement is attempted ($a < 1$).

Problems of Software Design Fault Modeling

In complex software the interactions between various modules cannot be covered by a simple model:

- A software system cannot be arbitrarily subdivided into independent components.

Consequently, software components can be "big" modules.

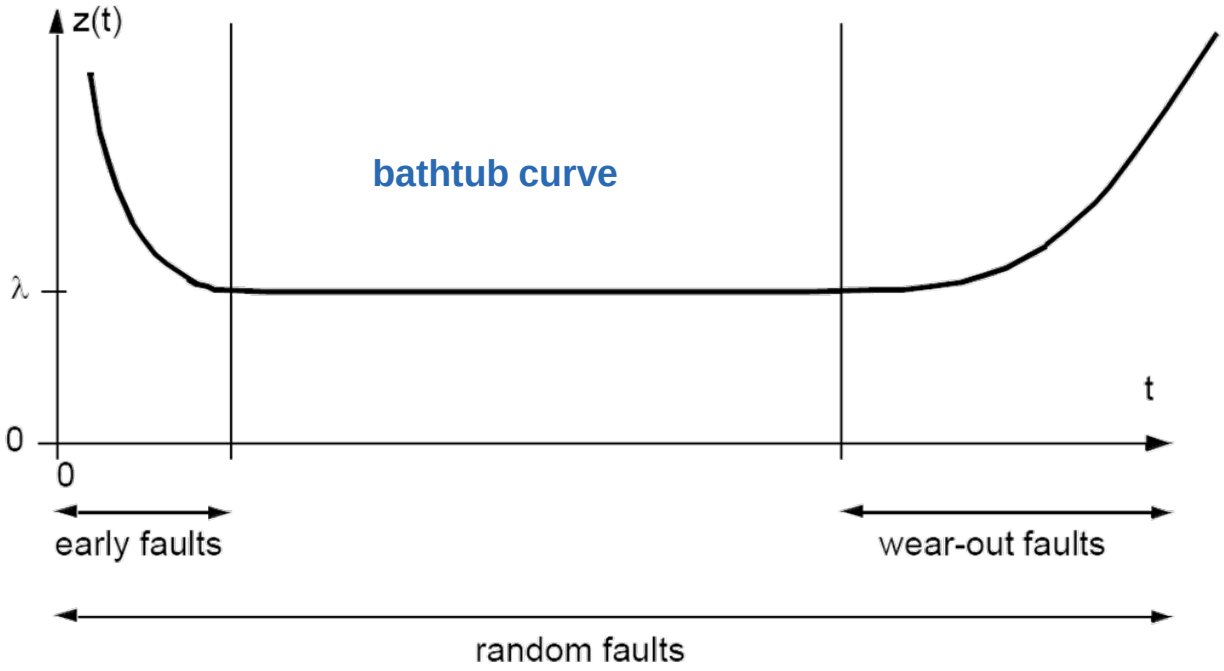
- The failure rate depends not only on the component (the program), but also on its **environment** (application, input data, load).

Consequently, reliability measures cannot be imported from one application area to another one.

- Due to the design fault improvements the failure rate is decreasing rather than constant.

A constant failure rate may only be assumed for the time interval between to program improvements.

Can software faults be modeled by analogy to hardware faults?



How can the three phases of the bathtub curve be interpreted in this case?

Modeling Software Faults by Analogy to Hardware Faults

The three phases of the so-called bathtub curve could also be meaningful for the software:

- Early faults characterize the design fault undiscovered during the testing phase and causing a failure during the early operation.
- Random faults occur sporadically in the middle of the software life time where "improvements" remove approximately the same number of design faults as are newly inserted into the software.
- Wear-out is completely impossible in the software, of course. However, when the environment changes during the time or the software is used for different applications, the failure rate may increase similarly to a wear-out.

On the one hand, there are analogies and similarities in the interpretation of early faults and random faults.
On the other hand, there are a lot of differences in the interpretation of the late phase.

Software Design Fault Models

Software faults deviate from hardware faults:

- The transition from the early to the middle phase is rather transient.
- The failure rate increases after major **program changes** (adding new functions) or major **environmental changes**.
- Otherwise, the failure rate decreases due to the **improvements** and the **experience** in the use of the program.
- The failure rate **cannot be quantified** as **exactly** as in the hardware due to changes in the input data, the load and/or the environment.

Hence, we will look for simple software design fault models to be applied to the phase **between two major program changes**.
The **decreasing failure rate** should be modeled accurately.

Design Redundancy

Usually, design faults (in both the software and the hardware) cannot be completely avoided – even by a high improvement effort. Examples:

- software faults in space applications (moon landing, Space Shuttle)
- software faults in avionics (passing the equator line, missile firing)

Usual structural redundancy is completely useless here, because it would replicate the design faults.

Instead, **diversity** is required (in other words: design redundancy):

A design is done multiply to generate multiple **variants**
(not to be mixed up with sequentially designed versions of a software).

Each variant is generated by a **separate team of designers**.

Objectives of Diverse Design

- increased probability to generate a correct variant
- majority of correct variants
- independent design faults in the variants, where the design faults are not activated by the same input data

Diversity does not guarantee that these goals are met.
Even a decrease in reliability is possible.

However, an improved reliability can be expected ("the principle of hope").

Diversity cannot substitute the efforts to make the software more perfect.

Summary: Benefit of Diversity

- Diversity is based on "the principle of hope". The benefit is difficult to quantify due to the dependence among design faults.
- Experiments show that diversity leads to an effective reliability improvement – at the cost of designing multiple variants.
- Usually the improvement factors are the higher the more reliable the single variants are.
- Diversity does not solve the so-called software crisis.
  In particular, purely increasing the number of ("low-cost") fault-prone variants does not provide reliable software.
- An increased software reliability should be primarily obtained by making the software more perfect (use of CASE tools, testing and improvement of detected design faults, etc.). After these methods have been exhausted diversity can be recommended for improving the reliability further.

Cost of Design Diversity

Compared to the non-diverse design the cost of n diverse variants is as follows:

- usually nearly the same specification cost
- approximately n-fold design cost for the implementation
- lower test cost than in the non-diverse case due to back-to-back testing * (more than 50% of the development cost may be caused by testing)
- increased cost for installation
- increased operation and maintainance cost
- approximately n-fold design cost for program modifications

In total, 50% additional cost has been estimated for n = 3 variants.

*Back-to-back testing involves cross-comparison of all responses obtained from functionally equivalent software components. Back-to-back testing can remain an efficient way of detecting failures even when the probability of identical and wrong responses from all participating versions, is very close to one.

Software Design Fault Models

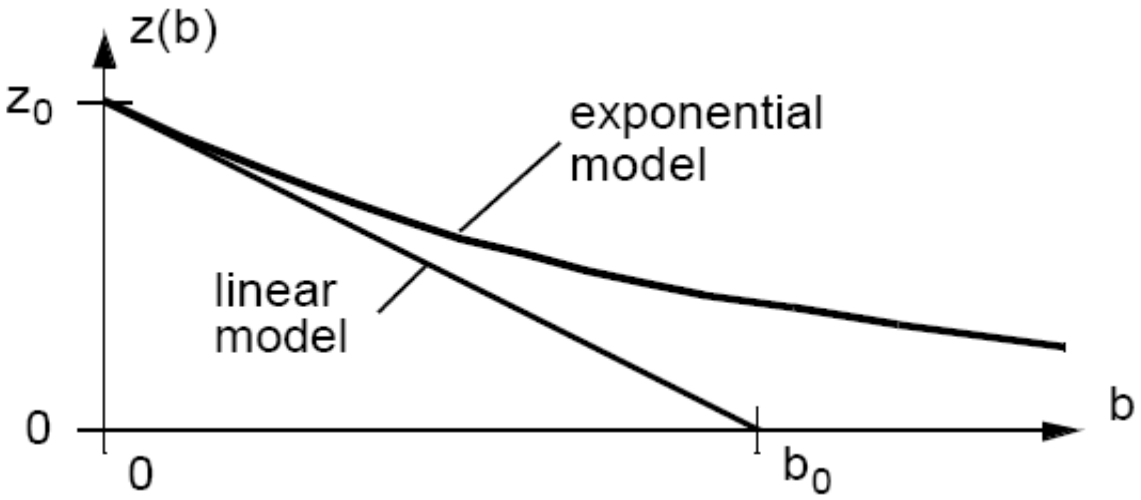
Software reliability measures can be derived from:

- **Failure rate** depending on the **time** (calendar time, processing time)
- **Failure frequency** depending on the **number of program executions**

Two simplified fault models:

Linear Model (somewhat optimistic)

Exponential Model (somewhat pessimistic)



z(b) failure rate, b number of fault locations

Comparison of Linear and Exponential Model

linear model (somewhat optimistic):

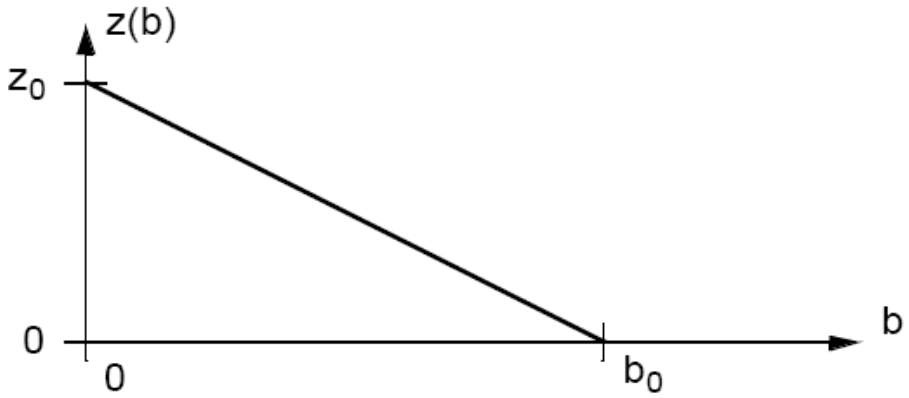
- program is perfect after all b_0 design fault locations have been corrected.
- The probability that an improvement attempt is successful is the same for all design fault

exponential model (somewhat pessimistic):

- The last ("hidden") design faults can be corrected successfully with lower probability.
- By an improvement attempt also new design faults can be inserted into the program.

Linear Model

The failure rate decreases linearly with the number of improvements.
Initial failure rate = z_0. Number of necessary improvements = b_0.



$$z(b) = z_0\left(1 - \frac{b}{b_0}\right)$$

depending on the number of improvements,
not on the time !

Simple model, frequently used.

Time-Dependent Number of Improvements in the Linear M.

From the strictly monotonic function $t(b) = \frac{b_0}{az_0} \cdot (-\ln(b_0 - b) + \ln(b_0))$

we can derive the reverse function b(t):

$$\frac{az_0 t}{b_0} - \ln(b_0) = -\ln(b_0 - b(t))$$

$$\ln(b_0 - b(t)) = \ln(b_0) - \frac{az_0 t}{b_0}$$

$$b_0 - b(t) = b_0 e^{-\frac{az_0}{b_0} t}$$

$$b(t) = b_0 \left(1 - e^{-\frac{az_0}{b_0} t}\right)$$

The reverse function b(t) shows that in the linear model the number of improvements does **not** linearly increase with the time.
The increase is only negative exponential because improved programs fail more rarely.
Hence further improvements are required more rarely.

Time-Dependent Failure Rate in the Linear Model

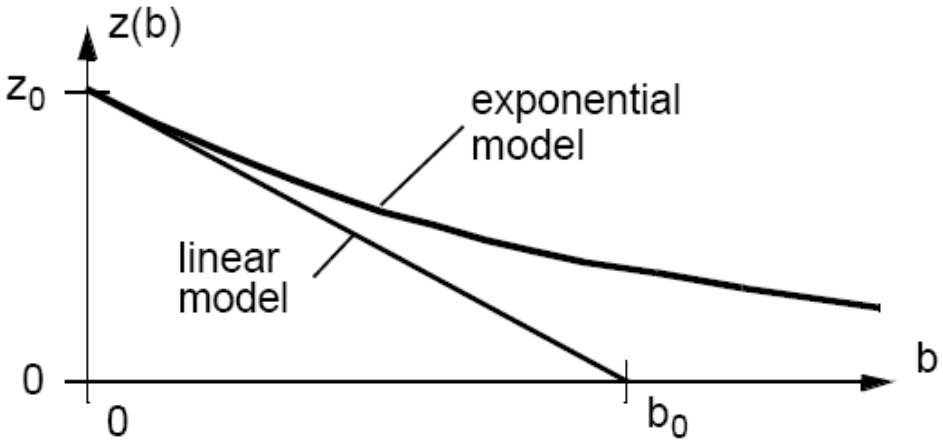
Substituting b(t) in z(b) yields:

$$z(t) = z(b(t)) = z_0\left(1 - \frac{b_0\left(1 - e^{-\frac{az_0}{b_0}t}\right)}{b_0}\right) = z_0 e^{-\frac{az_0}{b_0}t}$$

The time-dependent z(t) in the linear model is a negative exponential function – the same as b(t). At any time z(t) > 0 holds.

Exponential Model

The failure rate decreases negative exponentially with the number of improvements (ccording to function parameter c). Initial failure rate = z_0.



$z(b) = z_0 e^{-cb}$ The initial "success" of an improvement is equal in both the linear and the exponential model, if:

$$\frac{dz_{linear}(0)}{db} = \frac{dz_{Exp}(0)}{db} \Leftrightarrow z_0\left(-\frac{1}{b_0}\right) = z_0(-ce^{-cb}) \Leftrightarrow c = \frac{1}{b_0}$$

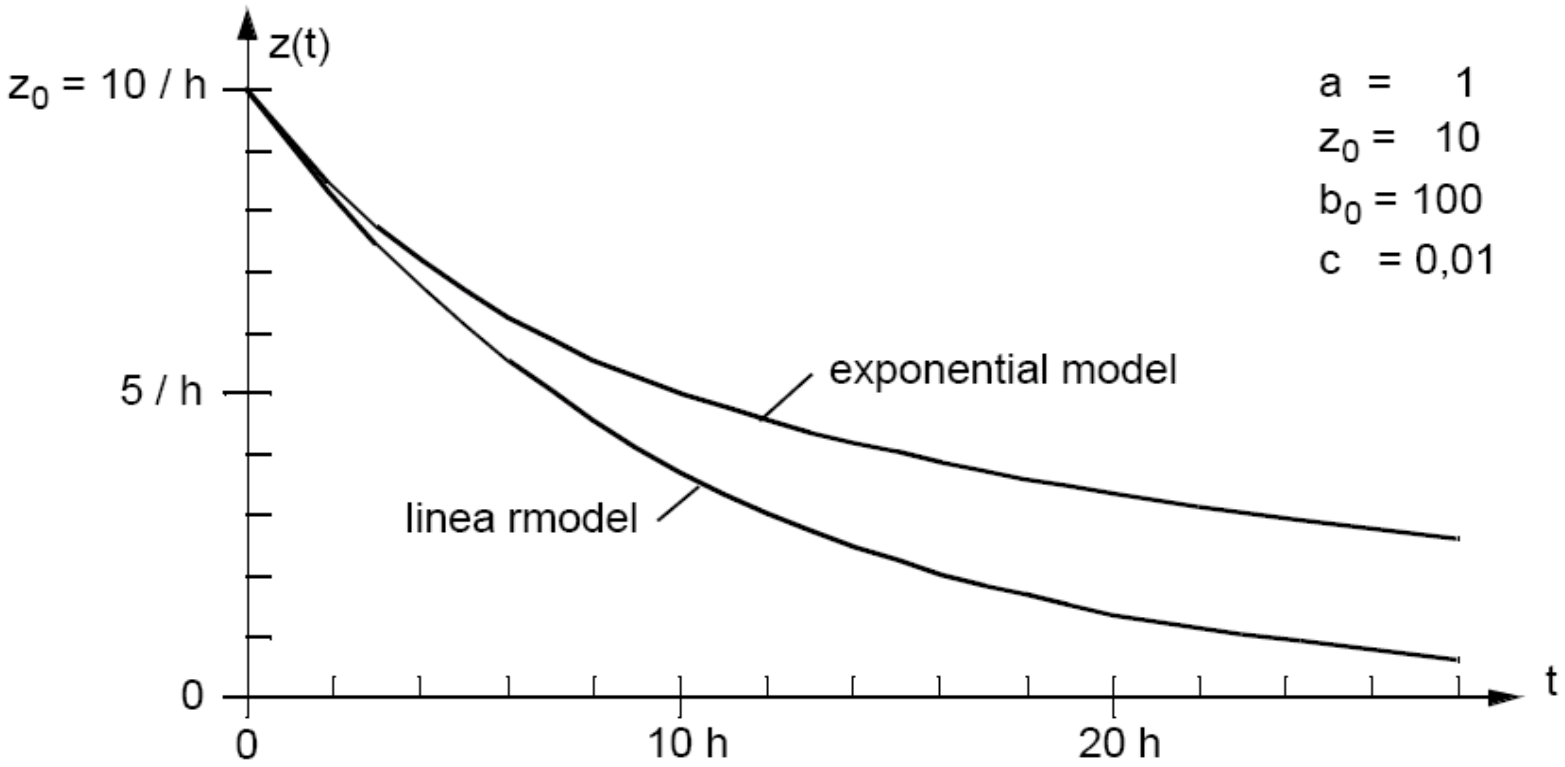
Time-Dependent Failure Rate in the Exponential Model

Substituting b(t) in z(b) yields:

$$z(t) = z(b(t)) = z_0 e^{-c \cdot \frac{1}{c} \cdot \ln(caz_0 t + 1)} = \frac{1}{cat + \frac{1}{z_0}}$$

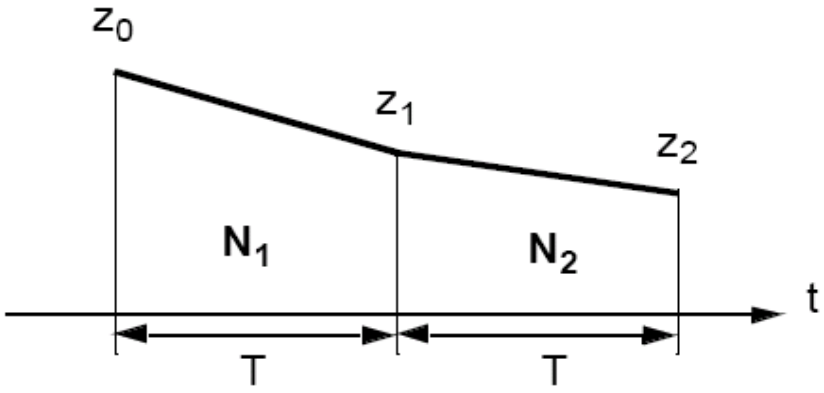
For the time-dependent failure rate z(t) in the exponential model $z(t) > 0$ holds at any time – the same as in the linear model.

Time-Dependent Failure Rate in the Linear and the Exponential Model



Linear Model: Determine Parameters for Given Software

Execute software for a duration of 2T, improve design faults on failure occurrence (a = 1), numbers of improvements N_1 and N_2, respectively.



4 equalities, 4 variables: z_0, b_0, z_1, z_2, to be solved for: z_0, b_0

depending on N_1, N_2: $z_1 = z_0\left(1 - \frac{N_1}{b_0}\right)$ $z_2 = z_0\left(1 - \frac{N_1 + N_2}{b_0}\right)$

depending on T: $z_1 = z_0 e^{-\frac{a z_0}{b_0} T}$ $z_2 = z_0 e^{-\frac{a z_0}{b_0} 2T}$

Linear Model: Determine Parameters for Given Software

By equating the expressions and reducing by z_0 we obtain:

left: $1 - \frac{N_1}{b_0} = e^{-\frac{az_0}{b_0}T}$ right: $1 - \frac{N_1 + N_2}{b_0} = e^{-\frac{az_0}{b_0}2T}$

By squaring the left equality and equating we obtain:

$$\left(1 - \frac{N_1}{b_0}\right)^2 = 1 - \frac{N_1 + N_2}{b_0} \quad \Leftrightarrow \quad (b_0 - N_1)^2 = b_0^2 - N_1 b_0 - N_2 b_0$$

$$\ldots \quad \Leftrightarrow \quad b_0 = \frac{N_1^2}{N_1 - N_2}$$

Applying the logarithm to the equality at the top left yields:

$$\ln\left(1 - \frac{N_1}{b_0}\right) = -\frac{az_0}{b_0}T \quad \Leftrightarrow \quad z_0 = \frac{b_0}{aT}\ln\left(\frac{b_0}{b_0 - N_1}\right) = \ldots = \frac{b_0}{aT}\ln\left(\frac{N_1}{N_2}\right)$$

After the parameters z_0 and b_0 have been determined, the linear model for the given software is complete.

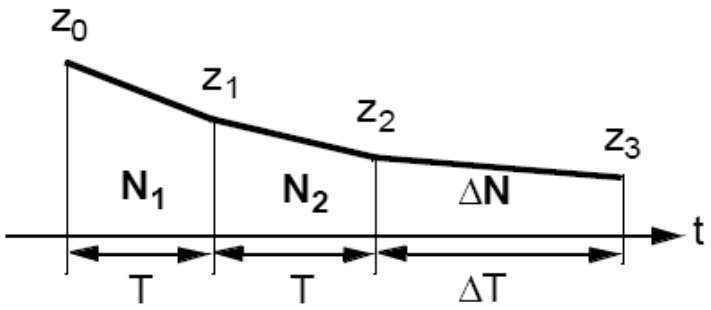
Linear Modell: Expense to Reduce the Failure Rate

From the parameters z_0 and b_0 we obtain the failure rates z_1, z_2:

$$z_1 = z_0\left(1 - \frac{N_1}{b_0}\right) = \ldots = z_0\frac{N_2}{N_1} = \frac{N_1 N_2}{aT(N_1 - N_2)}\ln\left(\frac{N_1}{N_2}\right)$$

$$z_2 = z_0\left(1 - \frac{N_1 + N_2}{b_0}\right) = \ldots = z_0\frac{N_2^2}{N_1^2} = \frac{N_2^2}{aT(N_1 - N_2)}\ln\left(\frac{N_1}{N_2}\right)$$

After which number of improvements ΔN and which time ΔT the failure rate can be reduced from z_2 at time $2T$ down to $z_3 < z_2$?



Linear Model: What time is required to detect and to improve ΔN design faults?

required runtime ΔT:

$$z_3 = z_0 e^{-\frac{a z_0}{b_0}(2T + \Delta T)} \quad \Leftrightarrow \quad \ln\left(\frac{z_3}{z_0}\right) = -\frac{a z_0}{b_0}(2T + \Delta T)$$

$$\Leftrightarrow \quad \Delta T = \frac{b_0}{a z_0} \ln\left(\frac{z_0}{z_3}\right) - 2T$$

Linear Model: How many improvements must be made in order to reduce the failure rate from z_2 down to z_3?

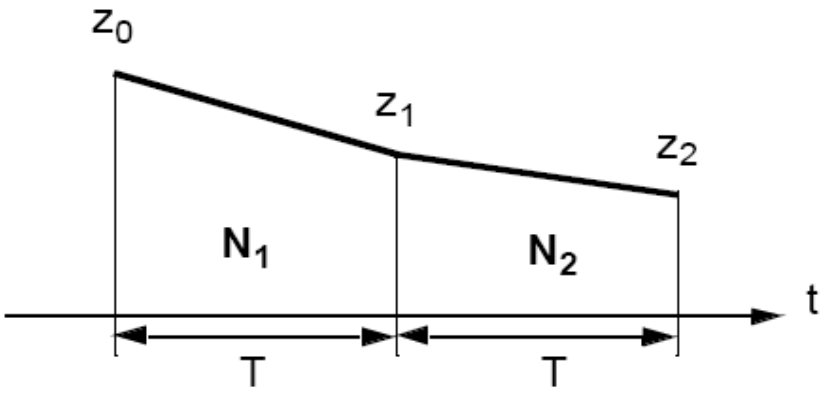
Number of improvements ΔN:

$$z_3 = z_0\left(1 - \frac{N_1 + N_2 + \Delta N}{b_0}\right) \quad \Leftrightarrow \quad \frac{N_1 + N_2 + \Delta N}{b_0} = 1 - \frac{z_3}{b_0}$$

$$\Leftrightarrow \quad \Delta N = \left(1 - \frac{z_3}{z_0}\right) \cdot b_0 - N_1 - N_2$$

Expon. Model: Determine Parameters for Given Software

Execute software for a duration of 2T, improve design faults on failure occurrence (a = 1), numbers of improvements N_1 and N_2, respectively.



4 equalities, 4 variables: z_0, c, z_1, z_2, to be solved for: z_0, c

depending on N_1, N_2: $z_1 = z_0 e^{-cN_1}$ $z_2 = z_0 e^{-c(N_1+N_2)}$

depending on T: $z_1 = \frac{1}{caT + \frac{1}{z_0}}$ $z_2 = \frac{1}{ca2T + \frac{1}{z_0}}$

Expon. Model: Determine Parameters for Given Software

By equating the expressions and reducing by z_0 we obtain:

left: $caz_0T = e^{cN_1} - 1$ right: $caz_02T = e^{c(N_1+N_2)} - 1$

By doubling the left equality and equating we obtain:

$2e^{cN_1} - 1 = e^{cN_1+cN_2}$ Solve $x^k - 2x + 1 = 0$ for $k = 1 + \frac{N_2}{N_1}$

where $x = e^{cN_1}$ and $c = \frac{\ln(x)}{N_1}$ (see table)

By substituting in the equality at the top left and in z(b) = ... we obtain:

$z_0 = \frac{1}{caT}\left(e^{cN_1} - 1\right)$ $z_1 = z_0 e^{-cN_1}$ $z_2 = z_0 e^{-c(N_1+N_2)}$

The following table gives the dependence between k and ln(x),
where we only must take into account the solutions x > 1 because others are not meaningful here:

k	ln(x)	k	ln(x)	k	ln(x)	k	ln(x)
1.12	5.76313	1.34	1.77944	1.56	0.76466	1.78	0.28061
1.14	4.92507	1.36	1.64287	1.58	0.70692	1.80	0.24897
1.16	4.28916	1.38	1.51898	1.60	0.65266	1.82	0.21880
1.18	3.78754	1.40	1.40600	1.62	0.60158	1.84	0.19000
1.20	3.37987	1.42	1.30250	1.64	0.55341	1.86	0.16248
1.22	3.04071	1.44	1.20731	1.66	0.50791	1.88	0.13618
1.24	2.75319	1.46	1.11943	1.68	0.46487	1.90	0.11101
1.26	2.50570	1.48	1.03804	1.70	0.42410	1.92	0.08691
1.28	2.28994	1.50	0.,96242	1.72	0.38544	1.94	0.06381
1.30	2.09985	1.52	0.89199	1.74	0.34873	1.96	0.04166
1.32	1.93085	1.54	0.82622	1.76	0.31383	1.98	0.02041

For we substitute $k = 1 + N_2 / N_1$. Parameter c is calculated by: $c = \ln(x) / N_1$.

Exponential Model: How many improvements must be made in order to reduce the failure rate from z_2 down to z_3?

- **number of improvements ΔN:**

$$z_3 = z_0 e^{-c(N_1 + N_2 + \Delta N)} \quad \Leftrightarrow \quad \ldots \quad \Leftrightarrow \quad \Delta N = \frac{1}{c} \ln\left(\frac{z_0}{z_3}\right) - N_1 - N_2$$

Exponential Model: What time is required to detect and to improve ΔN design faults?

- **required runtime ΔT:**

$$z_3 = \frac{1}{ca(2T + \Delta T) + \frac{1}{z_0}} \quad \Leftrightarrow \quad \ldots \quad \Leftrightarrow \quad \Delta T = \frac{1}{ca}\left(\frac{1}{z_3} - \frac{1}{z_0}\right) - 2T$$

Example Task

For the control of a continuous physical process a program is required with a failure rate of $z_3 = 4 \cdot 10^{-3}/\text{h}$ at the most.

During a test duration of T = 720 h (approximately 1 CPU month)
N_1 = 40 failures, after a further test duration of T = 720 h
N_2 = 22 failures detected and improved
(improvement portion a = 1).

Questions:

- What are the **parameters** of the linear and the exponential model ?
- What is the **failure rate z_2** after the end of the test duration 2T = 1,440 h ?
- How many **improvements** ΔN and which **runtime** ΔT are necessary to reach the required failure rate z_3 ?

Literature

Echtle, K. Fault-Tolerant Software Systems, Lecture-Script, Informatik, University of Duisburg-Essen.

D. Musa, A. Iannino, and K. Okumoto. Software Reliability: Measurement, Prediction, Application. McGraw-Hill, 1987. ISBN 0-07-044093-X