Test and Dependability 2: Test, Fault Elimination and Fault Prevention

Prof. G. Kemnitz

November 23, 2026

Contents

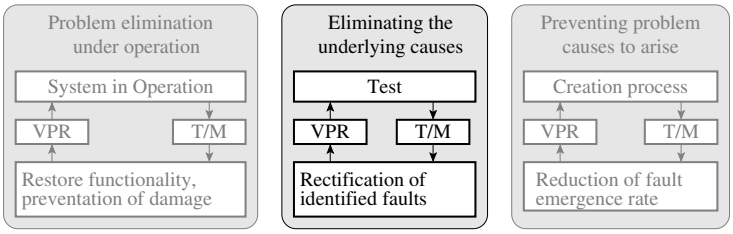
1 Test **2**
1.1 The Wide Variety of Tests 2
1.2 Measures 4
1.3 Fault Model, stuck-at faults 5
1.4 Criterion coverage 10
1.5 Yield, Defect Level 14
1.6 Isolated module test 15

2 Fault elimination **19**
2.1 Experimental Repair 19
2.2 Fault diagnosis & isolation 20

3 Reliability & Test **23**
3.1 Simple estimation 23
3.2 Improved model 25
3.3 Interpretation, simulation 28
3.4 Pre- and reliability tests 30
3.5 Reliability and Safety 31
3.6 Effective number of tests 33
3.7 Maturation process 35
3.8 Usage and test profiles 38

4 Fault prevention **44**
4.1 Maturation of processes 47
4.2 Process capabilities 49
4.3 Centring, improvement 50
4.4 Process models 51
4.5 Quality and Creativity 53

2.2 Eliminating the root causes of problems



Classification according to rectification options:

- **Fault**: A single, rectifiable cause of malfunction. The success of the rectification can be verified by repeating the test.
- **Disruption**: The opposite of a fault. No single, rectifiable cause. No success check with a yes/no result. Only a reduction in frequency is possible, with improvement check by lots of test repetition.
- **Failure**: Faults occurring after deployment when the system is already in use.

T/M, VPR yTesting and monitoring for problems. Verification of problem resolution.

2.3 Elimination iteration and fault definition

Faults are rectifiable causes of malfunctions and crashes.

Fault elimination iteration: If a test fails, check whether the fault behaviour is the same when the test is repeated. If so, fault elimination in an iteration of repair attempts and repeated testing until success (sec.2.2).

Eliminated faults can be described as the changes made to rectify them. If, for example, a case distinction was added, the fault was a missing case distinction. Generally several options.

Potential faults: Possible changes in the fault-free system that caused a malfunction. Reversely repair hypothesis/fault diagnosis, what changes might be the causes?

For **hardware production testing**, there is a fault-free description of fault modelling. For software, it is only after the bugs have been rectified, for retrospective assessment of the test quality and the level of reliability achieved.

1 Test

2.4 Testing

Procedures for detecting faults. Basic classification:

- Static tests: Checking features directly.
- Dynamic testing: Checking the functionality with a samples of test inputs.

Static tests for:

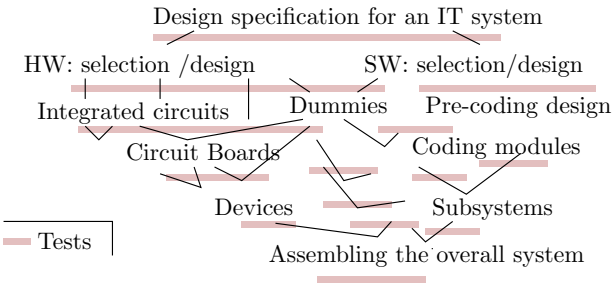
- Documentation: clarity, completeness, ...
- Software: syntax, design rules, type compatibility, API usage, ...
- Printed circuit boards: Resistance along and between lines to detect short circuits and open circuits.

Dynamic tests can only be applied to (sub-)systems that are already operational.

IT systems are subject to many different static and dynamic tests before they are put into use.

1.1 The Wide Variety of Tests

2.5 Design and Testing



There isn't just one test, but rather a large number of different tests for different parts and aspects.

IT design is a complex process in which a modular system of hardware and software is created. Throughout the design stages and upon completion, **a wide range of static and dynamic tests are carried out**, both on individual components and on the system as a whole.

Dummy Simplified substitute for components not yet designed, for testing purposes.

2.6 Test-driven design

The design results from the initial phases (collections of requirements, solution ideas, decisions) are tested for feasibility, comprehensibility, consistency, etc., usually statically through inspection (sec.5.1), and in exceptional cases dynamically through simulation.

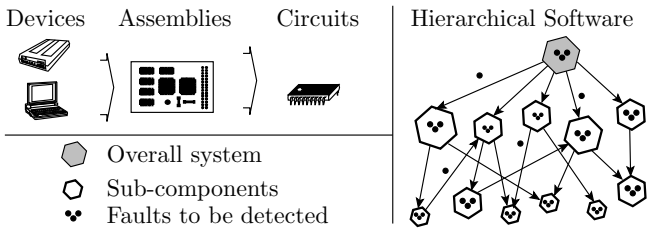
Dynamic testing is only possible once the description can be executed with input/output examples:

- Preferably isolated from external influences (container test).
- Hierarchical, starting with module or component tests, followed by testing within the higher-level system.
- Start by doing everything roughly to eliminate the majority of the most serious faults. Then, perform many more tests to eliminate the ever-decreasing fraction of faults, each of which causes fewer and fewer malfunctions$^{(PP)}$.

Containertest: Tests als deterministischer Service in einer isolierten Testumgebung.

$^{(PP)}$ Pareto-Prinzip: Ein kleiner Teil der Ursachen ist für den größten Teil der Wirkungen verantwortlich. Wenige Tests decken die überwiegende Mehrheit der Fehler auf. Ein kleiner Teil der Fehler verursacht die überwiegende Mehrheit der Fehlfunktionen. ....

2.7 Hierarchy and Isolated Testing



IT systems are organised hierarchically. The tests follow this hierarchy. Printed circuit boards are made from thoroughly tested circuits. For software, test starts with modules. Fault removal. Then integration (interface) tests, ... Finally, testing of the entire system.

Thorough testing of components and modules prior to their integration into the larger system, where testing is usually restricted to faults in interaction.

Isolated component tests promise higher fault coverage with less test than embedded in the higher-level system (see section below, 2.3.6).

2.8 Failures, Maintenance Tests

A **hardware failure** is a new fault that arises during the operational phase. Just like faults that arise during manufacture or repair, failures can impair functionality to varying degrees, from rarely to completely.

Serious failures lead to system failure. Before further use, repair is required. Loss of availability.

Handling: Safe state, repair. Reduction in downtime through Spare parts stock management and redundancy (sec.6.6).

Minor failures increase the malfunction rate and reduce reliability, without affecting availability. The threshold between minor and serious is a matter of judgement.

Handling: Reducing decline in reliability through regular retesting:

- **Power-on test** on every restart.
- **Maintenance tests**, e.g. for motor vehicles: MOT every two years (slide 6.127).

2.9 Rough Test, Repair Test, Regression Test, ...

Rough Test: Testing the functionality of a newly designed or manufactured component usually begins with a small number of samples. This identifies most of the faults, and in particular the most serious ones.

Reliability Test: Extensive and thorough testing to identify and rectify even those faults which, individually, have only a minor impact on reliability. The aim is to achieve a high level of reliability and safety.

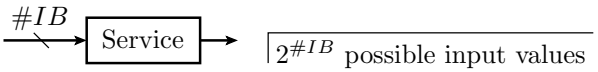
Repair Test: A test for observable fault behaviour (crash, malfunction, violated assertion, etc.) to verify success following each attempt to rectify the fault.

Regression Test: Successful tests (rough or reliability tests) that run without any problems. Required to check for new faults following fault rectification and further development.

Reliability is determined by the most thorough test, i.e. the reliability tests. Hence the name.

1.2 Measures

2.10 Functional Test Coverage?



Hypothetical test duration with all possible input values:

	$\#IB$	$2^{\#IB}$	t_T
Gate, 4 inputs	4	16	16 µs
ALU, 68 inputs	68	$3 \cdot 10^{20}$	10^7 years
fn(x:i32[4])->tY	128	$3 \cdot 10^{38}$	10^{25} years(1)

t_T – Test duration, where each individual test takes $1\mu s$.

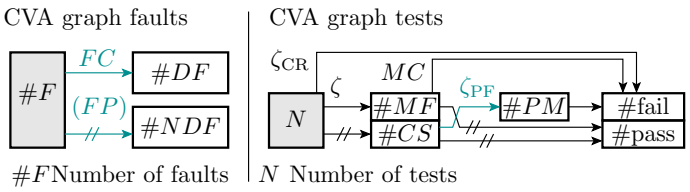
IT systems process and store thousands of input bits. The share of possible input valuess that can be verified before use and also during the whole lifespan is almost zero.

Functional coverage is not a useful measure of test quality.

$\#IB$ yNumber of input bits.

(1) Estimated time since the Big Bang: just $4 \cdot 10^9$ years.

2.11 Appropriate measures for test quality



- Fault Coverage and fault permeability:

$$FC = \left.\frac{\#DF}{\#F}\right|_{SLC} \tag{2.1}$$

$$FP = 1 - FC \tag{2.2}$$

- Phantom malfunction rate during the test. Proportion of failed tests whilst the object under test is behaving correctly:

$$\zeta_{PF} = \left.\frac{\#PM}{\#CS}\right|_{SLC} \tag{2.3}$$

FC, FP yFault coverage. Fault permeability. (Proportion of (un)detectable faults).

$\#DF$ yNumber of detectable faults.

$\#NDF$ yNumber of undetectable faults.

ζ_{PF} yPhantom malfunction rate of the test.

$\#CS$,$\#PM$ yNumber of correct services. Number of phantom malfunctions.

2.12 Phantom faults

Phantom faults are the cause of reproducible failing repair tests even though the test object behaves correctly. Causes include a faulty test execution function or incorrect test example values, e.g. incorrect setpoints or too small compare windows for results with rounding errors. If not recognised as a phantom fault

- unnecessary attempts to rectify faults within the test object,
- where new faults may arise.

Preventing problems:

- If a test fails, check the result to make sure it is actually incorrect.
- Debugging in the test execution function or test data.
- Success monitoring.

Dealing sensibly with phantom faults results in a negligible impact on reliability.

2.14 Selection of dynamic tests

At the time the tests are selected, the faults to be found and, after the test, those not found are unknown. Predicting fault coverage requires assumptions about possible faults or coverage criteria. A Key issue is whether these assumptions are used only for evaluation or also for selection:

- Evaluation only: Random selection without regard to fault detection conditions (dumb random tests).
- Also selection: Targeted search to detect alleged faults or to ensure that criteria are met.
- Hybrid approaches, such as fault-driven search for input profiles, to adjust the random selection (smart random tests).

Dump random patterns require orders of magnitude longer test sets to achieve the same coverage, compared with targeted searches. In return, they offer significantly higher and more accurately predictable actual fault coverage. Hybrid forms produce shorter test sets whilst still offering usefully predictable actual fault coverage.

1.3 Fault Model, stuck-at faults

2.15 Fault Modelling

The faults to be found are unknown at the time of test selection. Test selection and evaluation requires fault models.

Fault model: Algorithm that drives a set of model faults from a description of the test object.

Model Fault: minor alteration (**mutation**) of the description.

Fault-oriented test evaluation:

```
Repeat for each model fault:
    Count the detections.
```

Fault-oriented test search:

```
Repeat for each model fault:
    Find inputs to confirm them.
```

For hardware, **stuck-at-faults** are generally used, whilst for software, reachability criteria are preferred (see the following section).

2.16 The stuck-at fault model

For a circuit consisting of logic gates, the stuck-at fault model generates two model faults (description mutations) for each gate connection:

- Value is always zero (sa0, stuck-at-0) and
- Value is always one (sa1, stuck-at-1).

The initial fault set is cleansed of identical and implicitly verifiable model faults, as well as of redundant (non-verifiable) model faults.

Key characteristics for the usefulness of fault models in general and of stuck-at fault in particular:

- Costs of fault simulation and test generation (sec.6.2),
- Order of magnitude of the probability p_{ij} that tests for model faults will detect similar verifiable actual faults (sec.3.2.3).

For decades used for testing digital circuits. Here as fault model example to illustrate the general relationships between model faults and actual faults.

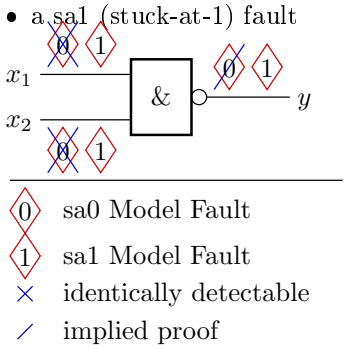
---

p_{ij} yProbability that a test for model fault j detects fault i.

2.17 Stuck-at Faults for a single Logic Gate

For each gate connection, two model faults are assumed:

- a sa0 (stuck-at-0) fault and
- a sa1 (stuck-at-1) fault



x_2	x_1	$\overline{x_2 \wedge x_1}$	sa0(x_1)	sa1(x_1)	sa0(x_2)	sa1(x_2)	sa0(y)	sa1(y)
0	0	1	1	1	1	1	0	1
0	1	1	1	1	1	0	0	1
1	0	1	1	0	1	1	0	1
1	1	0	1	0	1	0	0	1

identically detectable (identical proof set)

⋯→ Implies proof

corresponding input is element of proof set

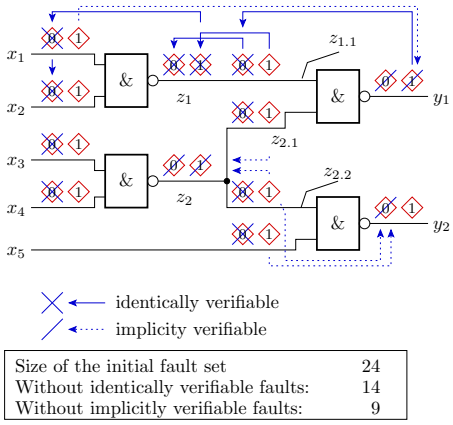
In the process, identical and implicitly proofable model faults arise at each gate. Identical faults are detectable by the same sets of inputs. Implicit proof of fault A when fault B is detectable means that the proof set for B is a subset of that for A.

sa0(x_1), sa0(x_2) and sa1(y) are identically provable; they all invert the gate output for the same set of inputs $x_1x_0 \in \{11\}$. Implicit proof of sa0(y) with the proof set $x_1x_0 \in \{00, 01, 10\}$ via sa1(x_1) and sa1(x_2), whose proof sets $\{01\}$ and $\{10\}$ are subsets of $\{00, 01, 10\}$ respectively.

Of identically verifiable faults, just one is sufficient, and implicitly verifiable faults are generally not used for test case generation and evaluation either.

Of the six initial model faults, three remain for which there is no identical or implicit proof.

2.18 Within a circuit

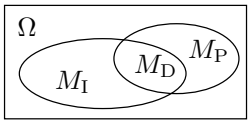


Size of the initial fault set	24
Without identically verifiable faults:	14
Without implicitly verifiable faults:	9

	Identically prooofable subsets of model faults	Proofe implied by
1	sa0(x_1), sa0(x_2), sa1(z_1), sa1($z_{1.1}$)	
2	sa1(x_1)	
3	sa1(x_2)	
4	sa0(x_3), sa0(x_4), sa1(z_2)	9, 12
5	sa1(x_3)	
6	sa1(x_4)	
7	sa0(z_2)	5, 6, 8, 11
8	sa0(z_1), sa0($z_{1.1}$), sa0($z_{2.1}$), sa1(y_1)	2, 3
9	sa1($z_{2.1}$)	
10	sa0(y_1)	1, 9
11	sa0($z_{2.2}$), sa0(x_5), sa1(y_2)	
12	sa1($z_{2.2}$)	
13	sa1(x_5)	
14	sa0(y_2)	12, 13

Identical proof on unbranched lines. Implicit detection of faults at outputs through the corresponding model faults at line branches.

2.20 Detection condition



Ω Input Space
M_I Infection Set
M_P Propagation Set
M_D Detection Set

Fault detection in a digital circuit can be divided into

- an infection condition and
- a propagation (stread) condition.

Inputs can be searched for separately for both. Detection then requires that the inputs found satisfy both conditions.

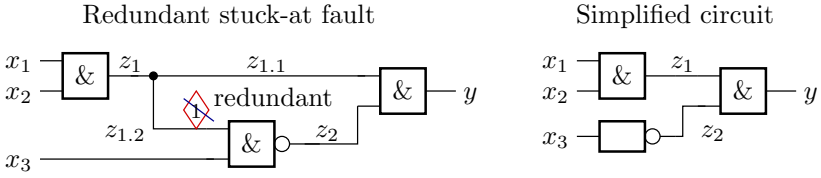
Can also be described using sets. In the figure, M_I is the set of all inputs that satisfy the infection conditions, and M_P is the set of inputs for which local bit flipping is observable (propagated) to an output. The detection set is the intersection M_D of these two sets.

Can be further extended to probability estimates.

Infection: Local value corruption during execution.
Propagation: Mapping of local corruption onto an observable ouputs.

2.21 Redundant Faults

Redundant faults are mutations of the system description which do not alter its functionality and are therefore not detectable by dynamic testing.



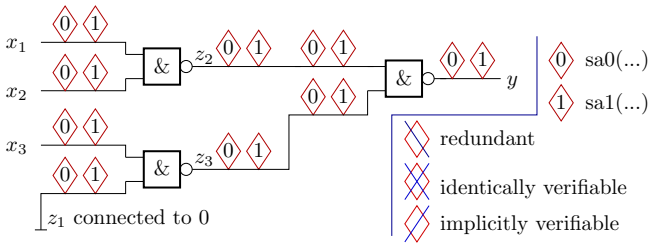
Infection requires $z_1 = 0$, and spread from z_2 to y requires $z_2 = 1$. Mutual exclusion.

Eliminating redundant faults also serves to simplify the circuit. This principle can also be applied to other systems.

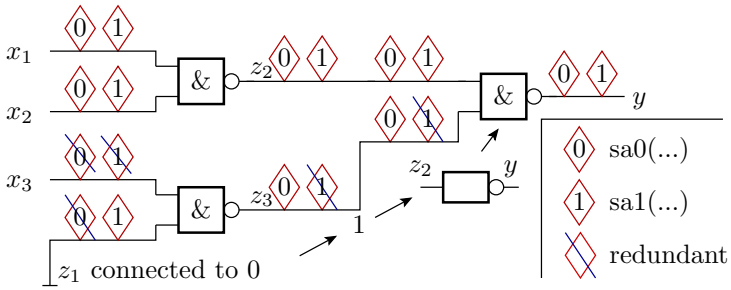
Infection: Local value corruption during execution.
Propagation: Mapping of local corruption onto an observable ouputs.

Example 2.1 Stuck-at Faults

Circuit with 18 marked stuck-at faults:

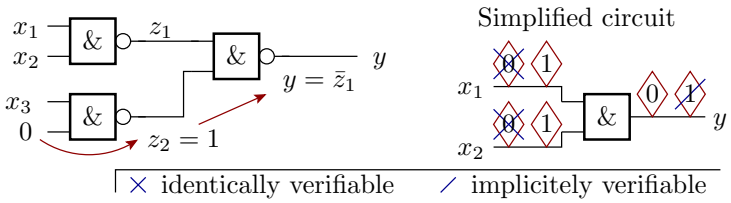


a) *Remove all marked stuck-at faults that are redundant.*



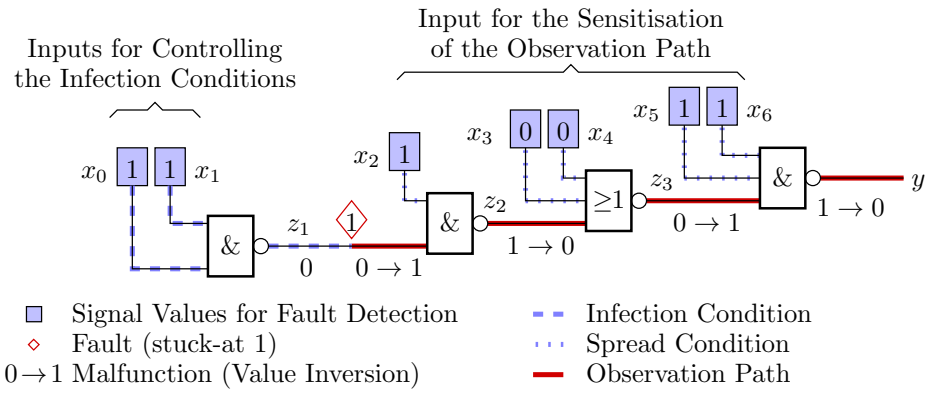
b) *Show a simplified circuit diagram without redundant stuck-at faults, remove identically detectable faults and mark the implicitly detectable stuck-at fault.*

The function does not depend on x_3 and is: $y = x_1 \wedge x_2$



At the remaining AND gate, it can be shown that sa0(x_i) is identical to sa0(y) and that the proof of sa1(x_1) or sa1(x_2) implies that of sa1(y).

2.23 Test Search

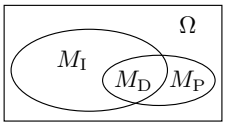


Search using path sensitisation (sec. 6.2.2):

- Search for inputs that set the value at the fault location to »0« and
- Sensitise a tracing path to an output.

In the example, the solution is simple. With thousands of wires, this is no longer the case, but there is software that can do that.

2.24 Separation into infection and spread



Ω	Input space:	$2^7 = 128$ values
M_I	Infection set:	2^5 of the 128 values
M_P	Spread set:	2^2 of the 128 values
M_D	Detection set:	1 of the 128 values

Infection condition: $x_1 = x_0 = 1$, all others arbitrary:

$$c_I = x_1 x_0$$

Spread condition: $x_6 = x_5 = \bar{x}_4 = \bar{x}_3 = x_2 = 1$, the rest arbitrary:

$$c_P = x_6 x_5 \bar{x}_4 \bar{x}_3 x_2$$

Detection condition: AND of the infection and propagation conditions:

$$c_D = c_I \wedge c_P = x_6 x_5 \bar{x}_4 \bar{x}_3 x_2 x_1 x_0$$

The sizes of the sets:

- Infection: 5 arbitrary bit values, 2^5 possibilities
- Spread: 2 arbitrary bit values, 2^2 possibilities
- Detection: only one input value: $x_6 \cdots x_0 = 1100111$

2.25 Probability of fault detection

With random inputs, infection, spread and detection are random events. If all 2^7 possible inputs occur with equal frequency:

Probability of infection:	$p_I = 2^{-2}$
Probability of transmission:	$p_P = 2^{-5}$
Probability of detection:	$p_D = p_I \cdot p_P = 2^{-7}$

The actual faults are unknown, but they share verification conditions with model faults. Tests designed to detect similar verifiable model faults are far more likely to detect the real faults than random inputs, selected without any assumptions about possible faults. A targeted search for tests can only give chance a little help.

---

p_I, p_P, p_D yProbability of infection, propagation and detection.

p_i yDetection probability for a fault i with a randomly selected test.

1.4 Criterion coverage

2.26 Fault modelling for software

Programme mutation:

- Off-by-one faults, e.g. for a value assignment: (y=a+x ⇒ y=a+x+1),
- minor changes of arithmetic expressions: (x=a+b ⇒ x=a*b),
- falsification of a Boolean expression: (if(a>b){} ⇒ if(a<b){}),
- wrong address assignment (ref=obj1 ⇒ ref=obj2),
- removal of a keyword (static int x=5 ⇒ int x=5), ...

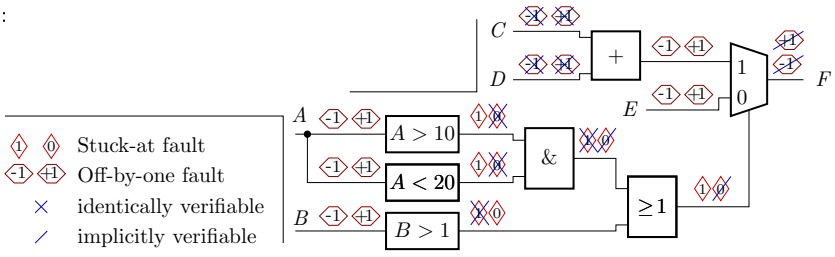
Many more options than with digital circuits. Requires restriction:

1. Off-by-One faults: Based on the stuck-at-fault model, for each input and output there are two model faults, value is slightly too large and value is slightly too small (for integer values ∓1).
2. A random sample of typical faults (fault injection),
3. **Criterion coverage:** only checks reachability [and, optionally, infection], but not propagation (spread of local falsifications).

Further reading: [6, 2]

2.27 Off-by-one faults

```
if (A>10 and A<20) or B>1:
  F=C+D;
else:
  F=E;
```



Inspired by the the stuck-at-fault model. Two model faults for all inputs and outputs of each operation:

- ⟨+1⟩ Value slightly too large and
- ⟨−1⟩ value slightly too small.

Cleaning up from initial identival, implicitely proofable and redundant (undetectable) faults.

Similar to, but not the practical significance of stuck-at faults.

2.28 Fault injection

Injection of a **sample** of typical faults or distortions

- Programme code mutation (see two slides earlier),
- minor data falsification.

Application:

1. Coverage estimation for random tests.
2. Investigation of system behaviour in critical situations.
3. Verification of the effectiveness of malfunction handling, safety mechanisms, etc. ...

Not suitable for targeted test selection or for test objects that still contain (many) faults, as it is not possible to generate similarly detectable mutations for many potential faults.

Fault injection is therefore primarily used to verify reliability, safety, etc. after fault rectification, not the test search beforehand.

High estimation accuracy requires large sample sizes.

2.29 Coverage criteria rather than model faults

Verification conditions for programming faults can be broken down into

- reachability,
- infection (creation of a local corruption) and
- spread of local observable misbehaviour to checked results.

Actual faults are not only unknown to the test search, but are also present in the programme. For faults of the type:

- Omitted requirement, case distinction, ...

the **only common detection condition** that can be modelled **is the reachability** of the programme location with the missing branch.

The probability that a test which satisfies the reachability criterion will also satisfy the infection and propagation criteria is generally low. Detecting missing aspects therefore requires an **independent coverage of each criterion by numerous tests** (slide 3.69).

Many tests per reachability criterion **will also uncover other faults** for which more close verification conditions can be modeled.

2.30 Cost of checkung coverage

In fault injection, the test object must be executed **with each assumed fault** or each injected data corruption until sufficient coverage is achieved. **On average**, this requires **many executions per test**.

Verifying reachability (and infection coverage) simply requires the instrumentation of counters at all reachable programme locations. **Only one programme execution per test** for all criteria combined.

The other reason, why coverage criteria rather than model faults are used for software.

In addition to checking coverage, **checks for incorrect behaviour** are required. Unlike in hardware manufacturing tests, target values cannot be determined by simulating the test object description. Checks for missbehaviour include

- **monitoring for crashes** and
- all types of automatic **checks for malfunctions** (format checks, values checks, ...).

2.31 Coverage modelling and instrumentation

A coverage model, similar to a fault model, is an algorithm for generating a set of coverage criteria from the description of the test object.

As with fault modelling after initial criteria compilation

- Combining criteria that are covered identically,
- Removal of redundant (uncoverable) criteria and
- optional removal of criteria that are implicitly covered.

Coverage instrumentation by inserting a coverage statement at the target position, optionally with an additional infection condition:

```
cov(Ci:tCov=auto);           // reachability only
cov(ci:bool,Ci:tCov=auto); // with infection condition
```

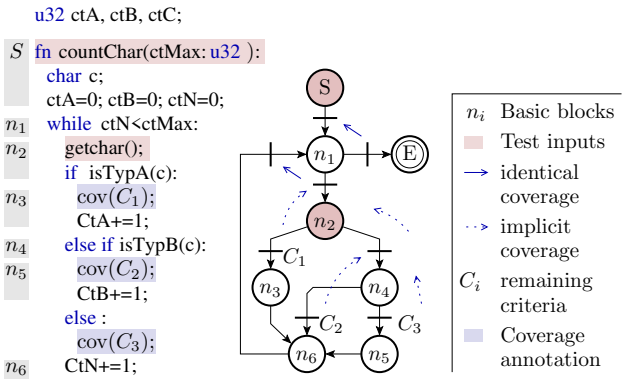
The coverage criteria C_i are symbols or numbers. If not specified, they are assigned automatically. Infection conditions c_i are logical expressions.

`tCov`, C_i yEnumeration type and symbolic value for coverage criterions.

c_i yConditions for infection.

Annotation: Integration of structured data that is not part of the programme into the source code.

2.32 Branch coverage



- Compiling the flow graph from basic blocks.
- The initial set of covering criteria comprises all edges.
- Remove edges that are identical and implicitly covered,
- Annotation of cov() statements for the remaining edges.

Basic block: a block of statements that are always executed together.

2.33 Programming language for examples

```
fn fname(x:i16,...)->(u16,char):
  if x==5:
    ...;
    ret (4, 'a');
```

Fictional programming language with descriptive elements drawn from various languages, each of which effectively highlights key aspects:

- Block structure through indentation (Python),
- Keywords, basic types (u8, u16, ...), function definitions returning structures, ... (Rust), but
- logical operators 'and', 'or' and 'not' instead of '&&', '||' and '!',
- omitting 'let' before assignments, but using 'ret' before return values,
- 'mut' (mutability) as the default and 'const' (immutability) to be specified explicitly. ...

Reasons for choosing a fictional language and a short introduction to Rust – see later section 7.1.

Continue coverage criteria: ... For the test search, the programme, extended with the cov() statements

- will be executed with random tests and
- tests that improve coverage, are collected

until the number of coverings for all criteria is sufficient.

An alternative to branch coverage is **instruction coverage**. Instead of edge reachability, it is sufficient to cover the reachability of all instructions. Slightly fewer criteria.

According to the current state of the art, only single coverage – i.e. only one test per criterion – is required, because, with software, the (actual) coverage of faults is still not in focus (slide 2.37).

Tools for measuring instruction and branch coverage: e.g. gcov for C/C++ programmes. Multiple coverage is not yet supported here either, but extending the tool to this will not pose problems.

Annotation: Integration of structured data that is not part of the programme into the source code.

2.35 Coverage of conditions

```
if ((a>10 and a<20) or (b>1 and b<85) and c==1:
  <Statement block A>
else:
  <Statement block B>
```

Apart from branch and instruction coverage, the **third established coverage model**; for us the first to **include** additional **infection criteria**.

Conditional coverage visually places the coverage criteria **at the edges of the decisions**, such that **small changes** in the values **of** any **decision variable reverse** the **decision** [4, 8]. Can also be regarded as an »off-by-one« fault that propagates to decision falsification (slide 2.27).

Table of infection condition for the coverage criteria for the example:

	A_{10}	A_{11}	A_{19}	A_{20}	B_1	B_2	B_{84}	B_{85}	C_0	C_1	C_2	
a	10	11	19	20	a_*	a_*	a_*	a_*	a_*	a_*	a_*	$a* = \overline{(a>10) \wedge (a<20)}$
b	b_*	b_*	b_*	b_*	1	2	84	85	b_*	b_*	b_*	$b* = \overline{(a>1) \wedge (a<85)}$
c	c_*	c_*	c_*	c_*	c_*	c_*	c_*	c_*	0	1	2	$c_* = (c=1)$

2.36 Instrumentation of coverage counters

	A_{10}	A_{11}	A_{19}	A_{20}	B_1	B_2	B_{84}	B_{85}	C_0	C_1	C_2	
a	10	11	19	20	a_*	a_*	a_*	a_*	a_*	a_*	a_*	$a* = \overline{(a>10) \wedge (a<20)}$
b	b_*	b_*	b_*	b_*	1	2	84	85	b_*	b_*	b_*	$b* = \overline{(a>1) \wedge (a<85)}$
c	c_*	c_*	c_*	c_*	c_*	c_*	c_*	c_*	0	1	2	$c_* = (c=1)$

```
if not(b>1 and b<85) and c==1:
  cov(a==10, CxA10); cov(a==11, CxA11);
  cov(a==19, CxA19); cov(a==20, CxA20);
if not(a>10 and a<20) and c==1:
  cov(b== 1, CxB1);  cov(b==2 , CxB2);
  cov(b==84, Cx84);  cov(b==85, CxB85);
if not(a>10 and a<20) and not(b>1 and b<85):
  cov(c==0,CxC0); cov(c==1,CxC1);cov(c==2,CxC2);
if ((a>10 and a<20) or (b>1 and b<85)) and c==1:
  <<Statement block A>
else:
  <<Statement block B>
```

Key point for now: there are far more criteria, and they are much more difficult to satisfy, compared to branch and statement coverage.

2.37 Stage of development, research objectives

Even in safety-critical applications, W1 coverage is still considered sufficient today, i.e. coverage of all criteria with at least one test. For example, the DO-178 B avionics standard requires only:

- W1 instruction coverage for non-safety-critical software,
- W1 branch coverage for software that can cause significant failures, and
- W1 condition coverage for flight-critical software.

The statistical relationship between criterion coverage and actual fault coverage – which recommends coverage with $w \gg 1$ tests for reachability [and infection criteria] – is apparently not yet a focus of attention (slide 2.29).

This is certainly not due to a lack of awareness that $w \gg 1$ is desirable, but rather to the fact that the number of tests required for this is **prohibitively expensive with current manual test programming**.

The research objective is therefore the automation of test generation.

W1 yEach criterion must be covered by at least one test.

1.5 Yield, Defect Level

2.38 Yield and defect level

In the case of systems and components that cannot be repaired, it is not the number of faults but the percentage of usable or defective products that matters. Key figures:

- Yield, defined as the proportion of products found to be good:

$$Y = 1 - \left.\frac{\#DD}{\#P}\right|_{\text{SLC}} \tag{2.4}$$

- Defect level: proportion of defective products:

$$DL = \left.\frac{\#D}{\#P}\right|_{\text{SLC}} \tag{2.5}$$

The defect level is expressed in dpu (defects per unit) or dpm (defects per million):

$$1\,\text{dpu} = 10^6\,\text{dpm}$$

Y, DL yYield. Defect level.
$\#D$, $\#DD$ yNumber of all defective products. Number of defective products detected.
$\#P$ yNumber of all products tested.
SLC yUseful estimates only with Sufficiently large count values.

2.39 Defect coverage and defect permeability

$$DC = \left.\frac{\#DD}{\#D}\right|_{\text{SLC}} \tag{2.6}$$

$$DP = 1 - DC \tag{2.7}$$

Yield and defect level of untested products:

$$Y = 1 - DL_{\text{M}} \cdot DC \tag{2.8}$$

Without a test ($DC = 0$), the yield is always $Y = 1$.

Rejecting detected defective parts reduces the numerator and denominator in (eq. 2.5) by the number of detected defective parts:

$$DL = \frac{\#P \cdot DL_{\text{M}} - \#P \cdot DL_{\text{M}} \cdot DC}{\#P - \#P \cdot DL_{\text{M}} \cdot DC}$$

DC, DP yDefect coverage. Defect permeability. (Share of (non-)detectable defective parts).
$\#D$, $\#DD$ yNumber of all defective products. Number of defective products detected.
Y, DL yYield. Defect level.
DL_{M} yDefect level after manufacturing before discarding detected defective parts.

$$DL = \frac{DL_M \cdot (1-DC)}{1-DL_M \cdot DC} \tag{2.9}$$

Substituting of the defect level DL_M of the manufacured untested parts with the yield Y after production:

(2.8) $$Y = 1 - DL_M \cdot DC$$

$$DL = \frac{(1-Y)\cdot(1-DC)}{Y \cdot DC} \tag{2.10}$$

Required defect level DL or permissible defect permeability DP for sorting out the detected defective parts, depending on the yield Y:

$$DC = \frac{1-Y}{1+(DL-1)\cdot Y} \tag{2.11}$$

$$DP = 1 - DC = \frac{1+(DL-1)\cdot Y-(1-Y)}{1+(DL-1)\cdot Y}$$

$$DP = \frac{DL \cdot Y}{1+(DL-1)\cdot Y} \tag{2.12}$$

Y, DL yYield. Defect level.

DC, DP yDefect coverage. Defect permeability. (Share of (non-)detectable defective parts).

2.41 Defect level of digital circuits

Typical order of magnitude of the values for circuits are, found in the literature:

- Yield: $Y = 10\% \ldots 90\%$
- Defect level: $DL = 200\,\text{dpm} \ldots 1000\,\text{dpm}$
- Fault permeability for stuck-at faults: $FP_{sa} = 1\% \ldots 10\%$..

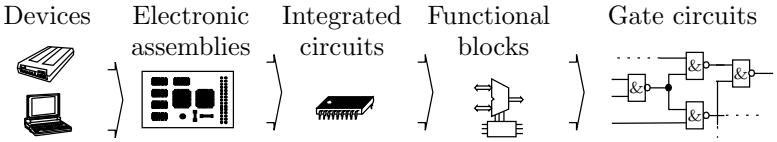
DL	$Y = 10\%$	$Y = 50\%$	$Y = 90\%$
200 dpm	$DP = 2.2 \cdot 10^{-5}$	$DP = 2 \cdot 10^{-4}$	$DP = 1.7 \cdot 10^{-3}$
1000 dpm	$DP = 1.1 \cdot 10^{-4}$	$DP = 2 \cdot 10^{-3}$	$DP = 8.9 \cdot 10^{-3}$

According to the estimated values, the defect permeability must be one to two orders of magnitude lower than the fault permeability for stuck-at-faults. This raises the following questions:

- Does $DP \ll FP_{sa}$ actually apply to circuits, or
- is the defect level actually that much higher?

1.6 Isolated module test

2.42 Testing, Debugging and Hierarchy



- Computer systems consist of computers, input/output devices, printers and network components, which in turn consist of ...
- The hardware provides basic functions for the software (machine instructions, input and output, ...).
- Software is organised into subsystems, modules, libraries, ...

Testing and troubleshooting follow this hierarchy:

- thorough, isolated testing of the subsystems prior to integration,
- in the higher-level system, test focus on interaction.

Subsystem faults not detected by the subsystem tests are, for the most part, not detected by the tests of the overall system either.

2.43 Number of faults and fault level

The total number of faults comprises

- component faults not detected by component tests and
- connection faults not detected by the overall test:

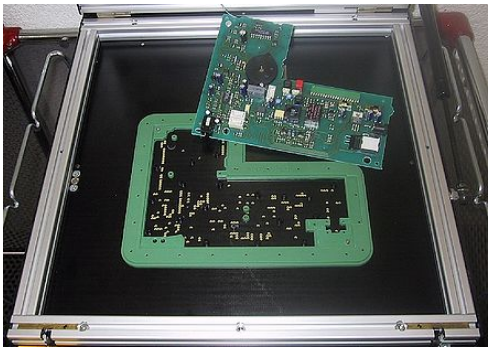
$$\mu_{\mathrm{F}} = \mu_{\mathrm{F.Con}} \cdot FP_{\mathrm{Con}} + \sum_{i=1}^{\#Prt} \mu_{\mathrm{F}.i} \tag{2.13}$$

For a very small number of total expected faults $\mu_{\mathrm{F}} \ll 1$, each component has at most one fault. The expected fault level $\mu_{\mathrm{DL}.i}$ per component is approximately equal to its expected number of faults $\mu_{\mathrm{F}.i}$. Estimated total number of faults:

$$\mu_{\mathrm{F}} = \mu_{\mathrm{F.Con}} \cdot FP_{\mathrm{Con}} + \sum_{i=1}^{\#Prt} \mu_{DL_i}$$

μ_{F}, $\mu_{\mathrm{F.Con}}$ yExpected number of overall faults in the system. Expected number of connection faults.
$\#Prt$, $\mu_{\mathrm{F}.i}$ yNumber of subsystems. Expected number of faults of subsystem i.
FP_{Con} Fault permeability for connection faults.
$\mu_{\mathrm{DL}.i}$ yExpected defect level of subsystem or component i.

2.44 Printed circuit board testing



Assembled printed circuit boards consist of tested components with very small defect level:

$$\mu_{\mathrm{F}.i} = \mu_{\mathrm{DL}.i}$$

For testing they are clamped onto a pin bed. Target faults: open circuits, short circuits and assembly errors.

Fault coverange for connection faults is practically $FC_{\mathrm{Con}} = 1$ and defective components almost never detected in this way:

$$\mu_{\mathrm{F}} = \sum_{i=1}^{\#Prt} \mu_{\mathrm{DL}.i} \tag{2.14}$$

For $\mu_{\mathrm{F}} \ll 1$, the expected defect level of the assembly is equal to the expected overall number of faults.

μ_{F} yExpected number of faults in the overall system.
$\#Prt$ yNumber of parts.
$\mu_{\mathrm{DL}.i}$ yExpected defect level of subsystem or component i.

2.45 Example estimation

Printed circuit board with the following components:

Type	Quantity	$\mu_{\mathrm{DL}.i}$
Printed circuit board	1	20 dpm
Integrated Circuits	20	200 dpm
Discrete components	35	10 dpm
Solder joints	560	1 dpm

$$\begin{aligned}\mu_{\mathrm{DL.Sys}} = \mu_{\mathrm{F}} &= 10\,\mathrm{dpm} + 20 \cdot 200\,\mathrm{dpm} + 35 \cdot 10\,\mathrm{dpm} + 560 \cdot 1\,\mathrm{dpm} \\ &= 5000\,\mathrm{dpm} = 0{,}005\,\mathrm{dpu}\end{aligned}$$

Approximately one in every 200 devices contains an undetected defective component. If the defect level of the circuits is indeed one or two orders of magnitude higher, as considered possible on slide 2.41, then the proportion of devices shipped with a hardware fault that reduces reliability is on of 20 or of two.

$\mu_{\mathrm{DL.Sys}}$ yDefect level of the system.

Summary

2.46 Test variety, static and dynamic testing

IT systems are subjected to a wide range of tests (𝕃):

- following the design flow, after each design phase,
- following the production flow, module by module, and then examining how the modules interact within the higher-level system,
- to rectify faults prior to deployment and as maintenance tests,
- both either statically and dynamically.

Static tests (direct feature control) (𝕃):

- Checks on features relating to the description or implementation.
- see also the degree programme in Measurement Technology and the section related to monitoring of IT results.

Dynamic tests (testing with sample inputs) (𝕃): Important whether

- model faults / coverage criteria used for evaluation only, or
- also a for targeted search, or hybrid approaches.

2.47 Test metrics

Fault-related parameters are fault coverage, fault permeability and phantom failure rate (𝕃):

(2.1) $$FC = \left.\frac{\#DF}{\#F}\right|_{\mathrm{SLC}} \tag{2.2}$$

$$FP = 1 - FC \tag{2.3}$$

When rectifying faults by sorting out defective components, the proportion of defective components matters, not the number of faults per component. Key parameters: yield, defect level, defect coverage, defect permeability (𝔹):

(2.7) $$DP = 1 - DC$$

(2.8) $$Y = 1 - DL_{\mathrm{M}} \cdot DC$$

(2.9) $$DL = \frac{DL_{\mathrm{M}} \cdot (1-DC)}{1 - DL_{\mathrm{M}} \cdot DC}$$

(2.10) $$DL = \frac{(1-Y) \cdot (1-DC)}{Y \cdot DC}$$

(2.11) $$DC = \frac{1-Y}{1+(DL-1) \cdot Y}$$

(2.12) $$DP = \frac{DL \cdot Y}{1+(DL-1) \cdot Y}$$

2.48 Isolated test

(𝕃) Thorough isolated testing of components. The higher-level system is made up of thoroughly tested, high-quality parts. Target faults in the composed systems are the interactions faults.

(𝔹) Expected number of faults in composite systems:

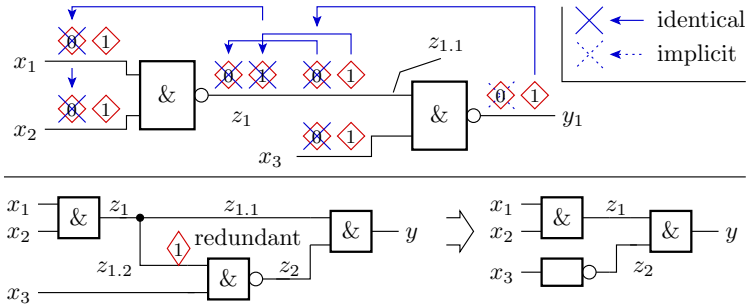
(2.13) $$\mu_{\mathrm{F}} = \mu_{\mathrm{F.Con}} \cdot FP_{\mathrm{Con}} + \sum_{i=1}^{\#Prt} \mu_{\mathrm{F}.i}$$

Where the expected number of faults per component is low and all interconnection faults are to be detected (typical for the testing of integrated circuit assemblies):

(2.14) $$\mu_{\mathrm{F}} = \sum_{i=1}^{\#Prt} \mu_{\mathrm{DL}.i}$$

Despite the low defect level of the components, it is not unlikely that a new computer contain a defective circuit which reduces its reliability.

2.49 Stuck-at faults(𝔹)



The most important fault model for digital circuits for decades:

- (𝕃) Initial fault set: sa0 and sa1 for each gate connection.
- (𝕃) Group together identically verifiable faults; remove redundant and implicitly verifiable faults.

Further this will be the example fault model, where illustrations require specific assumptions and sets of model faults.

2.50 Criterion coverage(𝕃)

Software contains the faults to be found. For missing aspects (branches, exception handling, etc.), **only** similarly verifiable **reachability [plus infection] criteria** can be derived. High actual fault coverage requires many tests per criterion, which cover it independently of each another. Many tests per reachability criterion also detect the other faults for which similarly verifiable model faults could be constructed.

The need for multiple coverage has not yet been widely accepted by the professional community. The much longer test suites required for this are prohibitively expensive when tests are designed manually. The objective is therefore automation.

Common coverage models: branch, statement and condition coverage. Similar to model faults, an initial coverage set is generated. Identical criteria are merged, whilst implicitly covered and redundant criteria are skiped. Instrumentation with coverage verification statements. Simultaneous verification of all criteria in a single programme run per test.

2.51 Fault Injection(𝕃)

Injection of a sample of typical coding mutations or data corruptions and verification as to whether they are detectable. For a significant proportion of the expected faults, the generation of similarly detectable mutations is only possible after the actual faults have been rectified.

A method for retrospectively checking how effective the test was. Also used to estimate reliability, security, etc.

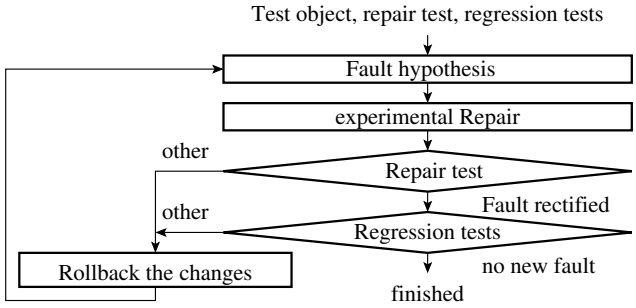
Coverage verification by fault injection requires the execution of each test with all mutations. Consequently, this involves a much greater computational effort compared with the verification of reachability [and infection] criteria.

Required fault sample size for determining the fault coverage and the reliability with a given estimation accuracy and error probability, see later section 4.2 ff..

2 Fault elimination

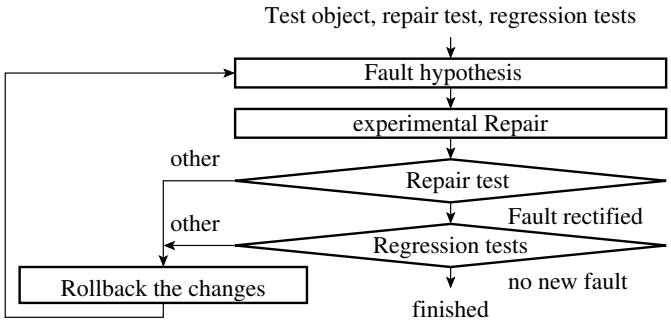
2.1 Experimental Repair

2.52 Experimental Repair



Iteration of attempts to rectify hypothetical faults and verification of success by repeating the test. Rollback after unsuccessful repair attempts, in order to rectify any faults that may have arisen during the repair.

The unit under tests as well as the tests used to verify the success must behave deterministically, i.e. produce the same result when repeated.

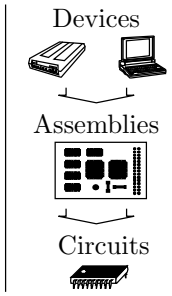


The tests for verifying success are divided into:

- **Repair tests**: Single tests that have failed to verify that the observed problem is resolved.
- **Regression tests**: All successful tests identified to date that do not reveal any faults. Required to check for any new faults that may have been created by repair or ongoing development.

If a regression test fails, roll back and continue searching.

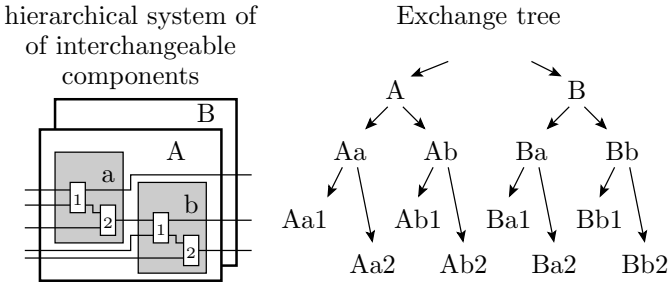
2.54 Systematic replacement



A system designed for repair has a hierarchical structure comprising interchangeable components, e.g.

1. Level: Replaceable devices.
2. Level: Replaceable circuit boards.
3. Level: Replaceable circuits.

Fault localisation through systematic replacement:



2.55 Standard procedure for a repair technician

- A rough assessment of which part of the computer might be faulty, based on the observed symptoms.
- Checking connectors for contact issues by unplugging, cleaning, reconnecting and repeating the test.
- Replacing potentially faulty parts with spare parts, repeating the test, ...

Prerequisites:

- Repeatable tests that detect the fault.
- Sufficient spare parts. General repair skills(1).

Questions:

- Why it is advisable to replace half of the components, and then half of the faulty half, and so on?(2)
- Fault diagnosis through systematic replacement for software?(3)

(1) An understanding of the system to be repaired is not essential.

(2) Binary swap tree; for n alternatives, approximately $\log(n)$ trials (see previous slide).

(3) In the case of a new fault and an old, working version.

2.2 Fault diagnosis & isolation

2.56 Fault diagnosis

Estimation of the location, cause and possible remedies for faults based on test results to minimise:

- the number of repair trials,
- the demand for/consumption of spare parts(1),
- the number of faults arising during repair attempts,
- including those not rectified by rollback.

General diagnostic techniques:

- success- oriented replacement and

- tracing data corruption against the flow of calculations.

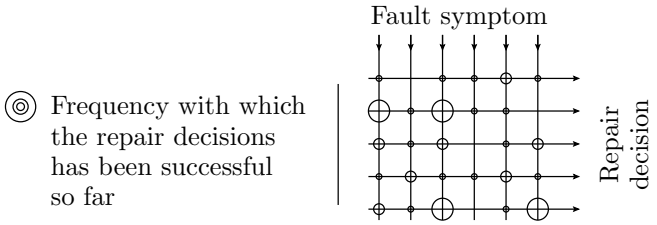
Prerequisite: repair-friendly design.

(1) Spare parts which have been fitted but have not rectified the fault will remain in the system and will be charged to the customer.

2.57 Success-oriented replacement

Products have weak points. Most problems can be traced back to a small proportion of possible causes (Pareto principle)[PP]:

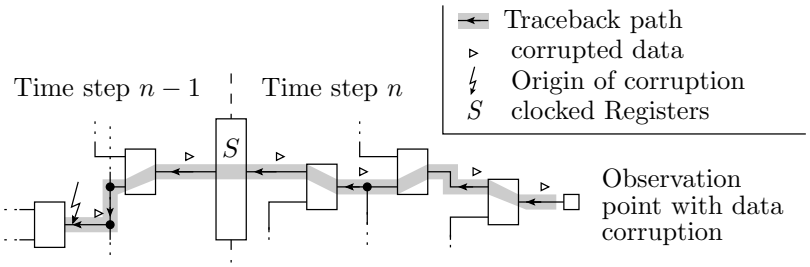
- Counting the success rates of different repair alternatives.
- Start repairs with the most promising option.



After unsuccessful attempts rollback of all changes to reduce the rate of faults arising during the repair.

(PP) yPareto principle: A small proportion of the causes account for the majority of the effects. A few tests reveal the vast majority of faults. A small proportion of faults cause the vast majority of malfunctions. ....

2.58 Tracing back data corruptions



From a detected falsification (output or other checked point), trace back against the signal flow to the component with falsified output and correct inputs, where necessary via registers to the previous clock cycle.

In software, the calculation is traced back by repeating with a display of previous results or by trace recording.

Possible faults, apart from a »defective source«, also include short circuits or, in the case of software, misdirected write accesses.

Tracing back from an instruction that produces incorrect results by

- inserting additional outputs along the calculation flow back, or
- manually checking values step by step or using breakpoints.

Both approaches may require multiple test repetitions. A trace log of the programme's execution, including interim results prior to the termination caused by the malfunction, to trace the malfunction back through the recorded data, provides a great deal of relief.

2.60 Fault isolation during operation

Fault isolation: Preventing malfunctions in subsystems from affecting the rest of the system.

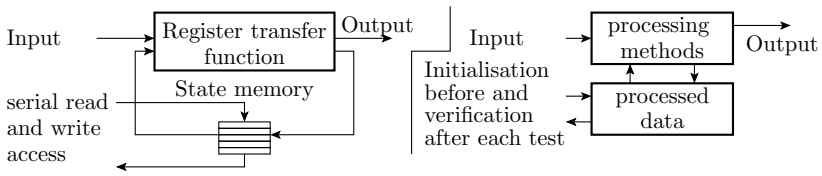
Fault isolation during operation:

- Checking intermediate results at subsystem boundaries. Denial of service or correction prior to data transfer.
- Prevention of unauthorised resource blocking (computing time, memory, I/O devices, etc.) by faulty subsystems.
- No access to data and resources of other functional units except via defined interfaces.

is

- **a prerequisite for a reliable response to detected problem situations**, including the generation of accurate problem descriptions for troubleshooting and rectification, and
- a fundamental architectural pattern, particularly for operating systems, embedded systems, distributed systems, safety-critical systems, ...

2.61 Fault isolation under test conditions



Isolated test: Testing of all subsystems, including the identification and rectification of detected faults prior to integration into the higher-level system(slide 2.7).

State monitoring (and specification) between test steps to reduce traceback and propagation paths to a single time step / a single service request. For hardware, e.g. by serially reading and overwriting the state bits (scan method) ; for software, by monitoring the state after each test step.

Shorter propagation or backtracking paths mean fewer wrong fault hypotheses and fewer unsuccessful repair attempts.

2.62 Blind troubleshooting

The alternative to systematic replacement, with or without fault diagnosis, is »blind troubleshooting«, i.e. intuitive trial and error.

Time-consuming, often unsuccessful and frustrating, yet:

- with no option to swap parts,
- with no means of tracing the fault,
- with no known workarounds (slide 1.103), and
- with no proper documentation of the system

the only way to find and rectify faults.

2.63 Repair-friendly design

Collections of

- good practic' rules, designed to facilitate or simplify testing, fault localisation and repair, and
- anti-patterns, typical patterns that cause problems.

Useful/necessary as described:

- Deterministic behaviour with a directed flow of computation.
- A modular system consisting of interchangeable / separately testable functional blocks.
- Fault isolation (slide 2.60).

Bad: Anything that forces a blind troubleshooting.

Repairability and testability are important design disciplines. If everything is done correctly, the impact on reliability is negligible.

Summary

2.64 Fault elimination (debugging) (𝕃)

Debugging: Iteration consisting of attempts to correct hypothetical faults and verification of success through test repetition:

- Elimination of all detectable faults.
- Rollback following unsuccessful repair attempts.
- For interchangeable subsystems, through systematic swapping.

Fault diagnosis: Assessment of the location, cause and possible remedies for faults based on test results:

- Start with most promising troubleshooting attempts (Pareto principle).
- Tracking falsifications contrary to the direction of the calculation or the signal flow.

Design for repair:

- Interchangeable modules, deterministic behaviour,
- directed computation flow, fault isolation, ...

If handled sensibly, the impact of bugs fixing on reliability is negligible compared with that of test quality.

3 Reliability & Test

3.1 Simple estimation

Example 2.2 Number of faults and reliability

Programme size 10,000 NLOC. 30–100 faults per 1,000 NLOC. Overall fault permeability for all tests together $FP = 30\%$.

a) How many faults remain after all identifiable faults have been rectified??

$$10{,}000\,\text{NLOC} \cdot \frac{30\ [\text{F}]\ \ldots\ 100\,[\text{F}]}{1000\,\text{NLOC}} \cdot 30\% = 90\,[\text{F}]\ \ldots\ 300\ [\text{F}]$$

b) How reliable is a system with approximately 90 to 300 faults?

It depends on the malfunction rate of the undetected faults, and this in turn depends on the number of tests used to identify the rectified faults.

As the number of tests increases, not only does the proportion of undetectable faults decrease, but also the average malfunction rate of the faults left.

We'd like to know more about this!

[F] yCount in faults.
FC yFault coverage, percentage of detectable faults.
NLOC yNetto lines of code, number of code lines without comments and blank lines).

2.66 Malfunction rate due to faults

Every unresolved fault i causes malfunctions at a rate ζ_i (in MF per DS). The sum of the malfunction rates of all faults

$$\zeta_{\sum} = \sum_{i=1}^{\#F} \zeta_i$$

is an upper bound $\zeta_{\mathrm{F}} \leq \zeta_{\sum}$ and, if almost all malfunctions are caused by a single fault, equal to the sum of the malfunction rates of all faults:

$$\zeta_{\mathrm{F}} \overset{(\leq 1)}{=} \sum_{i=1}^{\#F} \zeta_i \quad \text{for} \quad \zeta_{\mathrm{F}} \ll 1$$

For the reliability tests, the systems have been pre-tested, cleared of most and the most serious faults, and they deliver largely correct results. This implies $\zeta_{\mathrm{F}} \ll 1$.

MF, DS yMalfunction. Delivered service.
$\#F$ yNumber of existing faults.
ζ_i yMF rate caused by fault i.
ζ_{F} yMalfunction rate caused by faults.
(≥ 1) yThe calculated value is an upper limit. The actual value is max. one.

2.67 After fault rectification

Assuming that the following applies:

- elimination of all detectable faults,
- mean malfunction rate per unresolved fault $\bar{\zeta}(N) \approx 1/N$,
- only one fault causing each malfunction,
- Testing under similar conditions as during operation (sec. 2.3.8).

Overall malfunction rate for all unresolved faults:

$$\zeta_{\mathrm{F}}(N) = \mu_{\mathrm{F}}(N) \cdot \bar{\zeta}(N) \tag{2.15}$$

Fault-related partial reliability as the reciprocal of the average malfunction rate due to faults:

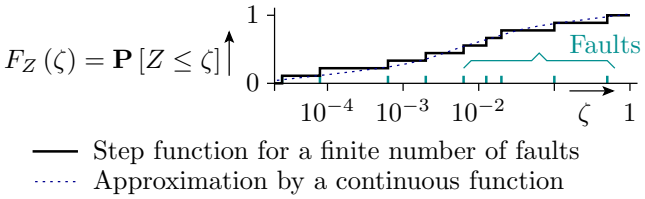
$$\bar{\zeta}(N) \approx N^{-1}$$

$$R_{\mathrm{F}}(N) \approx \frac{N}{\mu_{\mathrm{F}}(N)}$$

$\zeta_{\mathrm{F}}(N)$ yMalfunction rate caused by faults depending on the number of tests.
$\mu_{\mathrm{F}}(N)$ yExpected number of faults not detected and eliminated after N tests.
$\bar{\zeta}(N)$ yMean malfunction rate per fault, after removal of all faults detected by N tests.
N yNumber of tests for which all detected faults are eliminated.
$R_{\mathrm{F}}(N)$ yFault-related partial reliability after elimination of all faults detectable with N tests.

3.2 Improved model

2.68 Distribution of the malfunction rate



The distribution $F_Z(\zeta)$ of the random variable Z, denoting the fault related malfunction rate, describes the probability that Z (of a randomly selected fault) is not greater than ζ:

$$F_Z(\zeta) = \mathbb{P}[Z \le \zeta]$$

Density of the malfunction rate (see Slide Set 4 below):

$$h(\zeta) = \frac{\mathrm{d}F_Z(\zeta)}{\mathrm{d}\zeta} \text{ with } \int_0^1 h(\zeta) \cdot \mathrm{d}\zeta = 1$$

Z	yMalfunction rate per fault as a random variable.
$F_Z(\zeta)$	yDistribution function of Z. ζ malfunction rate per fault.
$h(\zeta)$	yDensity function of the malfunction rate.

2.69 Fault permeability and malfunction rate

Expected fault permeability for N randomly selected tests:

$$\mu_{\mathrm{FP}}(N) = \int_0^1 p_{\mathrm{FP}}(\zeta, N) \cdot h(\zeta) \cdot \mathrm{d}\zeta \tag{2.16}$$

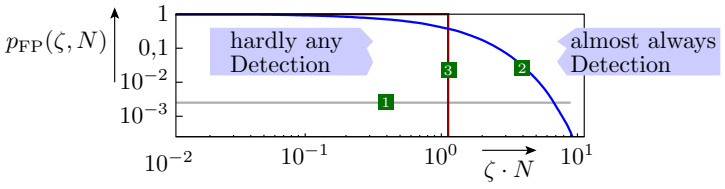
Average malfunction rate per fault after fixing all detectable faults:

$$\bar{\zeta}_{\mathrm{F}}(N) = \underbrace{\int_0^1 p_{\mathrm{FP}}(\zeta, N) \cdot h(\zeta) \cdot \zeta \cdot \mathrm{d}\zeta}_{\text{Average malfunction rate per fault}} \tag{2.17}$$

(Integration over the products of frequency of occurrence and malfunction rate).

$\mu_{\mathrm{FP}}(N)$	yExpected fault permeability for N randomly selected tests.
$p_{\mathrm{FP}}(\zeta, N)$	yProbability of passage for faults with an MF rate ζ and N tests.
$h(\zeta)$	yDensity of malfunction rate before fault elimination.
$\bar{\zeta}(N)$	yMean malfunction rate per fault, after removal of all faults detected by N tests.

2.70 Probability of permeability



The probability $p_{\mathrm{FP}}(\zeta, N)$ that a fault with a certain malfunction rate ζ will not be detected by testing also depends on the method of test selection:

1. static tests: No dependence of ζ of the fault.
2. Random test: Faults with $\zeta \ll N^{-1}$ are not detected and those with $\zeta \gg N^{-1}$ are allways detected. In between, see later

   (3.15) $$p_{\mathrm{FP}}(\zeta, N) = \mathrm{e}^{-\zeta \cdot N}$$

3. To simplify the integrals, the following approximation is first used:

$$p_{\mathrm{FP}}(\zeta, N) = \begin{cases} 1 & \zeta \le \frac{1}{N} \\ 0 & \text{else} \end{cases} \tag{2.18}$$

2.71 Typical fault permeability of random tests

In a random test, reducing the fault permeability by one decade requires an increase in the number of tests by more than one decade.

This is a property of a power function:

$$\mu_{\mathrm{FP}}\left(\tfrac{N}{N_0}\right) = \left(\tfrac{N}{N_0}\right)^{-K} \quad \text{with } 0 < K < 1 \tag{2.19}$$

K	1	0.5	0.33	0.25
$\frac{N}{N_0}$ für $\mu_{\mathrm{FP}}\,(N/N_0) = 0.1$	10	100	10^3	10^4

Fault permeability is due to (eq. 2.18) the fraction of faults whose failure rates are $\zeta \leq 1/N$. The exponent K subsequently becomes the shape parameter of the distribution of the malfunction rate.

$\mu_{\mathrm{FP}}\left(\frac{N}{N_0}\right)$ yExpected fault permeability of a test extension N/N_0.
K yShape parameter of the distribution of the malfunction rate $(0 < K < 1)$.
N, N_0 yTotal number of tests. Number of tests for which all detected faults are eliminated.

2.72 The density of the malfunction rate

Using the simplification for the probability of permeability and the empirically estimated fault permeability for all faults combined:

(2.18) $$p_{\mathrm{FP}}\,(\zeta, N) = \begin{cases} 1 & \zeta \leq \frac{1}{N} \\ 0 & \text{else} \end{cases}$$

(2.19) $$\mu_{\mathrm{FP}}\left(\tfrac{N}{N_0}\right) = \left(\tfrac{N}{N_0}\right)^{-K} \quad \text{with } 0 < K < 1$$

(2.16) $$\mu_{\mathrm{FP}}\,(N) = \int_0^1 p_{\mathrm{FP}}\,(\zeta, N) \cdot h\,(\zeta) \cdot \mathrm{d}\zeta$$

the density of the malfunction rate $h\,(\zeta)$ must satisfy the following:

$$\left(\tfrac{N}{N_0}\right)^{-K} = \int_0^{\frac{1}{N}} h\,(\zeta) \cdot \mathrm{d}\zeta \tag{2.20}$$

$\mu_{\mathrm{FP}}\left(\frac{N}{N_0}\right)$ yExpected fault permeability of a test extension N/N_0.
$h\,(\zeta)$ yDensity of malfunction rate before fault elimination.
K yShape parameter of the distribution of the malfunction rate $(0 < K < 1)$.
N, N_0 yTotal number of tests. Number of tests for which all detected faults are eliminated.
Pronounce: yζ: zeta, μ: my.

(2.20) $$\left(\tfrac{N}{N_0}\right)^{-K} = \int_0^{\frac{1}{N}} h\,(\zeta) \cdot \mathrm{d}\zeta$$

The solution is the density function:

$$h\,(\zeta) = N_0^K \cdot K \cdot \zeta^{K-1} \quad \text{for } 0 < \zeta \leq \tfrac{1}{N_0} \quad \text{else } 0 \tag{2.21}$$

Proof:

- Reference test count $N = N_0$:

$$N_0^K \cdot K \cdot \int_0^{\frac{1}{N_0}} \zeta^{K-1} = N_0^K \cdot \cancel{K} \cdot \left[\tfrac{1}{\cancel{K}} \cdot \zeta^K\right]_0^{\frac{1}{N_0}} = \left(\tfrac{\cancel{N_0}}{\cancel{N_0}}\right)^{-K} = 1 \surd$$

- Test extension $N > N_0$, integration limit $\frac{1}{N}$ instead of $\frac{1}{N_0}$:

$$\int_0^{\frac{1}{N}} h\,(\zeta) \cdot \mathrm{d}\zeta = \int_0^{\frac{1}{N}} N_0^K \cdot K \cdot \zeta^{K-1} \cdot \mathrm{d}\zeta$$
$$= N_0^K \cdot \cancel{K} \cdot \left[\tfrac{1}{\cancel{K}} \cdot \zeta^K\right]_0^{\frac{1}{N}} = \left(\tfrac{N}{N_0}\right)^{-K} \surd$$

$h(\zeta)$ yDensity of malfunction rate before fault elimination.
K yShape parameter of the distribution of the malfunction rate ($0 < K < 1$).
N, N_0 yTotal number of tests. Number of tests for which all detected faults are eliminated.

2.74 After fault rectification

The rectification of the additional detectable faults by an increased number of tests $N > N_0$ increases the scale factor of the fault density to N:

$$h(\zeta, N) = N^K \cdot K \cdot \zeta^{K-1} \quad \text{for } 0 < \zeta \leq \tfrac{1}{N} \quad \text{else } 0 \tag{2.22}$$

and reduces the average malfunction rate per fault to:

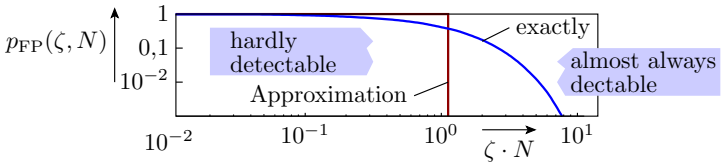
$$\begin{aligned} \bar{\zeta}(N) &= \int_0^1 \zeta \cdot h(\zeta, N) \cdot \mathrm{d}\zeta = N^K \cdot K \cdot \int_0^{\frac{1}{N}} \zeta^K \cdot \mathrm{d}\zeta \\ &= N^K \cdot K \cdot \left[\tfrac{1}{K+1} \cdot \zeta^{K+1} \right]_0^{\frac{1}{N}} \\ \bar{\zeta}(N) &= \tfrac{K}{(K+1)\cdot N} \quad \text{with } 0 < K < 1 \end{aligned} \tag{2.23}$$

Confirmation of hypothesis $\bar{\zeta}(N) \approx \frac{1}{N}$ on slide 2.67.

The term $(K+1)$ in the denominator can be omitted; see the next slide.

$h(\zeta, N)$ yDensity of the malfunction rate after elimination of the faults detectable with N tests.
K yShape parameter of the distribution of the malfunction rate ($0 < K < 1$).
N, N_0 yTotal number of tests. Number of tests for which all detected faults are eliminated.
$\bar{\zeta}(N)$ yMean malfunction rate per fault, after removal of all faults detected by N tests.

2.75 Subsequent model refinement



In section 3.2.1, it will be shown later that the probability of fault passage decreases according to an exponential function, rather than the step function assumed for simplicity:

(3.15) $$p_{FP}(\zeta, N) = \mathrm{e}^{-\zeta \cdot N}$$

This alters the density of the malfunction rate to:

(3.20) $$h(\zeta, N) = K \cdot N^K \cdot \mathrm{e}^{-\zeta \cdot N} \cdot \zeta^{K-1}$$

The fault per
$K+1$ is dropp

(3.21) $$\bar{\zeta} = \tfrac{K}{N}$$

We're doing **without the term** $K+1$ out from here on.

2.76 Number of faults and malfunction rate

(2.19) $$\mu_{FP}\left(\tfrac{N}{N_0}\right) = \left(\tfrac{N}{N_0}\right)^{-K} \quad \text{with } 0 < K < 1$$

(3.21) $$\bar{\zeta} = \tfrac{K}{N}$$

The expected number of faults decreases proportionally to the fault permeability, with an exponent $0 \leq K \leq 1$. Decrease in the malfunction rate due to faults with exponent $K+1$:

$$\mu_F(N) = \mu_F(N_0) \cdot \left(\tfrac{N}{N_0}\right)^{-K} \tag{2.24}$$

(2.15) $$\zeta_F(N) = \mu_F(N) \cdot \bar{\zeta}(N)$$

$$\zeta_F(N) = \frac{K \cdot \mu_F(N)}{N} \tag{2.25}$$

$$\zeta_F(N) = \zeta_F(N_0) \cdot \left(\frac{N}{N_0}\right)^{-(K+1)} \tag{2.26}$$

$\mu_{FP}\left(\frac{N}{N_0}\right)$	yExpected fault permeability of a test extension N/N_0.
K	yShape parameter of the distribution of the malfunction rate ($0 < K < 1$).
N, N_0	yTotal number of tests. Number of tests for which all detected faults are eliminated.
$\mu_F(N)$	yExpected number of faults not detected and eliminated after N tests.
$\zeta_F(N)$	yMalfunction rate caused by faults depending on the number of tests.

2.77 Estimating the form factor

(2.24) $$\mu_F(N) = \mu_F(N_0) \cdot \left(\frac{N}{N_0}\right)^{-K}$$

(2.25) $$\zeta_F(N) = \frac{K \cdot \mu_F(N)}{N}$$

(2.26) $$\zeta_F(N) = \zeta_F(N_0) \cdot \left(\frac{N}{N_0}\right)^{-(K+1)}$$

- From the decline in the number of faults:

$$K = -\ln\left(\frac{\mu_F(N)}{\mu_F(N_0)}\right) / \ln\left(\frac{N}{N_0}\right) \tag{2.27}$$

- From the decline in the malfunction rate:

$$K = \ln\left(\frac{\zeta_F(N_0)}{\zeta_F(N)}\right) / \ln\left(\frac{N}{N_0}\right) - 1 \tag{2.28}$$

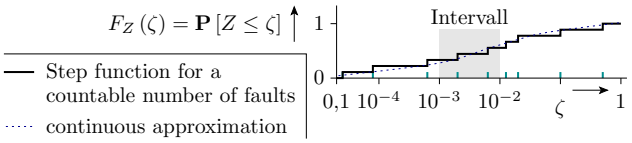
- From the ratio of the malfunction rate to the number of faults:

$$K = \frac{N \cdot \zeta_F(N)}{\mu_F(N)} \tag{2.29}$$

$\mu_F(N)$	yExpected number of faults not detected and eliminated after N tests.
$\zeta_F(N)$	yMalfunction rate caused by faults depending on the number of tests.
N_0, N_1	yTest number with known / searched malfunction rate or fault number.
K	yShape parameter of the distribution of the malfunction rate ($0 < K < 1$).

3.3 Interpretation, simulation

2.78 Distribution of fault by malfunction ranges



(2.21) $$h(\zeta) = N_0^K \cdot K \cdot \zeta^{K-1} \quad \text{for } 0 < \zeta \leq \tfrac{1}{N_0} \quad \text{else } 0$$

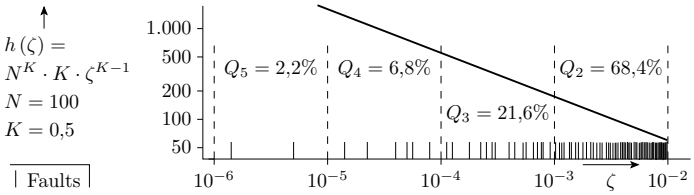
Distribution function:

$$F_Z(\zeta) = \int_0^{\zeta} h(x) \cdot \mathrm{d}x = (N_0 \cdot \zeta)^K \quad \text{for } 0 < \zeta \leq \tfrac{1}{N_0} \tag{2.30}$$

Partitioning into areas with boundaries in increments of ten::

$$\zeta_i \in \tfrac{1}{N}\left[10^{-(i+1)}, 10^{-i}\right]$$

To prove the presence of faults in each area requires power-of-ten longer test sequences. According to the Pareto principle, which states that a small proportion of tests reveals the majority of faults, the frequency of faults per area should decrease as the range number increases. ...



Area sizes and probabilities of belonging to an area:

$$q_i = 10^{-i} - 10^{-(i+1)} = 0{,}9 \cdot 10^{-i}; \quad \boxed{q_{i+1}{=}q_i{\cdot}10^{-1}}$$

$$Q_i = \int_{10^{-(i+1)}}^{10^{-i}} h(\zeta) \cdot \mathrm{d}\zeta = \left(N \cdot 10^{-i}\right)^K - \left(N \cdot 10^{-(i-1)}\right)^K$$

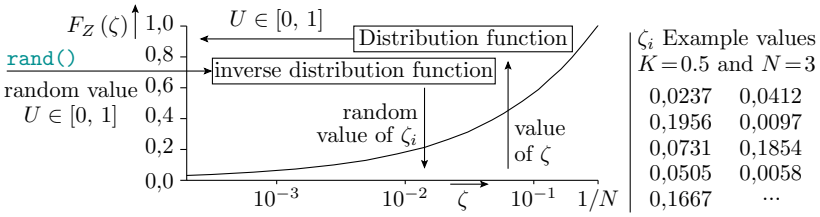
$$= \left(1 - 10^{-K}\right) \cdot N^K \cdot 10^{-i \cdot K}; \quad \boxed{Q_i{=}Q_{i-1}{\cdot}10^{-K}}$$

With regard to the sum of a geometric series, the same reduction factors also apply to the sums for all other areas up to ∞.

Move to the next range: Detecting the faults requires a **tests set length ten times as long**. Reduction of the size of the uncovered areas to one-tenth. **Reduction in the overall fraction** of faults in the **uncovered** areas **by a factor of** 10^{-K}.

Graphical interpretation of the initial assumptions on the slide 2.71.

2.80 Generation of sample malfunction rates

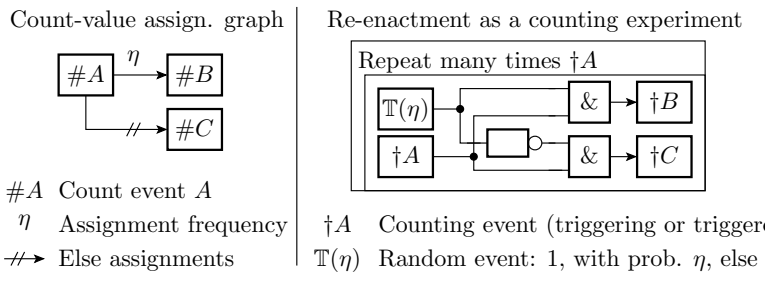


For a given probability distribution, we generate random sample values as follows:Für eine bekannte Verteilungsfunktion lassen wie folgt Stichprobenwerte auswürfeln (sec.4.1.6): The distribution function $F_Z(\zeta)$ maps the values $\zeta \in [0, 1/N_0]$ to values $U \in [0, 1]$. Conversely, uniformly distributed sample values $U \in [0, 1]$ are mapped to sample values $\zeta \in [0, 1/N_0]$ with the inverse function $\zeta = F_Z^{-1}(U)$. Random function for sample values:

$$U = F_Z(\zeta) = (N_0 \cdot \zeta)^K \quad \Rightarrow \; \zeta_i = \tfrac{1}{N_0} \cdot (\texttt{rand}())^{\frac{1}{K}} \tag{2.31}$$

rand() Function for generating uniformly distributed random values between 0 and 1.

2.81 Simulation of counting experiments



Mapping graph on the left: Count event for A is althought for B with frequency η , otherwise for C . Re-enactment as a counting experiment:

Repeat this for many counting trials (many events $\dagger A$):
Generate random event (1 with rate η, else 0)
• if 1: Counting event $\dagger B$: $(\#B = \#B + 1)$
• else: Counting event $\dagger C$: $(\#C = \#C + 1)$

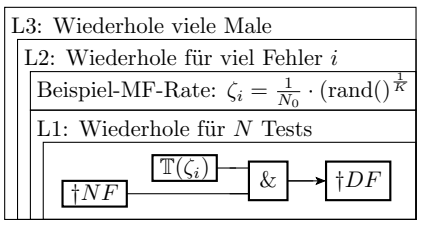
Repeat this for many counting trials (many events $\dagger A$):
Generate random event (1 with rate η, else 0)
• if 1: Counting event $\dagger B$: $(\#B = \#B + 1)$
• else: Counting event $\dagger C$: $(\#C = \#C + 1)$

As the number of counting trials increases, the count ratio $\#B/\#A$ approaches the assignment frequency η.
The counting experiment not only allows us to estimate assignment frequencies, but also, for example,

- to investigate the accuracy of the estimates for a finite number of trials,
- including situations that cannot be described using count-value assignment graphs, or
- in general, for situations for which mathematics does not offer simple solutions.

Also, a **means of verifying the theory** in this section, which has not yet been recognised in professional circles.

2.83 Counting experiment for fault detection



L1 Bestimmung der Nachweisanzahl von Fehler i mit N Tests
L2 Bestimmung der Anzahl der nachweisbaren Fehler für N Tests
L3 Wiederholung mit einer grossen Versuchanzahl für gute Schätzgenauigkeit, ...

In this particular experiment, a random value for the malfunction rate is generated in loop L2, which is then used in loop L1 to carry out the elementary random experiment:

- Increment the counter for the total number of faults $\#NF$ and
- with frequency ζ_i , also the count $\#DF$ for detectable faults.

In this way, you can estimate: the expected value, the frequency distribution, ... as a function of the number of tests N. Programming with Malab – see slide 3.57 (after showing that the malfunction rate is gamma-distributed).

3.4 Pre- and reliability tests

2.83 Preliminary tests

Before a reliability test preliminary tests are carried out:

- static tests: reviews, syntax, ...
- Rough tests to see if anything works at all,
- tests specifically designed for borderline and special cases, ...

In accordance with the Pareto principle, these identify the most common and the most problematic faults, which – in line with our idealised fault culture – are thus eliminated at the start of the reliability test.

Assumption: All pre-tests together have a fault permeability of $FP_{\mathrm{PT}} \ll 1$ and effectively comprise $N_0 \gg 1$ dynamic tests. Number of faults and malfunction rate at the start of the reliability test:

$$\mu_{\mathrm{F}}(N_0) = \mu_{\mathrm{FCR}} \cdot FP_{\mathrm{PT}} \tag{2.32}$$

$$\zeta_{\mathrm{F}}(N_0) = \frac{K \cdot \mu_{\mathrm{F}}(N_0)}{N_0} \quad \text{for } \zeta_{\mathrm{F}}(N_0) \ll 1 \tag{2.33}$$

μ_{FCR} yTotal expected number of faults from creation and repair processes.
$\mu_{\mathrm{F}}(N_0)$ yExpected fault number after elimination of all faults detected in preliminary tests.
FP_{PT} yOverall fault permeability of all preliminary tests combined.
N_0 yNumber of dynamic tests included in the preliminary tests.
$\zeta_{\mathrm{F}}(N_0)$ yMalfunction rate after elimination of faults detected by pre-tests.

2.84 Reliability test

A reliability test increases the number of tests from N_0 a total of N tests. Decrease in the expected fault permeability and number of faults by exponent K, and in the malfunction rate due to faults by exponent $K+1$:

(2.19) $$\mu_{\mathrm{FP}}\left(\frac{N}{N_0}\right) = \left(\frac{N}{N_0}\right)^{-K} \quad \text{with } 0 < K < 1$$

(2.24) $$\mu_{\mathrm{F}}(N) = \mu_{\mathrm{F}}(N_0) \cdot \left(\frac{N}{N_0}\right)^{-K}$$

(2.25) $$\zeta_{\mathrm{F}}(N) = \frac{K \cdot \mu_{\mathrm{F}}(N)}{N}$$

(2.26) $$\zeta_{\mathrm{F}}(N) = \zeta_{\mathrm{F}}(N_0) \cdot \left(\frac{N}{N_0}\right)^{-(K+1)}$$

Fault coverage:

$$\mu_{\mathrm{FC}}\left(\frac{N}{N_0}\right) = 1 - \left(\frac{N}{N_0}\right)^{-K} \tag{2.34}$$

$\mu_{\mathrm{FP}}\left(\frac{N}{N_0}\right)$ yExpected fault permeability of a test extension N/N_0.
$\mu_{\mathrm{FC}}(...)$ yCorresponding expected fault coverage of the reliability test.
N, N_0 yTotal nuber of tests. Number of dynamic pre-tests.
$\mu_{\mathrm{F}}(N)$ yExpected number of faults not detected and eliminated after N tests.
$\zeta_{\mathrm{F}}(N)$ yMalfunction rate caused by faults depending on the number of tests.

3.5 Reliability and Safety

2.85 Faults and disruptions

Effective measures to prevent disruptions:

- digital processing,
- electromagnetic shielding, ...

Highly effective malfunctions treatments for disruptions:

- Data transmission and storage with a verification code,
- Monitoring by doubling and comparison, ...

Where required, the malfunction rate due to disruptions can be reduced to a negligible level.

Classification of malfunctions according to their effect when the test is repeated:

- the same wrongdoing, usually caused by faults, and
- random or no wrongdoings, usually caused by disruptions.

Elimination of disruption-related problems, see the box above. The following assessments of reliability and safety ignore disruptions: $\zeta = \zeta_{\mathrm{F}}$.

2.86 Reliability

(2.25) $$\zeta_F(N) = \frac{K \cdot \mu_F(N)}{N} \quad (2.26)$$

$$\zeta_F(N) = \zeta_F(N_0) \cdot \left(\frac{N}{N_0}\right)^{-(K+1)} \quad (1.51)$$

Reliability, in the case of low failure rates with or without retest requirements, is the reciprocal of the proportion of malfunctions not detected by monitoring, limited here to those caused by faults:

$$R(N) = \frac{N}{K \cdot \mu_F(N) \cdot MP} \quad (2.35)$$

$$R(N) = R(N_0) \cdot \left(\frac{N}{N_0}\right)^{K+1} \quad (2.36)$$

Reliability increase with the number of effective tests, ideally with exponent $K+1$.

$\zeta_F(N)$	yMalfunction rate caused by faults depending on the number of tests.
R, MP	Reliability with monitoring. Malfunction permeability of the monitoring system.
N, N_0	yTotal nuber of tests. Number of dynamic pre-tests.
K	yShape parameter of the distribution of the malfunction rate ($0 < K < 1$).

2.87 Safety

(1.25) $$S = \frac{R}{\rho}$$

Under the addi
safety is the ra

$$S(N) = \frac{N}{K \cdot \mu_F(N) \cdot MP \cdot \sigma} \quad (2.37)$$

$$S(N) = S(N_0) \cdot \left(\frac{N}{N_0}\right)^{K+1} \quad (2.38)$$

Therefore, the increase in safety is greater than proportional to the number of tests N; also, it can be improved by reducing the number of unidentified faults μ_F, a lower rate of malfunction permeability in monitoring MP and a lower percentage of safety-threatening malfunctions ρ.

R, S	yReliability, safety or security, each under ideal conditions.
$\mu_F(N)$	yExpected number of faults not detected and eliminated after N tests.
MP, ρ	yMalfunction permeability. Proportion of malfunctions posing a safety risk.
N, N_0	yTotal number of tests. Number of tests for which all detected faults are eliminated.
Pronounce:	yζ: zeta, ρ: rho, μ: my.

Example 2.3 Reliability with triple the testing effort

a) *By what factors do the failure rate $\zeta_F(N)$ and the number of failures $\mu_F(N)$ decrease when the number of dynamic tests is tripled? Shape parameters of the malfunction rate distribution $K \in \{0{,}3;\ 0{,}5\}$.*

Estimated reduction in the MF rate and the number of faults, as well as the increase in reliability as the reciprocal of the MF rate:

$$\frac{\mu_F(3 \cdot N)}{\mu_F(N)} = 3^{-K}; \quad \frac{\zeta_F(3 \cdot N)}{\zeta_F(N)} = 3^{-(K+1)}; \quad \frac{R_F(3 \cdot N)}{R_F(N)} = 3^{K+1}$$

	$\frac{\mu_F(3 \cdot N)}{\mu_F(N)}$	$\frac{\zeta_F(3 \cdot N)}{\zeta_F(N)}$	$\frac{R_F(3 \cdot N)}{R_F(N)}$
$K = 0{,}3$	0.72	0.24	4.17
$K = 0{,}5$	0.56	0.19	5.19

The number of faults is reduced to between 56% and 72%, and the failure rate due to unresolved faults to between 19% and 24%.

We usually only have a rough idea of the range of K. Its influence on the reduction in the number of faults is significant, but it has only a moderate influence on the improvement in reliability.

b) *Assuming we ignore malfunctions caused by disruptions, how much more reliable becomes the systems if the number of staff in the testing department were tripled?*

If tripling the number of staff implies a threefold increase in the testing workload, reliability can be expected to increase approximately four- to fivefold.

$\zeta_F(N)$ yMalfunction rate caused by faults depending on the number of tests.
$\mu_F(N)$ yExpected number of faults not detected and eliminated after N tests.
K yShape parameter of the distribution of the malfunction rate ($0 < K < 1$).
$R(N)$ yReliability after elimination of all faults detectable with the N tests.

3.6 Effective number of tests

2.89 Effective number of tests

In previous estimates, N is the **number of tests for which all detectable faults are eliminated**; hereinafter, this is referred to as the **effective number of tests**. There are situations in which this is a fraction or a multiple of the number of tests carried out or used to determine coverage:

$$N = C \cdot N_T \tag{2.39}$$

- Isolated test $C \gg 1$: Module tests tend to require far fewer randomly selected tests than a test in the system environment to detect the same internal module faults.
- The failure rate of model faults may, on average, be greater or less than that of actual faults.
- Maturation process $C \ll 1$: Faults in software in use must cause many malfunctions before they are rectified, ...

N yEffective number of tests: the number of tests required to eliminate all detected faults.
N_T yNumber of tests carried out or used to determine coverage.
C yTest scaling, ratio of effective to actual number of tests.

2.90 Relative increase in the number of tests

In the case of a relative increase, the scaling factors C cancel out.

(2.24) $$\mu_F(N) = \mu_F(N_0) \cdot \left(\frac{N}{N_0}\right)^{-K}$$

(2.36) $$R(N) = R(N_0) \cdot \left(\frac{N}{N_0}\right)^{K+1}$$

(2.38) $$S(N) = S(N_0) \cdot \left(\frac{N}{N_0}\right)^{K+1}$$

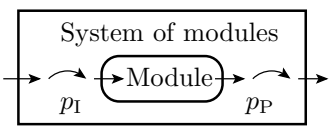
The scaling factor only has an influence when the result depends on the absolute number of tests, e.g. in:

(2.25) $$\zeta_F(N) = \frac{K \cdot \mu_F(N)}{N}$$

$$\zeta_F(N_T) = \frac{K \cdot \mu_F(N_T)}{C \cdot N_T} \tag{2.40}$$

N/N_0 yRelative increase in the effective or actual number of tests.
N, N_T, C yEffective number of tests, actual number of tests, test scaling.
ζ_F, μ_F yMalfunction rate caused by faults. Expected number of faults.
K yShape parameter of the distribution of the malfunction rate ($0 < K < 1$).

2.91 Isolated module testing



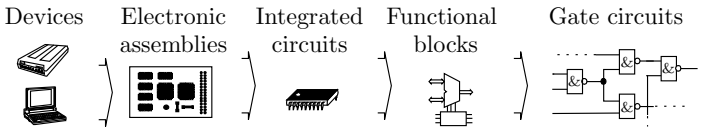
When embedded in a higher-level system, internal module faults are less likely to be triggered and are less likely to be observed than in an isolated test. The failure rate – and thus also the effective number of tests in the isolated test – is typically by a factor of $C \gg 1$ greater than when embedded:

(2.39) $$N = C \cdot N_T$$

A module test therefore typically replaces $C \gg 1$ tests of the upper-level system, but only for the faults within the modules.

p_I, p_P yInfection probability. Propagation probability.
N, N_T, C yEffective number of tests. Number of module tests. Test scaling.

2.92 Hierarchy (see section 2.1.6)

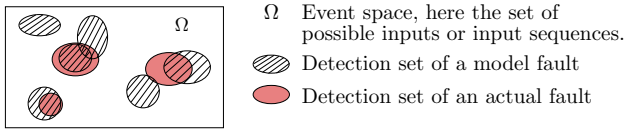


Hierarchical structuring; in software, this also refers to modular programming. The composition of a complex system from subsystems. A prerequisite for

- the reuse of subsystems,
- repair through component replacement and,
- in particular, for **hierarchical testing**:
  - thorough testing of all components before installation and
  - Focusing of overall tests on faults in interaction.

A drastic reduction in the total number of tests required to achieve the same level of reliability.

2.93 Faults and model faults



For the random test, only the ratios of the detection sets are of interest; more precisely, the test scaling factor C, defined as the ratio of the mean detection rate of the model model or criteria to the mean malfunction rates of the actual faults:

- $C > 1$: detection rate greater than malfunction rate,
- $C < 1$: detection rate less than the malfunction rate.

Assuming that, as the number of tests increases, the number of detectable faults remains roughly the same, the effective number of tests is C-times greater than the number of tests used to determine coverage:

(2.39) $$N = C \cdot N_T$$

C yTest scaling.
N, N_T yEffective number of tests for actual faults. Number of tests for determining coverage.

2.94 Fault model scaling

According to (eq. 2.39), the same expected coverage or permeability requires C times the number of tests:

$$\mu_{FC}(N) = \mu_{FCM}(N_T) \quad \text{for } N = C \cdot N_T \tag{2.41}$$
$$\mu_{FP}(N) = \mu_{FPM}(N_T) \quad \text{for } N = C \cdot N_T \tag{2.42}$$

For model faults with similarly large detection set, the test scaling factor is $C \approx 1$. The actual number of tests is roughly the same as the simulated number.

For coverage criteria with test scaling $C \gg 1$, the test set must be chosen to be C times longer for the same actual coverage.

A similar, but not identical, requirement that a test must cover every criterion with a large number of different tests (slide 2.29). Similar, because C-fold criterion coverage also requires approximately C-fold the number of tests.

μ_{FC}, μ_{FCM} yExpected fault coverage for actual fault or modelled faults.
μ_{FP}, μ_{FTM} yExpected fault permeability, for actual fault or modelled faults.
N_T, C yNumber of tests carried out to determine coverage. Test scaling.

3.7 Maturation process

2.95 The problem of ever-larger IT systems

Number of faults arising from the development processes (slide 2.126):

(2.57) $$\mu_{CF} = \xi_C \cdot M_C$$ Added

to this are the new faults arising during repair (sec.3.3.2):

(3.36) $$\mu_{FCR} = \mu_{CF} \cdot (1 + \xi_{RFR})$$ Reliability, ign

(2.32) $$\mu_F(N_0) = \mu_{FCR} \cdot FP_{PT}$$

(2.24) $$\mu_F(N) = \mu_F(N_0) \cdot \left(\frac{N}{N_0}\right)^{-K}$$

(2.35) $$R(N) = \frac{N}{K \cdot \mu_F(N) \cdot MP}$$

$$R = \frac{N^{K+1}}{K \cdot \xi_C \cdot (1+\xi_{RFR}) \cdot FC_{PT} \cdot M_C \cdot MT \cdot N_0^{-K}} \quad (2.43)$$

ξ_C yFault emergence rate in faults per reference unit M_C.
ξ_{RFR} yFault emergence rate in new faults per existing original fault.
M_C yMetric for the effort of creation or the size of the product.
N, N_0 yTotal nuber of tests. Number of dynamic pre-tests.
FP_{PT}, MP yFault permeability pre-tests, malfunction permeability of monitoring under operation.
Pronounce: yζ: zeta, ξ: xi, μ: my.

Transformed to the required number of tests:

$$N = \left(R \cdot M_C \cdot \xi_C \cdot MT \cdot K \cdot FT_{PT} \cdot (1 + \zeta_{RFR}) \cdot N_0^K\right)^{\frac{1}{K+1}} \quad (2.44)$$

M_C grows exponentially over the years. To achieve the same or a higher level of reliability R this must be offset by:

- lower fault emergence rate ξ_C,
- low malfunction permeability MP of the monitoring system, and
- a higher effective number of tests N.

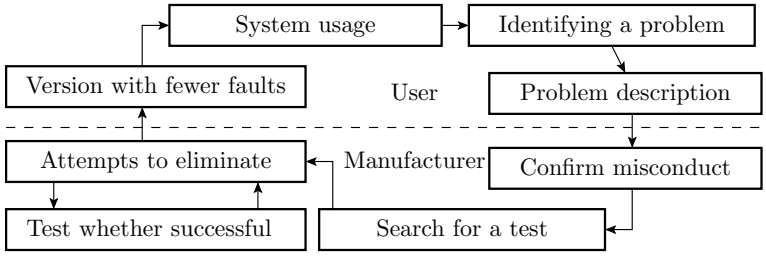
The affordable options for this have largely been exhausted.

What to do to counter the loss of reliability?

Extending the cycle of testing and fault rectification as a maturation process throughout the operational phase.

R, K yReliability. Shape parameter of the distribution of the malfunction rate ($0 < K < 1$).
N yEffective number of tests: the number of tests required to eliminate all detected faults.
M_C yMetric for the effort of creation or the size of the product.

2.97 Maturation process of products under use



Iteration of testing and fault rectification during the operational phase:

1. Identifying problems under use (crashes, malfunctions, etc.).
2. Compile the data to reproduce the problem.
3. Forward the problem description to the producer.
4. Search for tests to reproduce the fault.
5. Resolve the issue via experimental repairs.
6. Release and installation of subsequent versions.

2.98 Probability of fault rectification

The **fault rectification probability** p_{FE} is the conditional probability that, if a problem occurs,

1. the user or the system will detect it and send a change request or bug repert to the manufacturer,
2. problem confirmed by the manufacturer and is given priority,
3. the manufacturer finds tests to prove the fault,
4. identifies and rectifies the underlying fault, and
5. the user installs the update once the fault has been rectified.

Overall, $p_{\mathrm{FE}} \ll 1$ (**very lows**).

Re 2: Generating and sending **problem reports**, preferably automatically. Manual problem reports as change requests for desired changes and erroneous behaviour.

Re 3: Problem reports **are grouped into categories with** suspected **common causes**. Prioritisation by frequency and severity of fault [7].

2.99 Effective number of tests

Maturing products are used by **many users over a long period of time** with countless sample inputs. Effective number of tests:

$$N = p_{\mathrm{FE}} \cdot \mu_{\mathrm{NU}} \cdot \eta_{\mathrm{SS}} \cdot (t_{\mathrm{M}} + t_{\mathrm{V0}}) \quad \text{with} \quad t_{\mathrm{V0}} = \frac{N_{\mathrm{V0}}}{p_{\mathrm{FE}} \cdot \mu_{\mathrm{NU}} \cdot \eta_{\mathrm{SS}}} \tag{2.45}$$

In practice, the effective number of tests does not increase continuously with the duration of the maturation period, but rather in discrete time steps with each version release. Effective number of tests for equal time intervals between updates:

$$N = \underbrace{p_{\mathrm{FE}} \cdot \mu_{\mathrm{NU}} \cdot \eta_{\mathrm{SS}} \cdot t_{\mathrm{VR}}}_{N_{\mathrm{MV}}} \cdot (u + u_{\mathrm{V0}}) \quad \text{with} \quad u_{\mathrm{V0}} = \frac{N_{\mathrm{V0}}}{N_{\mathrm{MV}}} \tag{2.46}$$

N	yEffective number of tests: the number of tests required to eliminate all detected faults.
p_{FE}	yProbability that a fault will be eliminated if it causes a MF.
μ_{NU}	Expected **n**umber of **u**ser.
η_{SS}	ySystem utilisation rate, percentage of available service time used.
t_{M}, t_{V0}	yMaturing time. Equivalent maturing time of manufacturer tests.
N_{V0}	yEffective number of tests prior to version release; approx. number of regression tests.
N_{MV}	yIncrease the effective number of tests in each version interval.

t_{VR} yVersion interval, time between the release of successive versions.
u yVersion number of the maturing object, counting mode 0, 1, 2, ...
Pronounce: yμ: my, η: eta.

2.100 Decline of faults with maturation

This section considers only the simplified case, where no new faults occur during rectification or new faults rectified before the version is released. Based on

(2.24) $$\mu_F(N) = \mu_F(N_0) \cdot \left(\frac{N}{N_0}\right)^{-K}$$

Using Equations 2.45 and 2.46, the expected number of defects decreases by the Kth power of the maturation time or, where release intervals are of equal duration, by the version number:

$$\mu_F(t_M) = \mu_F(t_{M0}) \cdot \left(\frac{t_M + t_{V0}}{t_{M0} + t_{V0}}\right)^{-K} \tag{2.47}$$

$$\mu_F(u) = \mu_F(v) \cdot \left(\frac{u + u_{V0}}{v + u_{V0}}\right)^{-K} \tag{2.48}$$

μ_F yExpected number of faults.
t_M, t_{M0} yMaturation time. Reference naturation time.
t_{V0} yEquivalent maturation time of manufacturer tests.
u, v yVersion number and reference version number of the maturing object.
u_{V0} yRatio of the maturing time before version release to maturing time increase per version.
K yShape parameter of the distribution of the malfunction rate ($0 < K < 1$).

2.101 Malfunction rate, reliability and safety

Decrease in the malfunction rate due to faults with exponent $K+1$:

$$\zeta_F(t_M) = \zeta_F(t_{M0}) \cdot \left(\frac{t_M + t_{V0}}{t_{M0} + t_{V0}}\right)^{-(K+1)} \tag{2.49}$$

$$\zeta_F(u) = \zeta_F(v) \cdot \left(\frac{u + u_{V0}}{v + u_{V0}}\right)^{-(K+1)} \tag{2.50}$$

With appropriate system design and fault handling, and disregarding disruptions, safety is proportional to reliability (slide 2.88), ... Increase in reliability and safety by exponent $K+1$ of the maturation period:

$$\frac{R(t_M)}{R(t_{M0})} = \frac{S(t_M)}{S(t_{M0})} = \left(\frac{t_M + t_{V0}}{t_{M0} + t_{V0}}\right)^{K+1} \tag{2.51}$$

$$\frac{R(u)}{R(v)} = \frac{S(u)}{S(v)} = \left(\frac{u + u_{V0}}{v + u_{V0}}\right)^{K+1} \tag{2.52}$$

R, S yReliablity. Savety (Security).
t_M, t_{M0} yMaturation time. Reference naturation time.
t_{V0} yEquivalent maturation time of manufacturer tests.
u, v yVersion number and reference version number of the maturing object.
u_{V0} yRatio of the maturing time before version release to maturing time increase per version.

2.102 High reliability, legacy code

High reliability and safety require maturation processes involving a very large number of effective tests:

- Monitoring with low false-positive rate,
- functions and an organisational framework for recording and transmitting data on detected malfunctions to manufacturers.
- many users, long maturation period, ...

Systems that have matured over many years possess high levels of reliability that cannot be achieved by other means. They are difficult to replace with new systems.

New or alternative systems are much less reliable in their first few years of use than the systems they replace. If this impairs their acceptance, they will not mature either.

Consequently, there is a significant proportion of legacy code (slide 7.104) in safety-critical areas; see also the Y2K bug [5]. Incorporating legacy code into new products poses a major challenge [3].

Legacy code: Successful, critical production code written in obsolete programming languages running on obsolete operating systems.

2.103 Maturation of user behaviour

A similar increase in reliability and security can be observed over the system's lifetime, even without fault correction.

Complex IT systems often offer many ways of solving a task, not all of which work (slide 1.103). As usage increases, users learn to avoid problematic inputs and to adapt their service requirements to the system's capabilities. Reliability and security increase in a similar way to the duration of use (eq. 2.51).

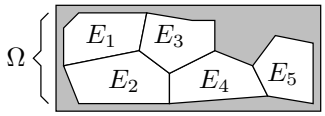
However, the duration of use for reliability growth is limited to that of the individual user. If the user changes, reliability plummets because the count of usage duration starts afresh.

If knowledge about ways to work around errors is shared, e.g. via forums or FAQ pages, the entire user community learns, so that the usage durations of many users are added together.

FAQ yWeb pages containing descriptions of problems and workarounds.

3.8 Usage and test profiles

2.104 Equivalence classes and input profiles



Ω Input space
E_i Equivalence class i
invalid inputs

The **input space** Ω of an IT system comprises all potential inputs and input sequences and is far too large for a complete verification (slide 2.10).

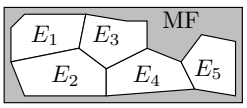
Division into **equivalence classes** E_i with similar usage and processing. Each equivalence class Ei has a function f_i which is executed under condition c_i.

Invalid inputs form equivalence classes for malfunctions. Target behaviour: malfunction handling.

An **input profile** describes the relative frequency of use h_i of the equivalence classes E_i and has a significant influence on the detection probabilities and malfunction rates of existing and modelled faults.

2.105 Usage profile

Operational profile



where $E_i = (c_i, f_i, h_i)$

Measurement programme

if c_1: f_1; H_1^{++};
[else] if c_2: f_2; H_2^{++};
[else] if c_3: f_3; H_3^{++};
...
[else ⟨MF treatment⟩; H_0^{++}]

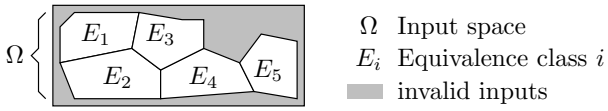
A **usage profile** is an input profile when the system is used for one of its intended purposes. In the case of software, the usage frequencies E_i of the equivalence classes Eican be measured by **instrumentation** using counters for the number of executions H_i:

$$h_i = \frac{H_i}{N}$$

The optional »else« above describes mutual exclusion.

E_i, h_i yEquivalence class i. Selection frequency for equivalence class i.
f_i, c_i yFunction and selection condition for equivalence class E_i.
H_i, N yNumber of executions equivalence class E_i. Total number of executions.
Instrumentation: Adding program code to examine the behaviour.

2.106 Required user profile coverage



(2.36) $$R(N) = R(N_0) \cdot \left(\frac{N}{N_0}\right)^{K+1}$$

To ensure suff
known reliabili

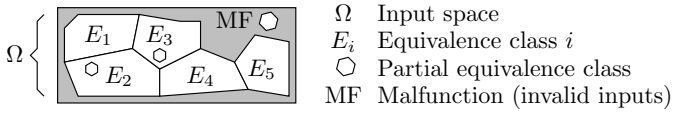
$$N \geq N_0 \cdot \left(\frac{R(N)}{R(N_0)}\right)^{\frac{1}{K+1}}$$

Breakdown of the total number of tests into the numbers of tests per equivalence class

$$N_i \geq h_{i.\max} \cdot N \tag{2.53}$$

R Reliability.
K yShape parameter of the distribution of the malfunction rate ($0 < K < 1$).
N, N_0 yTotal number of tests. Number of tests for which all detected faults are eliminated.
E_i, h_i yEquivalence class i. Selection frequency for equivalence class i.
$h_{i.\max}$ yMaximum frequency of use for all potential usage profiles.
N_i ySufficient number of tests for equivalence class i.

2.107 Test profiles



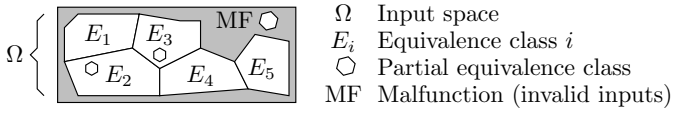
In an iteration of testing and fault elimination, the number of faults decreases with the number of tests by an exponent $0 < K < 1$, and the malfunction rate decrease by an exponent $K + 1$. The remainig faults with increasing test number belong to ever-smaller partial equivalence classes (sec.2.3.2).

Test profiles are fault-oriented selected input profiles that give extreme preference to sub-equivalence classes $E_{\mathrm{i.j}}$ with input conditions $c_{i.j}$ containing faults that are difficult to detect.

Test profile-oriented test selection:

- Search for test profiles that favour faults not yet covered,
- Select a large input sample in accordance with the profile, and
- Determine the coverage using the selected inputs.

2.108 The statistics behind the test profile concept

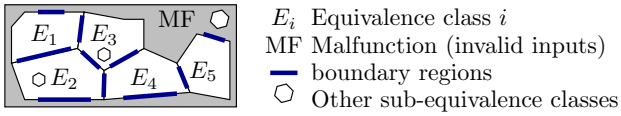


Assume that the input space of an equivalence class E_i comprises 10 variables. The sub-equivalence class $E_{i.j}$ restricts the selection range for each variable to one-tenth. Then the input set of $E_{i.j}$ is 10 orders of magnitude smaller and the selection probability for each value is 10 orders of magnitude greater.

For faults assigned to the partial equivalence class $E_{\mathrm{i.j}}$, the probability of detection increases proportionally. According to (eq. 2.39), the effective number of tests increases in proportion to the probability of detection; in the example, by a factor of $C = 10^{10}$.

Questions remain to be answered regarding the search criteria and the search algorithm for partial equivalence classes and test profiles.

2.109 Boundary regions

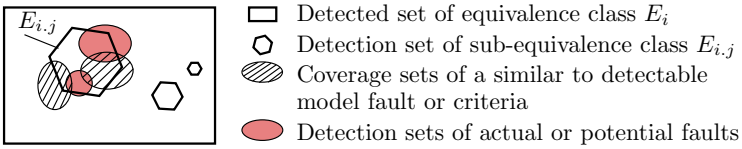


Typically, the boundaries of equivalence classes are those partitial equivalence classes with faults that are difficult to detect.

Boundaries can be visualised as $n-1$-dimensional surfaces in the n-dimensional input space. They encompass all input values that lie close to these surfaces. To illustrate, thin boundary surfaces contain far fewer input values than the enclosed volumes. The effective number of tests for the associated potential errors increases in inverse proportion to the reduction in the size of the region.

The input $c_{i.j}$ conditions for boundary values are derived from the input conditions of the enclosed equivalence classes.

2.110 Model faults and test coverage criteria



For other partial equivalence classes with potential faults that are difficult to verify, only model faults or coverage criteria can be specified that share verification conditions for reachability, infection and propagation with actual faults.

Figuratively speaking, input conditions or tests are sought for the striped or hexagonal areas in order to randomly hit the red areas of the unknown detection sets of associated faults.

- Estimation of selection conditions c_{ij} for partial equivalence classes E_{ij} and random generation of inputs that satisfy c_{ij}.
- Generating input values for model fault detection or criterion coverage without resorting to partial equivalence classes.

2.111 Model calculation for an ideal case

The different usage and test profiles tend to detect completely different faults with a high degree of probability. Ideally, the N_i tests for each input profile i increase the effective number of tests by a factor of N_i/N_0 compared with the same small baseline number of tests N_0. Total expected fault permeability for independent fault permeability fractions per equivalence class:

$$\mu_{\mathrm{FP}} = \frac{\mu_{\mathrm{F}}}{\mu_{\mathrm{F}}(N_0)} = \prod_{i=1}^{\#IP} \left(\frac{N_i}{N_0}\right)^{-K}$$

For the same number of tests for all input profiles $N_i = N$:

$$\mu_{\mathrm{FP}} = \left(\frac{N}{N_0}\right)^{-\#IP \cdot K} \tag{2.54}$$

N_0, N yReference test count. Number of tests per operation profile incl. N_0.
K yShape parameter of the distribution of the malfunction rate ($0 < K < 1$).
$\#IP$ yNumber of different input profiles used for testing.
$\mu_F(N_0)$ yExpected number of non-detected faults for the reference test number N_0.
μ_F yExpected number of non-detected faults after all tests.
$\mu_{FT}(\zeta, N)$ yExpected fault permeability for faults with MF rate ζ and N tests.

2.112 Reliability improvement

(2.54) $$\mu_{FP} = \left(\frac{N}{N_0}\right)^{-\#IP \cdot K}$$

For

(2.35) $$R(N) = \frac{N}{K \cdot \mu_F(N) \cdot MP}$$

the number of tests N is that per profile. Number of unresolved faults:

$$\mu_F(N) = \mu_F(N_0) \cdot \mu_{FP}(N/N_0)$$

Increase in reliability with the number of profiles and the number of tests per profile:

$$R(\#IP, N) = \frac{N}{K \cdot \mu_F(N_0)} \cdot \left(\frac{N}{N_0}\right)^{\#IP \cdot K} \sim N^{1+\#IP \cdot K} \tag{2.55}$$

Provided that each test profile reduces the proportion of remaining undetectable defects by approximately $(N/N_0)^{-K}$, reliability increases with the number of tests per profile with exponent $1 + \#OP \cdot K$.

N_0, N yReference test count. Number of tests per operation profile incl. N_0.
K yShape parameter of the distribution of the malfunction rate ($0 < K < 1$).
$\#IP$ yNumber of different input profiles used for testing.

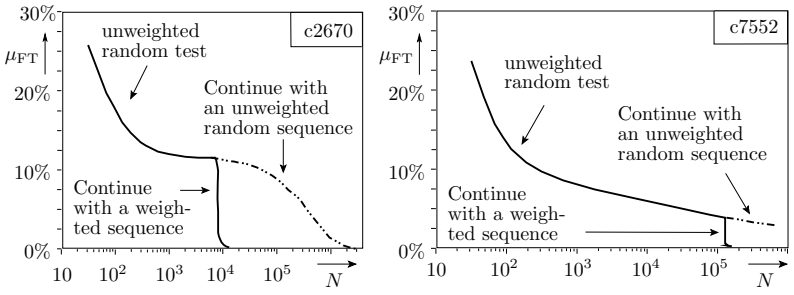
2.113 Weighted random test

Self-testing of large digital circuits. Test verification by stuck-bit faults. State bits as test inputs and outputs. Input profile settings via bit weighting, whre $g(x_i)$ Frequency that the input bit i is set to one. A very large number of tests per profile.

- Generation the model fault set, i.e. stuck-at faults
- For several thousand to millions randomly selected of tests:
  - randomly select unwighted test ($g = 50\%$)
  - mark off all detectable faults of the set
- Look for tests for any stuck-at faults not yet covered. Adjusting the weighting $g(x_i)$ for each input bit x_i according to the frequency of ones in the tests found.
- For several thousand to millions randomly selected of tests:
  - Selection of weighted bit values
  - mark off also all detectable faults.

Further details later in this section 6.3.4.

$g(\ldots)$ yWeighting, frequency of occurrence of the signal value 1.



Two test profiles, each with an expected fault permeability of

$$\mu_{FP.i} \approx (N/N_0)^{-K}$$

Total fault permeability (Eq. 2.54 with $\#IP = 2$):

$$\mu_{FP} = \mu_{FP.1} \cdot \mu_{FP.2} \approx (N/N_0)^{-2K}$$

The graph does indeed give the impression that, following a change in weighting, the effective test number N restarts newly from N_0.

μ_{FT}, N yExpected stuck-at fault permeability. Number of tests per test profile.
K, N_0 yShape parameter of the rate distribution ($0 < K < 1$). Reference number of tests.
$\#IP$ yNumber of different input profiles used for testing.

2.115 Test profile-based test selection for SW

Preparations

- Compilation of user profiles and coverage criteria.
- Determination of minimum coverage figures for all equivalence classes, equivalence class boundaries and other criteria.

A: For all input profiles derivable from specifications: Instrument the test object with coverage counters. Generate a large number of random inputs. Execute and count the coverage.

B: As in A, but only with test profiles that give extreme priority to the frequency of occurrence of criteria that have not been sufficiently covered up to that point.

Practical implementation requires generisable randomisation functions for inputs and the object space, translation versions with coverage checking and result verification, ... (see slide set 5 below and 7).

Summary

2.116 Pre-tests and reliability testing

On average, a total of μ_{FCR} faults occur during the development and fault- correction processes. Of these, detect the pre-tests

- static tests (reviews, syntax, etc.)
- dynamic rough tests, tests under specific conditions, ...

with N_0 dynamic tests included, a proportion FC_{PT}, which is rectified. Remaining number of faults and failure rate:

(2.32) $$\mu_F(N_0) = \mu_{FCR} \cdot FP_{PT}$$

(2.33) $$\zeta_F(N_0) = \frac{K \cdot \mu_F(N_0)}{N_0} \quad \text{for } \zeta_F(N_0) \ll 1$$

(𝕃) The subse
expected numb

(2.24) $$\mu_F(N) = \mu_F(N_0) \cdot \left(\frac{N}{N_0}\right)^{-K}$$

(2.26) $$\zeta_F(N) = \zeta_F(N_0) \cdot \left(\frac{N}{N_0}\right)^{-(K+1)}$$

Effective number of tests: (equivalent) number of tests for which all detectable faults are removed.

2.117 Shape parameter, reliability and safety

The shape parameter of the malfunction rate distribution can be estimated both from the decrease in the number of faults and from the decrease in the malfunction rate with effective number of tests N/N_0:

(2.27) $$K = -\ln\left(\frac{\mu_F(N)}{\mu_F(N_0)}\right) / \ln\left(\frac{N}{N_0}\right)$$

(2.28) $$K = \ln\left(\frac{\zeta_F(N_0)}{\zeta_F(N)}\right) / \ln\left(\frac{N}{N_0}\right) - 1$$

(𝔹) With appropriate system design and problem handling, reliability and safety increase by the $K+1$st power of the effective number of tests:

(2.36) $$R(N) = R(N_0) \cdot \left(\frac{N}{N_0}\right)^{K+1}$$

(2.38) $$S(N) = S(N_0) \cdot \left(\frac{N}{N_0}\right)^{K+1}$$

2.118 Effective and actual number of tests

(2.39) $$N = C \cdot N_T$$

Modular testing: For faults within a module, the effective number of module tests is much greater than that of tests in the overall system:

(2.39) $$N = C \cdot N_T$$

That's why modules are thoroughly tested before installation.

Fault-model-specific scaling. Expected fault coverage is approximately the model fault coverage of C times the number of tests:

(2.41) $$\mu_{FC}(N) = \mu_{FCM}(N_T) \quad \text{for } N = C \cdot N_T$$

$C > 1$: model faults have higher malfunction rates or criteria have a higher coverage rate than actual faults; C < 1 is the reverse.

For maturation processes, $C = p_{FE} \ll 1$ applies; however, in terms of the number of tests, the service usage of many users accumulates over a long maturation period.

Effective number of tests: (equivalent) number of tests for which all detectable faults are removed.

2.119 Reifeprozess

(𝕃) Continuation of the fault-elimination iteration during the deployment phase, with users acting as testers. Increase in the effective number of tests:

- with the time of maturation
- (2.45) $$N = p_{FE} \cdot \mu_{NU} \cdot \eta_{SS} \cdot (t_M + t_{V0}) \quad \text{with } t_{V0} = \frac{N_{V0}}{p_{FE} \cdot \mu_{NU} \cdot \eta_{SS}}$$
- equal lengt version intervals with the version number:
- (2.46) $$N = \underbrace{p_{FE} \cdot \mu_{NU} \cdot \eta_{SS} \cdot t_{VR}}_{N_{MV}} \cdot (u + u_{V0}) \quad \text{with } u_{V0} = \frac{N_{V0}}{N_{MV}}$$
- Decrease in the number of faults by exponte $0 < K < 1$:
- (2.47) $$\mu_F(t_M) = \mu_F(t_{M0}) \cdot \left(\frac{t_M + t_{V0}}{t_{M0} + t_{V0}}\right)^{-K}$$

  (2.48) $$\mu_F(u) = \mu_F(v) \cdot \left(\frac{u + u_{V0}}{v + u_{V0}}\right)^{-K}$$
- Decrease in the malfunction rate by en exponte $K+1$:
- (2.49) $$\zeta_F(t_M) = \zeta_F(t_{M0}) \cdot \left(\frac{t_M + t_{V0}}{t_{M0} + t_{V0}}\right)^{-(K+1)}$$

  (2.50) $$\zeta_F(u) = \zeta_F(v) \cdot \left(\frac{u + u_{V0}}{v + u_{V0}}\right)^{-(K+1)}$$

2.120 Reliability and Safety (𝕃)

With appropriate system design and problem-solving, the malfunction rate due to disturbances is negligible and safety is proportional to reliability; both increase by the $K+1$th power of the maturation time or version number:

$$\frac{R(t_M)}{R(t_{M0})} = \frac{S(t_M)}{S(t_{M0})} = \left(\frac{t_M+t_{V0}}{t_{M0}+t_{V0}}\right)^{K+1} \tag{2.51}$$

$$\frac{R(u)}{R(v)} = \frac{S(u)}{S(v)} = \left(\frac{u+u_{V0}}{v+u_{V0}}\right)^{K+1} \tag{2.52}$$

- Long maturation processes spanning decades achieve very high levels of reliability and security that would otherwise be almost impossible to attain.
- Old, long-matured software is difficult to replace because an equivalent replacement must first mature over a long period amongst many users.

There is also a maturation process for user behaviour. This means that the reliability and safety of systems also increase as users get used to working with it over timewith the length of time a user has been using it.

2.121 Input, usage and test profiles

(𝕃) Input profiles describe the relative frequency of use of equivalence classes, i.e. the input sets with comparable processing times. Sufficient reliability requires a sufficiently large number of tests for each equivalence class corresponding to a usage profile.

(𝕃) Test profiles are input profiles selected on the basis of faults, and give extreme priority to faults that are difficult to detect. Test profiles must also be covered by a large number of tests.

The test selection could, for example, start with a user profile and be repeated until all criteria have been covered by a sufficient number of tests::

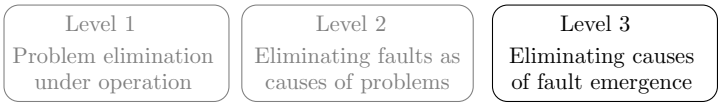
1. Random selection of many tests corresponding to the profile
2. Search for a new test profile for criteria not yet covered.

For digital circuits, a combination of unweighted and fault-oriented weighted random tests works well.

Also of interest for software. Ideas for implementation will follow later.

4 Fault prevention

2.122 Fault prevention



Fault prevention means to reduce fault emergence rates by eliminating problems in the creation processes.

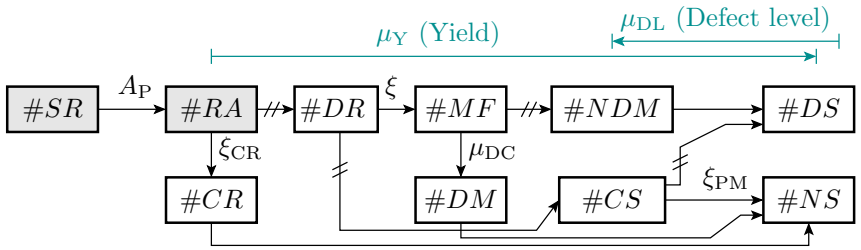
Creation processes can be viewed as service providers, much like IT systems, including processing steps, checks and troubleshooting. Unfixed problems cause faults in the products being created.

Addressing root causes once again at all three levels:

1. Process monitoring, countermeasures for identified problems.
2. Testing and troubleshooting before and during process usage.
3. Fault prevention whilst the processes are created.

We will focus on process maturation, i.e. resolving issues whilst the processes are in use.

2.123 Creation process without repair



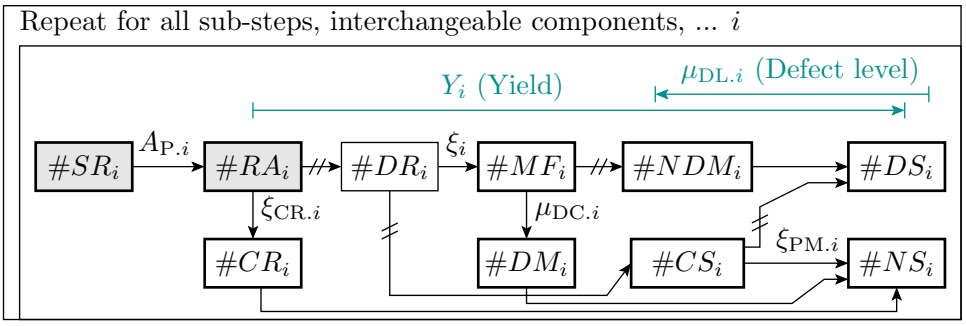
SR	Product ordered	DM	Fault detectable	A_P	Process availability
RA	Order accepted	NDM	Fault not detectable	ξ_{CR}	Cancellation rate
CR	Processing aborted	CS	Product fault-free	ξ	Defect emergence rate
DR	Product manufactured	DS	Product shipped	μ_{DC}	Defect coverage test
MF	Product defective	NS	Product rejected	ξ_{PM}	Pantom defect rate

A product (or design) is requested:

- If process available and order accepted, a product is deliverd
- at a rate equal to the expected yield.
- The products supplied have a defect level of μ_{DL}.

The labels for the counts are taken from the CVA graph for a service (slide 1.66), to illustrate the **similarity of both models**.

Pronounce: yμ: my, ξ: xi.



A count-value assignment graph can only count one fault per requirement. Suitable only for fault rectification by replacement.

If a high number of faults is expected, **fault rectification by repair** (sec. 3.3.2). Modelling the fault emergence requires a **count-assignment graph for each potential fault**, replaceable component, ... and summing up the expected number of faults for many sub-services.

Fault prevention means reducing fault emergence rates:

- through monitoring and troubleshooting,
- as a maturing process whilst the process is in use, ...

An interesting approach, though not be explored in depth here.

2.125 Problem indicators and metrics

In descriptive statistics, metrics are parameters or indicators used to describe or measure something quantitatively. In our context, these are expected, increased, decreased, ... rates and counts for problems.

Countable and measurable parameters of development processes:

- Development effort in work steps, working hours, NLOC, transistors, documentation pages, ...
- Faults detected per development effort, ...

Estimated fault emergence rate:

$$\xi_C = \frac{\mu_{CF}(M_C)}{M_C} \tag{2.56}$$

Expected number of defects in future products:

$$\mu_{CF} = \xi_C \cdot M_C \tag{2.57}$$

ξ_C	yFault emergence rate in faults per reference unit M_C.
M_C	yMetric for the effort of creation or the size of the product.
μ_{CF}	yExpected number of faults from the creation processes.
NLOC	yNetto lines of code, number of code lines without comments and blank lines).
Pronounce:	yμ: my, ξ: xi.

Example 2.4 Programming faults

$\xi_{NLOC} = 30$ faults / 1000 NLOC, Programme size $M_{NLOC} = 2000\,\text{NLOC}$.

What is the expected number of programming faults before testing and debugging?

$$\mu_{CF} = \xi_{NLOC} \cdot M_{NLOC} = \frac{30\,\text{faults} \cdot 2000\,\text{NLOC}}{1000\,\text{NLOC}} = 60\,\text{faults}$$

Example 2.5 Circuit faults

$\xi_{Tr} = 1$ fault per 10^6 transistors. Circuit with $M_{Tr} = 10^5$ transistors.

What is the expected number of faults per circuit before the recognisably defective circuits are sorted out?

$$\mu_{CF} = \xi_{Tr} \cdot M_{Tr} = \frac{1\,\text{fault} \cdot 10^5\,\text{transistors}}{10^6\,\text{transistors}} = 0{,}1\,\text{faults}$$

NLOC	yNetto lines of code, number of code lines without comments and blank lines).

2.127 Problem-sensitive metrics

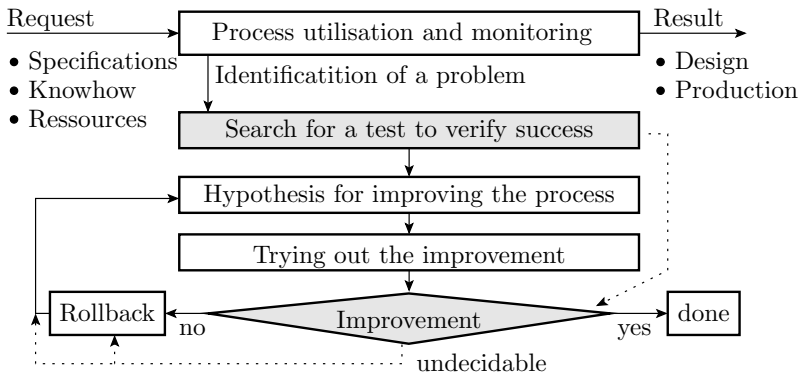
Empirical models suggest that there is a disproportionate increase in the fault emergence rate from around 3 pages of source code per function module, because the designers lose track of the big picture.

The number of text pages per functional module appears to be a metric exhibiting a kind of toggle behaviour between unproblematic and problematic. Problems indicated by metrics with toggle behaviour are best best mitigated by enforcing rules of good practice or avoiding anti-patterns (see section (sec. 7.2.6).

Identifying problem-sensitive metrics whose values can predict, with a high degree of accuracy, the presence or absence of problems is an important approach to fault prevention.

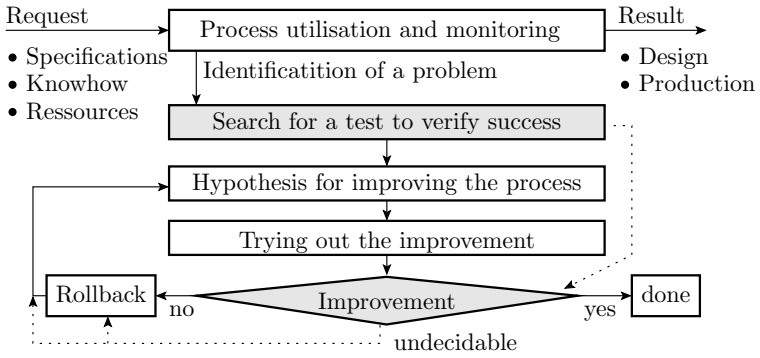
4.1 Maturation of processes

2.128 Maturation of creation processes



If a problem is noticed whilst using the process, look for ways to verify its success after attempting to debug it.

Problem-solving as an iteration. Formulation and testing of hypotheses aimed at reducing the frequency of problems. Success monitoring.



Potential bad decisions in the improvement evaluation:

- False positive: Stopping debugging without resolving the problem.
- False negative: Rollback of improvements.
- New, undetectable problems.

The rate at which identified problems are resolved and the rates at which new problems arise depend largely on the frequency of malfunctions of the success verification.

2.130 Ideal success monitoring, determinism

- Repair tests to verify that problems have been resolved and
- regression tests to rule out undetected new problems

guarantee that the problem has been resolved and largely rule out new, undetectable problems.

Prerequisite: determinism:

> For the same specifications identically reation flow, generating identical products with the same faults.

In contrast to IT systems, determinism in generation processes is a difficult-to-achieve ideal. Degrees of predictability of creation flow and product properties:

1. automated,

2. computer-assisted,
3. well-established manual procedure,
4. experimental manual procedure.

Unpredictability fosters poor decision-making in success monitoring.

2.131 Statistical success monitoring

x_{AV}	$\alpha = 2\%$	$\alpha = 20\%$
$x_{AV} = 10$	$2{,}63 \leq \mu_x \leq 17{,}3$	$6{,}84 \leq \mu_x \leq 13{,}2$
$x_{AV} = 100$	$76{,}7 \leq \mu_x \leq 123$	$90 \leq \mu_x \leq 110$
$x_{AV} = 1000$	$926 \leq \mu_x \leq 1073$	$968 \leq \mu_x \leq 1032$

Without determinism, success can only be measured statistically by counting positive and negative experiences. The table shows the predictable expected value ranges $[\mu_{x.\,min}, \mu_{x.\,min}]$ for normally distributed count values x_{AV} as a function of the error probability α (slide 4.76).

Meaningful success checks require hundreds or thousands of process repetitions. Processes whose success can only be check by many repetitions and with large error probabilitis mature very slowly and reach a state of saturation.

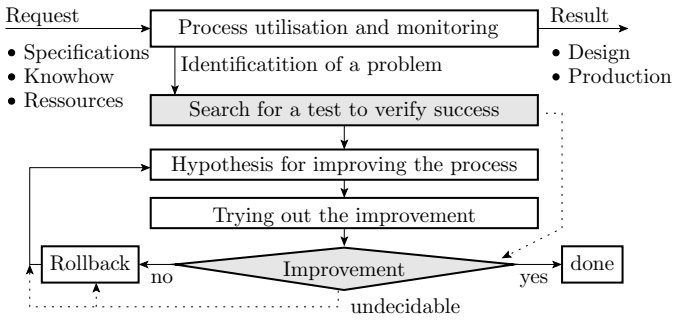
Targeted improvement is only possible if it can be monitored.

---

x_{AV}, n_{AV} yActual count value. Number of associated counting attempts.

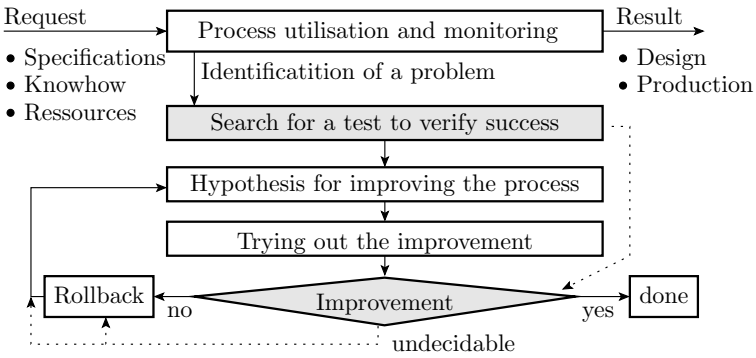
μ_x, α yExpected count value when repeating the experiment. Error probability.

2.132 High rate of wrong decisions



If even the positive and negative experiences to be counted are not precisely defined, ... improvement processes are nevertheless desired due to its high economic value.

Even if a successful check shows just over 50 per cent correct decisions, a process will improve over time, albeit very slowly. However, if more than 50 per cent of decisions are wrong, there will be a gradual deterioration.



Verifiable indicators of whether a process is likely to experience stagnation, **improvements or deterioration** over a longer period:

- Is there a procedure in place for monitoring and resolving identified problems?
- How, and by means of which metrics, are the process and the outcomes of improvement initiatives monitored??
- **Bad decision rate** above, approximately or below **50 per cent**?

4.2 Process capabilities

2.134 Process capabilities

Creation processes have the capability to produce products and designs with specific characteristics. These characteristics include how good, how accurate, how cheap, how free from defects, ...

Scientific and technological progress is an **iterative process of** creation, utilisation and **refinement of process capabilities**:

- Creation: (Further) development of tools, methods, controls (including measurement methods), programming languages, theories, models, ... separate from the use of the process.
- Usage: New capabilities are acquired. When it comes to fault prevention, new process capabilities represent not only progress but also new problems that must be circumvented or eliminated through a process of »learning from mistakes«.
- The experience gained from using the process is incorporated into the development of improved and new capabilities.

2.135 Die The capacity to mature

A maturation process – or, more generally, learning from one's mistakes – depends on certain prerequisites::

- A large number of repetitions of comparable tasks, so that identified problems can be rectified for subsequent process runs.
- Attempts to resolve problems.
- Stattfinden von Problembeseitigungsversuchen.
- Outcome monitoring of problem resolution.

Important process capabilities:

- Degree of repeatability and
- Quality of checks.

The CMMI (Capability Maturity Model Integration), for example, defines five capability levels for classifying processes, organisations, etc., according to their ability to learn from their mistakes.

2.136 The CMMI capability levels

1. Repeated processes, undocumented. Enables individuals to learn from their mistakes.
2. Documentation of processes and problems observed. Enables cross-team learning from mistakes.
3. Management and monitoring of processes. Enables a focus on reliably reaching desired objectives (duration, quality, etc.).
4. Quantitative management: definition and measurement of key performance indicators. Enables the monitoring of maturity progress.
5. Documented maturation process. Continuous process improvement through quantitative feedback from the process.

The higher the maturity level, the more likely it is that

- occurring problems will be resolved, and the faster
- the fault emergence rate will decrease over the operational life.

Capabilty levels are assessed using questionnaires. It is time-consuming and unreliable, but clearly useful.

A policy of small steps.

4.3 Centring, improvement

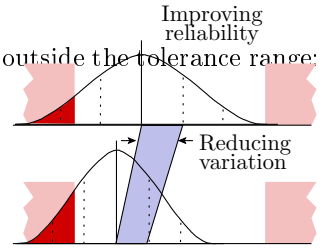
2.137 Process centring

A simple example of the complex interrelationships between

- capability improvement,
- process maturation and
- the resulting temporal trends in fault emergence rates

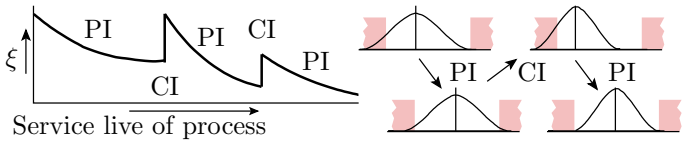
is a mechanical manufacturing step involving a parameter subject to variation, e.g. the diameter of a drill hole.

The fault emergence rate here is the probability that the parameter value lies outside the tolerance range:



- Process centring: Shifting the distribution to the centre of the tolerance range using adjustment options.
- Capability improvements: Reducing variation through technological innovations – new machinery, processes, etc.

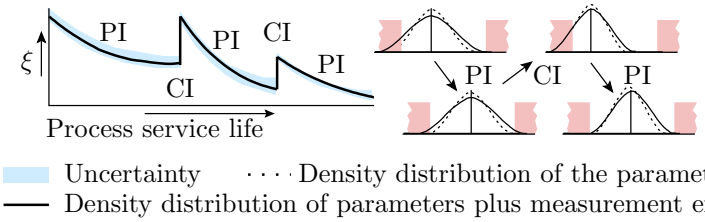
2.138 The sawtooth pattern of fault emergence



- The improvement in capability makes it possible to meet tolerances more accurately, but when new designs are introduced, centering is lost. Sharp increase in the fault emergence rate.
- During process operation, readjustment of the settings to shift the expected value to the centre of the tolerance window. Reduction in the fault energence rate.

CI, PI yCapability improvement, process improvement.

2.139 Estimation and measurement accuracies

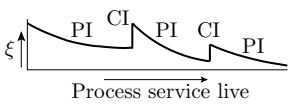


Measurement and estimation inaccuracies lead to mistakes in the fault elimination process:

- Rollback of Improvements and
- and no reversal of deteriorations.

The smaller the estimation and measurement errors, the greater the process's ability for fault prevention.

CI, PI yCapability improvement, process improvement.



- Improving capabilities in larger time steps and
- maturing through »learning from mistakes« in small steps

is typical of all technological processes, including the design and manufacturing processes of IT systems.

The model of the alternating sequence of capability improvement and maturation also provides numerous other useful insights::

- Sufficient maturation time between capability improvements..
- Only what is controllable will improve over time.
- The highest-quality products are often those produced shortly before a technological innovation, ...

The perspective of capability and maturation processes also offers interesting insights into

- the development of organisational processes within institutions,
- the evaluation of management strategies,
- the quality of knowledge transfer in educational institutions, ...

4.4 Process models

2.141 The concept of technology and projects

Repeated development processes are also referred to as technologies[(1)]. Technologies mature through the frequent repetition of similar processes, during which they are monitored and any identified problems are rectified.

What about projects:

- Manual, creative parts of a design process and
- Construction of prototypes, demonstrators, ... ?

A project is a unique planned endeavor to achieve a specific goal, create a unique product or service, ... within defined limits like time, cost, and scope.

From the perspective of fault prevention, projects lack the frequent repetition of similar processes that is needed to learn from identified faults.

Does this exclude projects from fault prevention?

(1) yThe term *Technology* was first used by Göttinger Professor Johann Beckmann (1739-1811) in the textbook "Grundsätze der teutschen Landwirthschaft" (Principles of German Agriculture).

2.142 Process models

Standardisation of workflow for large classes of projects

- to minimise effort, improve predictability and
- to achieve fault prevention by »learning from faults«.

Typical elements of process models include:

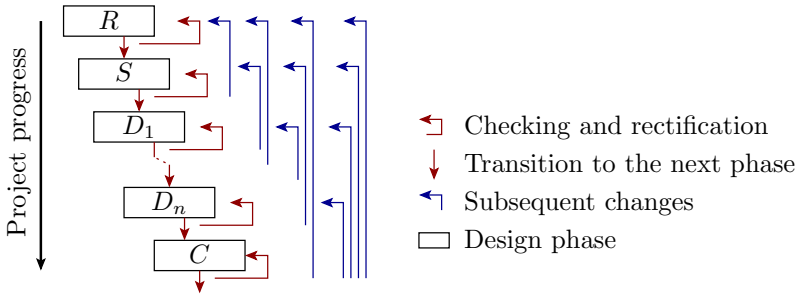
- Breakdown into phases,
- Reference procedures,
- Definition of interim and final checks, ...

Software process models divide the design into phases:

- Several phases for compiling and defining objectives.
- Phases for identifying a suitable architecture, a test solution and for breaking it down into programming and other sub-tasks.
- Design and integration of numerous sub-components.
- Phases for testing, deployment, ...

Block: Fault prevention in project work involves continuous empirical improvement, i.e. the maturing of the process [model].

2.143 Phase models



Adjustment parameters for process improvement:

- organisation of workflow in the phases,
- required tests, documentation, etc. at phase transitions,
- permitted subsequent changes(1), ...

R, *S* yRequirements analysis. Specification (Proposed solutions to meet the requirements).

D_i, *C* yStages of the pre-coding design. Coding.

(1) Subsequent changes can multiply the total number of faults; see the example on the slide 3.121.

Design Phase: Set of design subtasks to be completed, with results to be verified.

2.144 Evaluation of process models

Maturity, as a complex, labour-intensive stochastic process, only yields improvements if these can be verified.

Question arising from this

Which measurable or estimable parameters indicate an improvement in a process model??

These parameters must be comparable across different real-world projects and process models:

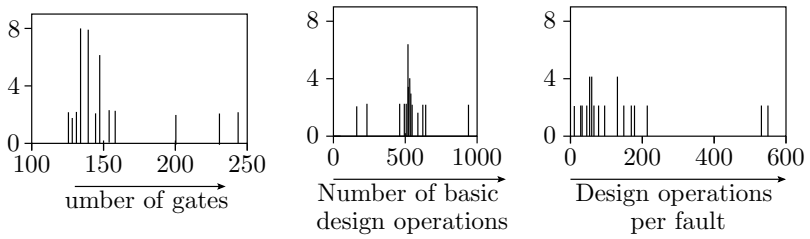
- duration, costs relative to project size,
- amount of work per fault incurred, ..., ...

Expected values, variances, scalability of project size, ...

Significant conclusions about process models require the observation of thousands of projects following comparable procedures.

Process model: Standardisation of procedures for large classes of projects in order to achieve a high number of repetitions as a prerequisite for learning from identified faults.

2.145 An experiment [1]



A group of 72 students was tasked with developing a gate circuit from a PLA (Programmable Logic Array) description and entering it into a CAD system via a GUI. For each design, the basic design operations, the number of gates and the design faults were counted. Basic design operations were the placement of a gate on the screen, the drawing of a connection, ...

The result was the count distributions shown above.

2.146 Conclusions from the experiment

Assuming the experiment is repeated in exactly the same way at other universities:

- Here too, the same statistics will be calculated for each student, and
- the distribution, mean and variance will be compared.
- Are the differences statistically significant?

If the comparison results show significant differences, it would be possible to conclude which university provides students with a better education for this task.

4.5 Quality and Creativity

2.147 Quality and Creativity

Quality requires the prevention of faults. Preventing faults requires

- a high repetition rate of identical or similar tasks,
- standardised workflows yielding reproducible results,
- the logging of all irregularities and problems, ...

Creativity requires »uniqueness«:

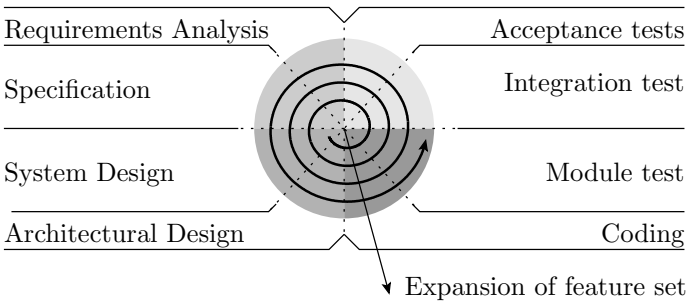
- introducing new concepts,
- trying out new approaches,
- flexible adaptation to changing requirements.

A resulting question

Quality and creativity place conflicting demands on the design of workflows. IT design requires both quality and creativity. How can both be combined within a process model?

2.148 Agile process models

Agile process models aim to provide a framework for projects in which client requirements, objectives, approaches, etc. evolve alongside the project. Less rigid. More creative freedom. Example: the spiral model:



Breaking down into several iterations of a phased model.

Process model: Standardisation of procedures for large classes of projects in order to achieve a high number of repetitions as a prerequisite for learning from identified faults.

... several iterations of a phased model.

- Iteration 1: Specification of basic requirements, design, coding, testing, ..., acceptance and deployment.
- Iteration 2 to n: Brainstorming and refinement of additional requirements and changes. Design through to deployment.

Objective:

- Minimising the number of steps involved and the number of faults generated per phase model run.
- Creative freedom in the form of collecting ideas for the next phase of the model run.

In existing system components, only faults should be rectified. New features, ideas, etc. will be incorporated into new system components. This requires forward-looking planning and a proper system architecture(sec.7.2.1). Every iteration requires testable milestones. This needs proper test solutions (see also test-driven design).

TDD yTest Driven Developement: Planning and designing tests before or in conjunction with the implementation of the function.

Agile Process Models: Dividing development into multiple iterations of a rigid phase-based model. Applying lessons learned to subsequent iterations.

2.150 A detour into learning processes

At school and when learning practical skills, a significant proportion of the teaching and training focuses on process models:

- arithmetic, writing, craftsmanship, programming, ...
- Assessment in terms of service output per man-hour and time.

This know-how that has matured over centuries.

Stages of knowledge transfer at universities:

1. Introduction to the subject: reading, explaining, ...

   Lectures, seminars, self-study, ...

2. Practice until the results are predictable.

   Practicals, exam preparation, laboratory practicals, etc.

3. Professionalisation: Process monitoring. Eliminating shortcomings and problems in workflows.

   this is only possible whilst practising one's profession within one's own limited scope of practice.

(1) Also, assessment based on the amount of work completed per exam duration and the number of faults per amount of work.

2.151 Cross-link to externally funded projects

- For graduates, the professionalisation phase starts in industry and applies only to their specific field of work.
- University graduates and students are not trained to carry out tasks with a low fault rate.
- In industrial software projects, academics tend to produce more faults.
- The costs of fixing faults must be borne by the industrial partner.
- That is why it is usually not cost-effective for industry to involve universities and students in software development.
- Industrial student projects are for educational purposes.
- Third-party-funded research is valuable for the transfer of expertise, literature reviews, demonstrators, ... but it is a poor business for projects where problems have to be fixed.

Demonstrator: Simplified implementation to investigate and demonstrate feasibility.

2.152 Quality assurance at our university

The introduction of the Master's and Bachelor's degree system (Bologna Process) aims to establish standardised procedures, comparable qualifications, ...

This involves establishing fundamental process capabilities:

- a high number of repetitions of comparable procedures,
- Process settings to try out,
- monitorig of key performance indicators,

to establish a maturing process for the education.

How does this work at our university:

- Which parameters are recorded for process monitoring?
- Where can process models be identified?
- How to ensure that creativity does not suffer?

Fault prevention offers interesting perspectives on technologies, institutions and public authorities, right through to our overall scientific and technological progress.

Summary

2.153 Fault emergence

- Faults arise during the design, manufacturing and repair processes, alongside the products themselves.
- Like IT systems, creation processes can be modelled as service providers with delivery rates (yield), malfunction rates (fault occurrence rate), ...
- In practice, fault emergence rates are expressed as a percentage of the measured values of metrics:

(2.57) $$\mu_{\mathrm{CF}} = \xi_{\mathrm{C}} \cdot M_{\mathrm{C}}$$

(2.56) $$\xi_{\mathrm{C}} = \frac{\mu_{\mathrm{CF}}(M_{\mathrm{C}})}{M_{\mathrm{C}}}$$

2.154 Fault prevention, determinism

Fault prevention is the elimination of faults within the creation processes in which they arise. Focus on process management, i.e. fault rectification during the operational phase, comprising the following elements:

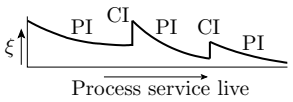
- Process utilisation and monitoring,
- Problem identification and investigation,
- Problem resolution through experimental repair as an iterative process involving troubleshooting attempts and verification of success.

This requires the following capabilities:

- a high repetition rate, a high problem detection rate,
- Setting parameters for process improvements,
- Means of verifying the success of the elimination, ...

The problem lies in the lack of determinism, particularly in creative work. Without determinism, there can be no reparation or regression tests, only less reliable checks with a high rate of poor decisions.

2.155 Capability improvement and maturation



Capabilities, including those to reduce the fault emergence rate, are developed offline – that is, separately from the process – and implemented at longer intervals. In the process, the »centering« gets lost, i.e. new problems arise in the process. The fault emergence rate is rising sharply.

Maturation involves many small improvement attempts with success monitoring. The fault emergence rate decreases until the potential of the new capabilities has been fully exploited.

When capability improvements are incorporated in larger time steps, the fault emergence rate tends to decrease in a sawtooth pattern.

2.156 Projects, Process Models, Creativity

Maturation processes require a large number of repetitions of the same procedures. That is why projects use process models. Phase models are a classic approach; they divide projects into phases and define checks and activities at the transition between phases. The problem lies in verifying whether a change has resulted in an improvement.

Process models can be found wherever there is a commitment to continuous learning from faults, including in public administrations, schools, ... There are common features that are independent of the specific application:

- required skills, the effort involved in the maturation process,
- phase division, restrictions on creativity, ...

Just these application-independent commonalities open up interesting perspectives on how and in what direction the development of technologies, institutions and public authorities, and also education in schools is progressing.

2.157 Bibliography

References

[1] J. E. Aas and I. Sundsbo. Harnessing the human factor for design quality. *IEEE Circuits and Devices Magazine*, 11(3):24–28, 1995.

[2] Richard A. DeMillo, Richard J. Lipton, and Fred G. Sayward. Hints on test data selection: Help for the practicing programmer. 11(4):34–41, April 1978.

[3] M.C. Feathers. *Effektives Arbeiten mit Legacy Code: Refactoring und Testen bestehender Software.* mitp Professional. mitp, 2011.

[4] Gregory Gay, Ajitha Rajan, Matt Staats, and Michael Whalen. The effect of program and model structure on the effectiveness of mc/dc test adequacy coverage. *ACM Transactions on Software Engineering and Methodology*, 25:1–34, 07 2016.

[5] P.J. HUFFSTUTTER. Frightened by y2k bug, many go to extremes, 1999.

[6] Ludovic Pintard, Jean-Charles Fabre, Karama Kanoun, Michel Leeman, and Matthieu Roy. Fault injection in the automotive standard iso 26262: An initial approach. Oct 2017.

[7] NSTL Test Report. Microsoft Windows 2000 Professional – Comparison of the Reliability of Desktop Operating Systems. http://??

[8] Sergiy A. Vilkomir and Jonathan P. Bowen. From mc/dc to rc/dc: formalization and analysis of control-flow testing criteria. *Form. Asp. Comput.*, 18(1):42–62, March 2006.