System - Technical and organizational means for the autonomous fulfillment of a task (based on Birolini, ETH)-Generally, a system can consist of hardware, software, people (service and maintenance personnel) and logistic assistance**.**
Technical System-System where influences by people and logistics are ignored.).
Technical system and environment: The coupling between the technical system and its environment is modeled by "inputs" and "outputs" coming from "sensors" and going to "actuators":
Important non-functional requirements for embedded systems: - Real-time behavior: Deadlines must be met as required by the environment - Minimal resource consumption: Memory, energy, dimensions, weight – Cost - Dependability:
• Availability – Readiness for correct service
• Reliability – Continuity of correct service
•Safety – Absence of catastrophic consequences on users and environment
•Confidentiality – Absence of unauthorized disclosure of information
•Integrity – Absence of improper system state alterations
•Maintainability – Ability to undergo repairs and modifications) Robustness: To deliver an acceptable behavior, also in exceptional situations.
Quality-Degree in which the inherent attributes of an entity fulfill quality requirements /DIN EN ISO 9000 05/.
Quality Requirement-Expectation or demand defined (by a customer) that is generally assumed or mandatory /DIN EN ISO 9000 05/.
Quality Characteristic •Property of an entity on the basis of which its quality is described and estimated, but which makes no statement about the degree of fulfillment of the characteristic •Inherent attribute of a process, product or a system that relates to a quality requirement /DIN EN ISO 9000 05/.
Quality Measure-Measure which allows to draw conclusions on the fulfillment of specific quality characteristics. For instance, MTTF (Mean Time To Failure) is a quality measure of the quality characteristic Reliability.
Influences among quality characteristics: • **Safety – (-) – Availability**: In systems with stable safe state (e.g. trains) • **Availability – (+) – Safety:** Availability of safety related functions (e.g. heart pacemaker)• **Reliability – (+) – Safety**: Reliability of safety related functions (e.g. reverse thrust) • **Reliability – (+) – Availability:** Longer time without failure Better uptime/downtime ratio • **Availability – (-) – Reliability:** System might become unreliable if maintenance is not being done properly (short repair time). • **Efficiency – (-) – Safety:** When resources that improve safety have to be left out (e.g. redundant components), so that the system is more efficient. • **Safety – (-) – Efficiency:** When safety related functions considerably consume machine resources and computation time takes long. (e.g. plausibility checks) • **Efficiency – (-) – Reliability:** The lack of resources at runtime might lead to reliability problems (e.g. exception caused by stack overflow).
Safety •Absence of unacceptable risks •State where the danger of a personal or property damage is reduced to an acceptable value •Birolini defines safety as a measure for the ability of an item to endanger neither persons, property nor the environment •A distinction is drawn between the safety of a failure-free system (accident prevention) and the technical safety of the failure afflicted system**.**
Security: Concurrent existence of a) Availability b) Confidentiality c) Integrity
Technical Safety-Measure for the ability of a failure afflicted item to endanger neither persons, property nor the environment.
Correctness •Correctness has a binary character •A fault-free realization is correct •An artifact is correct if it is consistent to its specification •If no specification exists for an artifact, correctness is not defined.
Robustness •Property to deliver an acceptable behavior also in exceptional situations •A correct system •Accordingly, robustness is rather a property of the specification than of the implementation •Robustness has a gradual character.
Reliability •Part of the quality with regard to the behavior of an entity during or after given time periods with given working conditions.
• Measure for the ability of an item to remain functional, expressed by the probability that the required function is executed failure-free under given working conditions during a given time period. **MTBF**
Availability-Measure for the ability of an item to be functional at a given time. **(MTBF/ (MTBF + MTTR))**
Failure-Inconsistent behavior w.r.t. specified behavior while running a system→ Each failure has a timestamp.
Fault-**defect**: Statically existent cause of a failure, (i.e., a „bug"). Usually the consequence of an error made by the programmer.

Chapter 3: Situation Analysis of Software Development in Practice

Design methods - Still widespread use of informal methods (text) -High interest in semi-formal methods (in particular UML)-Minor use of formal methods.
Quality management: Trend towards the certification of quality management processes (ISO 9001)-Stage of capability maturity model-based assessment methods (e.g. SPICE).
Categories Quality assurance methods:-
Informal methods are frequently applied (testing, review techniques). Formal methods (proofs) often fail concerning the complexity of the software and the properties of modern programming languages. Stochastic methods are not widespread, but are increasingly required in critical application areas in particular.
1-Informal Methods: Methods based on plausibility which produce incomplete results (Testing, Inspection and review)-
2-Stochastic Methods: Methods which produce statistically reliable, quantified results (Stochastic reliability analysis)-
3-Formal Methods: Methods which produce formally complete results on the basis of formal specifications (Formal verification techniques (Proofs)).
Quality Assurance Methods: Prognosis
- Systematic informal methods are widely used and are obligatory for many application areas where they are required by appropriate standards (Function-oriented test planning, Tool supported structural testing) - Test support is essential (e.g. regression tests)•Static analyses are additionally used.(Inspections in early phases, Tool supported analyses of code in addition to the analyses performed by the compiler.).
Situation Analysis: General Consequences
Mature Processes (are necessary, but barely offer a differentiation of competitors-operate confidence building, but provide no further statements)
•**Design methods** (Informal methods are simple and universal, but often insufficient- Semi formal methods allow the description of extensive software, but not the description of critical properties of technical software (e.g. safety)- Formal methods are powerful, but are often too specific)•Quality assurance methods (Informal methods are indispensable, but produce no sufficient completeness (testing, inspection methods)- Formal methods (proofs) provide to some degree complete results, but often fail due to preconditions, that are not fulfilled- Stochastic methods generate well-defined, reliable results, but require mathematical knowledge which is often not given in practice respectively tools which are not available on the market.
Test Phases:
•Module/Unit test: (Testing of the modules -Testing the correct function of a module w.r.t. the module specification.)
•Integration test (Testing of the interaction of the modules-Incremental assembly of the modules building the integrated system. Testing of their correct interaction.)
•System-/Acceptance test (Testing of the functionality and efficiency of a software with regard to the requirements determined in the definition phase.)•
Benefit of testing in different phases is the reduction of the respective complexity to a reasonable level.
Test Phases diagram (OOP Depiction) (Planning, Analysis, Design, Implementation || Module test, Integration Test, System Test, Field use).
Classification of the QA Methods:
(Formal verification, Symbolic testing, Dynamic Testing, Static analysis.)
Dynamic Test:
•Properties of dynamic testing
-> The executable program is provided with concrete input values and is executed
-> Program may be tested in the real environment-Never complete (it is not possible to test all possible inputs)-Correctness of the tested program cannot be proven.)
Pros:
-> widely-used.
-> Often unsystematically applied.
Cons:
-> Tests often not reproducible.-Diffuse activity (management difficulties).
Static Analysis
Pros
-> No program execution is required.
-> No input values are selected.
-> Some static analyses can detect faults directly.
Cons:
-> The static analysis concentrates on particular partial aspects.
->It is no proof of correctness.)•
Sub-categories
->Measurement (Metrics)
-> Generation of diagrams and tables
-> Data flow anomaly analysis
-> Testing of programming conventions-Inspection and review techniques.
Formal Verification:
-> Formal verification uses mathematical techniques to prove the consistency between specification and implementation.
-> A formal specification is necessary.
Pros:
- Verification may be almost completely automated (exception: e.g. finding loop invariants).
Cons:
- Requires preconditions which are often not fulfilled in practice.

Chap 4: Dynamic Testing

Goal of dynamic testing
Creating test cases that are:
- Representative
- Fault sensitive
- Distinct from each other (minimal redundancy)
- Economic.
Structural Testing: Evaluation of the adequacy and completeness of the test cases on the basis of the software structure. Determination of the correctness of the outputs based on the specification.
Benefit: Code structure is considered
Disadvantage: Forgotten (not implemented), but specified functions cannot be detected)
•Two approaches
• Control flow testing
• Data flow testing).
Control flow testing:
Statement Coverage:
- C0-test
• The goal of the statement coverage is to execute each statement at least once, i.e., the execution of all nodes of the control flow graph.
Branch Coverage:
• Branch coverage aims at executing all branches of the program to be tested. This requires the execution of all edges of the control flow graph.
- C1-test
Condition Coverage: Composite decisions are tested from left to right. The evaluation of decisions stops when its logical value is known. This is referred to as incomplete evaluation of decisions.
Simple Condition Coverage: •The simple condition coverage demands the test of all simple conditions concerning true and false
•Benefits: simple, low test costs
•Disadvantages •Limited performance •In general (concerning the complete evaluation of decisions) it cannot be guaranteed that the simple condition coverage subsumes the branch coverage.
- If decisions are evaluated incompletely, the simple condition coverage subsumes branch coverage.
Condition/Decision Coverage:
•The condition/decision coverage guarantees a complete branch coverage in addition to a simple condition coverage
•Benefits: simple, low test costs, branch coverage is ensured
•Disadvantages.(Limited performance - Structure of decisions is not really considered).
Minimal Multiple Condition Coverage:•The minimal multiple condition coverage test demands that besides the simple conditions and the decision also all composite conditions are tested against true and
Modified condition/decision coverage: •The modified condition/decision coverage requires test cases which demonstrate that every condition can influence the logical value of the overall decision independently of the other conditions
• The application of this method is required by the standard RTCA DO-178C for flight critical software (level A). Basically the method aims at a test as extensive as possible with justifiable test costs
• The relation between the number of conditions and the required test cases is linear •For the test of a decision with n conditions at least n+1test cases are required. The maximum number of test cases is 2n
• A complete modified condition/decision coverage causes a branch coverage on the object code level
• But not every branch coverage test on the object code level causes a complete modified condition/decision coverage.
Multiple Condition Coverage:•The multiple condition coverage requires the test of all value combinations of the conditions •
Benefits.(Very extensive test - Subsumes the branch coverage test and all other condition coverage test techniques)
•*Disadvantage*.(High test costs - Sometimes there exists no test data for certain combinations (e.g., because of incomplete evaluation of decisions or dependences between conditions)
Path Coverage:
Structured path test distinguished only paths that execute loop not **more than k times.** This avoids explosion of the number of paths caused by loops
• The structured path test with k=2 is called boundary interior coverage •The boundary interior coverage differentiates the three cases no loop execution, one loop execution and at least two loop executions
Data Flow Testing:
-> Def use
-> Predicate use (p-use)
-> Computation use (c-use)
•All c-uses / some p-uses-test resp. all p-uses / some c-uses-test
•If no c-uses resp. p-uses exist for some variable definitions, it is required that at least one p-use resp. c-use is tested •All uses-test •All c-uses-test + All p-uses-test.
Functional Test:
•Determination of the adequacy and the completeness of the test cases as well as derivation of the test data and evaluation of the outputs based on the specification
Benefits
- Completeness w.r.t. the specification is checked.
- The test cases are systematically drawn from the specification
Disadvantage:
- Information represented by the code is discarded.
Equivalence Partitioning:
- The test cases for valid equivalence classes are generated by the selection of test data from as many valid equivalence classes as possible.
- The test cases of invalid equivalence classes are generated by choice of test data from 1 invalid class with combination of the rest of data extracted from valid equivalence classes.
State-based Testing:
Pros:
+ State-based tests can be used in unit and system testing.
+ It has widespread use particularly in technical applications such as industry automation, avionics, or the automotive industry.
Cons:
-In state charts of large systems, there tends to be an explosion in the number of states, which leads to a considerable increase in transitions.
Transaction Flow Testing:
Pros:
+ A good basis for generating test cases.It directly specifies possible test cases.
Cons:
-Sequence diagrams display only one out of many different options.
Decision Tables or Decision Trees:
Pros:
+ They guarantee a certain test-completeness by way of their methodical approach.
Cons:
- The size of this representation increases exponentially with the number of conditions.
Diversified Test:
• Test of several software versions against each other
Back to Back Test
- Implementation of 2, 3, or even more versions by independent programmers based on the same specifications
- Evaluation of the outputs by automated comparison
• The Back to Back Test requires the multiple realization of software modules based on identical specifications
• The Back to Back Test is economically applicable, if outstanding safety and/or reliability requirements exist or an automatic evaluation of the outputs is desired or required.
- Benefit: test execution (incl. checking of outputs) can be done automatically (saves time and money)
- Disadvantages: Multiple implementation is required. Faults occurring in all versions are not detected.
Mutations Test
In fact no test method but a possibility to evaluate the efficiency (error detection rate) of test methods.
Regression Test
Test of the present version against the previous version in order to identify undesired changes of the behavior.

Chapter 5: OOT

Test of Individual Operations:
- Often have very simple control structure
- Highly dependent on attributes of object
- Have interdependencies
- Only in exceptional cases can be tested alone
Class Test:
• Usually, operations have to be tested in the context of their class
• The operations of an object of a class interact via shared attributes
• The values of the attributes define the present state of the object.
→state machines are appropriate means for specification
Hierarchy of Completeness Criteria:
all events ≥ all transitions ≥ all states
Inheritance at the Service Provider:
Call Interface:

Service Providing Operation	Call Interface	Action
Inherited	-	Repeat
Overwriting	Identical	Repeat
Overwriting	More specific	New Assertion; Repeat
Overwriting	More General	Repeat

Return Interface:

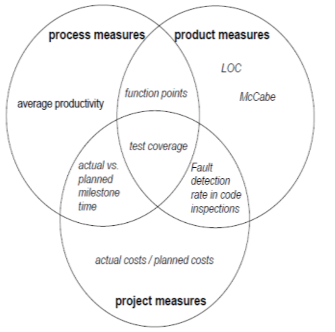
Service Providing Operation	Return Interface	Action
Inherited	-	Repeat
Overwriting	Identical	Repeat
Overwriting	More specific	Repeat
Overwriting	More General	Repeat; Generate additional test cases

Inheritance at Service User:

Service Using Operation	Call Interface	Action
Inherited	-	Repeat
Overwriting	Identical	Repeat
Overwriting	More specific	Repeat
Overwriting	More General	Repeat; Generate additional test cases

Chap 6: Measurements

- Measures can only indirectly point to potential sources of problems. A significant deviation of a measure from its usual value might be an indicator for a problem, but this is not guaranteed.
Application of Measures:
• Control of software quality
• Control of software complexity
• Control of the software development process
• Costs and time prediction
• Costs and time tracing
• Definition of standards
• Early problem identification
• Comparison and evaluation of products
• Feedback concerning the introduction of new methods, techniques, and tools



Measures Requirements:
• ***Simplicity*** (Is the result so simple that it can be interpreted easily?)
• ***Suitability*** (Does the measure show an appropriate correlation to the desired property?)
• ***Robustness*** (Is the value of the measure stable with regard to minor changes of the measured software?)
• ***Timeliness*** (Is the measure available early enough?)
• ***Processability*** (Is it possible to process the measures (e.g. statistically -> scale type)?)
• ***Reproducibility*** (The value of a measure should have an identical value for a particular product independently of the mode of generation)

Measure Scales:
1-Nominal scale
Free description of certain properties with labels-Inventory numbers of books of a library (DV 302, PH 002, CH 056,
2-Ordinal scale
- Mapping of an ordered aspect of a property to an ordered number of measurements in such a way that the order is maintained
- Mapping of the arrival of patients to the waiting numbers in a doctor's surgery
3-Interval scale
- A scale which is still valid if transformations $g(x) = ax + b$, with $a > 0$ are applied to it
- Temperature scales in degree Celsius or Fahrenheit.
4-Rational scale
- A scale where measurements can be correlated (statements like double, half, three times as much, ... make sense)
- Length in meters (it is twice as far from a to b as from c to d)-Temperature in Kelvin)
5-Absolute scale
- A scale which is the only possibility to measure the issue (-Counting-Probabilities)).

- A relation $\bullet\geq$ on a set A is called *order* if
  - a) $\forall x, y, z \in A: x \bullet\geq y \wedge y \bullet\geq z \Rightarrow x \bullet\geq z$ (transitivity)
  - b) $\forall x \in A, \exists y \in A: x \bullet\geq y$ or $y \bullet\geq x$ (comparability)
  - Example: the relation "is ancestor of" on the set of persons
- An order is called *quasi order* if
  - c) $\forall x \in A: x \bullet\geq x$ (reflexivity)
  - c implies b: every x is at least comparable to itself
  - Quasi orders can contain elements which cannot be ordered
  - Example: the identity "=" on every not empty set
- A *quasi order* is called *half order* if
  - d) $x \bullet\geq y \wedge y \bullet\geq x \Rightarrow x = y$ (anti-symmetry)
  - Half orders also can contain elements which cannot be ordered
  - Example: the relation „≥" on the set of integers

Measurement axioms:
- A half order is called *linear* if
  - e) $\forall x, y \in A: x \bullet\geq y$ or $y \bullet\geq x$ (connectivity, completeness)
  - Example: the relation "≥" on the set of integers
- Orders which fulfill the axioms a, c (and thus also b) and e, but not necessarily d, are called *weak order*
- In the following the empirical relation $\bullet\geq$ is considered. It is demanded that it generates a weak order on the set of the modules A, fulfilling the following axioms
  - axiom 1: reflexivity: $a \bullet\geq a, \forall a \in A$
  - axiom 2: transitivity: $a \bullet\geq b, b \bullet\geq c \Rightarrow a \bullet\geq c, \forall a, b, c \in A$
    (If the complexity of module a is greater equal the complexity of module b and the complexity of b is greater equal the complexity of c also the complexity of a is greater equal the complexity of c.)
  - axiom 3: connectivity (completeness): $a \bullet> b$ or $b \bullet> a, \forall a, b \in A$
- A rational scale has to meet all criteria of an ordinal scale. The empirical and formal relational system has to be enhanced as follows
  - $((A, \bullet>, \circ), (\mathbb{R}, >, +), f)$,
  - with $a \bullet\geq b \Leftrightarrow f(a) \geq f(b)$, (ordinal scale)
  - and $f(a \circ b) = f(a) + f(b)$, (rational scale)
  - $\forall a, b \in A$
- $\circ$ is a binary operation for the empirical relational system. + is the corresponding binary operation for the formal relational system
- A measure $f: A \rightarrow \mathbb{R}$ which meets the requirements of the rational scale mentioned above exists when
  - $(A, \bullet\geq)$ fulfills the axioms 1, 2, 3 (reflexivity, transitivity, connectivity)
  - axiom 4: $a \circ (b \circ c) \bullet\approx (a \circ b) \circ c, \forall a, b, c \in A$ (associativity)
  - axiom 5: $a \bullet\geq b \Leftrightarrow a \circ c \bullet\geq b \circ c \Leftrightarrow c \circ a \bullet\geq c \circ b, \forall a, b, c \in A$ (Monotony)
  - axiom 6: if $c \bullet> d$, it is valid: $\forall a, b \in A, \exists n \in \mathbb{N}, a \circ nc \bullet\geq b \circ nd$ (archimedic axiom)

Halstead Measure
•Set of measures concerning different aspects, e.g., complexity, size, costs, etc.
•Are all based on theoretical considerations
•Are based on the program text (number of the different operands and operators and total number of the operands and operators)
•Halstead's costs measure E does not necessarily fulfill the criterion of timeliness
•No direct relation to natural parameters; unnatural measures
•Common as measures in analysis and test tools
•Remark: Halstead's costs measure determines a quadratic dependence between the size of a module and the costs for its implementation modularization

The four basic parameters of the Halstead measures are
•n1 – number of different operators
•n2 – number of different operands
•N1 – total number of operators
•N2 – total number of operands
•From these four measures two further simple measures can be derived
•n = n1 + n2 – size of the vocabulary
•N = N1 + N2 – length of the implementation
•By considering some combinatorial rules the formula for the calculated program length N is derived
•N = n1 log2 n1 + n2 log2 n2
•Program volume V
V = N log2 n
•V is the volume of the program in bits provided that a binary coding with a fixed word length of the vocabulary is used

•The potential program volume V* depends only on the algorithm, not on the programming language used for the implementation

V*= (N1*+N2*) log2 (n1* + n2*) = (2 + n2*) log2 (2 + n2*) Every implementation has a level L which is smaller or at best equal one. The more L approximates the value one, the more appropriate is a programming language for the implementation of a given algorithm
•A measure for the difficulty to implement an algorithm in a programming language is the reciprocal D of the level (Difficulty)
D = 1 /L

The volumes L and D are a measure for the problem adequacy of the used programming language and for the difficulty to implement a given algorithm in a particular language

The Effort E necessary to code an algorithm is proportional to the program volume and to the difficulty of the coding. Difficulty D is the reciprocal of the program level L

Effort E then can be defined to

$$E = \frac{V}{L} = \frac{V^2}{V^*}$$

- Software measurement is, e.g., important for the following areas
• Flat management structures
• Compliancy to certain software engineering standards
• High capability maturity levels.

Software Engineering Standards:

Benefits:
• Proof of qualification for potential clients
• Marketing criterion; differentiation from non-accredited competitors
• Important in the context of product liability
• In some areas definitely required
• All standards underline the importance of a systematic procedure, transparency, and control of the development process.

Capability Maturity Model:

1-initial, 2-repeatable, 3-defined, 4-measured, 5-optimizing
• The attainment of the maturity levels 4 and 5 is possible only with the existence and use of a measuring system which enables the following operations
- Measuring of productivity and quality
- Evaluation of projects on the basis of these measuring
- Identification of deviations
- Corrective actions in the case of deviations
- Identification and control of project risks.

Chap 7: Data Flow Anomalies

Three Types:
1. A value must not be assigned twice to a variable (dd-anomaly)
2. An undefined variable must not be references (ur-anomaly)
3. The value of a variable must not be deleted directly after the value has been assigned (du-anomaly)

Chap 8: Slicing

Characteristics and Goals:

Backward Slicing: A program slice in this context describes which instruction affects an observed value in which way. A slice will always be given in relation to an observed value at a certain position of the program. In a strict sense, this is the definition of so-called backward slicing.
Forward Slicing: There is also forward slicing. Such a slicing shows which instruction is affected by an observed value in which way.

Chap 9: Reviews

Manual quality assurance in three variants.
• Review through sending documents to the review team members (•Fast, cheap, flexible, low performance)
• Structured walkthrough (•Medium use of resources and moderate performance)
• Fagan inspection (•Expensive and time consuming, but efficient and effective.)

Software inspection
• Manual quality control of a product
• Small group of participants with defined roles
• Aims at the detection of faults, not at finding the solutions
• Requires a functioning development process
• Executed as a formal process
• Input and output criteria
• Defined inspection phases
• Skilled participants
• Collection and analysis of inspection data including feedback to the inspection process
• Fault documentation
• Objectives for the results (e.g. Fault detection rates, inspection rate)

Reviews
• No formal inspection.
• Normally no formal procedure exists for the execution and the choice of the participants as well as their roles
• Often no record and analysis of review data
• Often no quantitative objectives.

Also means for:
- Decision making
- Solving of conflicts
- Exchange of information
- Brainstorming

The main differences between reviews respectively walkthroughs and formal software inspections are:
•Inspections have the sole aim to detect faults efficiently and effectively •Inspections are done as a defined process.

Why Software Inspections?
• Many quality characteristics –e.g. understandability, changeability, informational value of identifiers and comments –are testable only manually
• Undetected faults from the definition and design phase later cause high consequential costs
• As inspections are executed in a team, the knowledge base is enhanced
• Implementation of the principle of external quality contro
• Delivery of high-quality results to the subsequent software development phase (milestone)
• Responsibility for the quality is assigned to the whole team.
• Critical components are detected early.
• As several persons inspect the products, the authors try to use an understandable style.

Requirements for Inspections:
• The required time has to be scheduled → project planning
• The participants have to be skilled w.r.t. inspections
• The procedure of the inspections has to be written down and their observance has to be controlled
• The project has to be done well-structured and controlled
• There has to be a quality management process with defined quality objectives
• The results of inspections must not be used in personnel evaluation.

Inspection Team: (Peer to Peer Technique)
• Moderator: accepted specialist with special training.
• Author (editor): responsible for correction of faults
• Reader: leads the inspection team.
• Recorder: notes and classifies all faults
• Inspectors: all member of inspection team are instpectors.
• Size of the review team: 3 to 7 members.
minimal no of participants is 3(moder/reco,read, auth)

Inspection Phases:
• Planning: Organizational preparation
• Overview: The author informs (2-3 Hours, 500 NLoC per hour without comments)
• Preparation: Every inspector prepares (125 NLoC per hour)

Ariana5 Failure: break-down of flight controller resulting a mechanical destruction of rocket **Fault:** The integer conversion of the 64-bit floating point variable into a signed integer caused a data overflow **Error** blind reuse of software components of Ariane4, which have not been tested sufficiently within the new environment of Ariane5

• **Inspection meeting** (2-3 Hours, 90 NLoC/Hour to 125 NLoC/Hour, *Accepted, Conditionally accepted, Reinspection Required*)
• **Rework**: Fault correction
• **Follow-up**: Inspection of the fault corrections. (If conditionally accepted → Verification can be done by author and reader || if reinspection → Conventional inspection meeting)

TRUE	FALSE
Correctness has a binary character	A **correct** system is always **safe**
An **artifact** is not consistent to its spec, if it's not correct	**Correctness** is always defined
System is effected by human	**Correct** software always guarantees a **safe** system
The environment and people can be part of a system	Can a **reliable system** be incorrect?
If a system is **reliable** then it is available	It can always be decided whether an **artifact** is correct/not
A **correct** system can have low **robustness**	An **available** system is (always) **safe**
Robustness is depending on the specification	When analyzing a system, people r never taken into a/c-
Safety allows the presence of risk.	A system with **correct** specification is always **available**
If there are no defects the program is **correct**	A system can have a low robustness even if its **correct**
A system with a low technical **safety** can be of a tech sys-	**Robustness** has a binary character(gradual char)
High **efficiency** achieves high **safety**	**Robustness** is a property only of the implementation
Safety can be measured	A system that is **robust** is always **safe**
every / Each **Fault** leads (always) to a **Failure**	A **safe** system is always **available**
Every **failure** should have a timestamp	**Hazard** always leads to Accident. (may or may not)
An error can cause a **failure** lead-	High **availability** always leads to high **reliability**
errors can lead to **faults**	A **technical system** cannot influence the environment/people
every **failure** is caused by a **fault**	Technical **safety** is defined for **technical systems** only
MTTR is a **reliability** measure	A / Every **failure** leads (always) to a **Fault**
High **reliability** always leads to high **availability**	a **failure** is always the cause of a **fault**
Equipment may be **available** but not **reliable**	**Robust system** is always safe.
Low **Robustness** don't handle failure condition.	**Safe system** is always **reliable**.
Robust system need not be correct.	**Safety** is absence of risk. (acceptable risk)
Correctness is the property of code.	Environment can influence **system safety**
System with low technical safety can be a **technical system**	A **safe system** can suffer from security breach.

Simple Condition Coverage: test of all simple conditions concerning true and false.
Condition/Decision Coverage: test of all simple conditions + overall decision concerning true and false
Minimal Multiple Condition Coverage: all simple conditions + overall decision + composite conditions concerning true and false.
Modified condition/decision coverage: requires test cases which demonstrate that every condition can influence the logical value of the overall decision independently of the other conditions
Multiple Condtn Coverage: test of all value combinations of the conditions

(A && B) || (C && D)

		A	B	C	D	A&B	C&D	||
1	1,2,5,6	F	-	F	-	F	F	F
2	3,7	F	-	T	F	F	F	F
3	4,8	F	-	T	T	F	T	T
4	9,10	T	F	F	-	F	F	F
5	11	T	F	T	F	F	F	F
6	12	T	F	T	T	F	T	T
7	13,14,15,16	T	T	-	-	T	-	T
Simple Condition		Com: (1, 16):Incom: (2,3,4,7)						
con/Dec coverg		Com: (1, 16):						
Minimal Multiple		Com:(1, 16) Incom: (2,3,5,7)						
Modified Condition		Com: (6,7,8,10,14)Incom(1,2,3,4,7)						

(A | | B) && (C | | D)

		A	B	C	D	A||B	C||D	&&
1	1,2,3,4	F	F	-	-	F	-	F
2	5	F	T	F	F	T	F	F
3	6	F	T	F	T	T	T	T
4	7,8	F	T	T	-	T	T	T
5	9,13	T	-	F	F	T	F	F
6	10,14	T	-	F	T	T	T	T
7	11,12,15,16	T	-	T	-	T	T	T
Simple Condition		Com: (1, 16):Incom: (1,2,3,7)						
con/Dec coverg		Com:(5,12)						
Minimal Multiple		Com:(1, 16) Incom: (1,2,6,7)						
Modified Condition		Com:(2,6,9,10,11) Incom(1,2,3,4,7)						

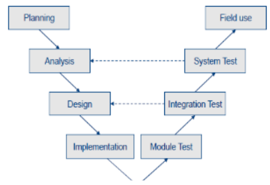
(A && B) | | (C | | D)

		A	B	C	D	A||B	C||D	||
1	1,5	F	-	F	F	F	F	F
2	2,6	F	-	F	T	F	T	T
3	3,4,7,8	F	-	T	-	F	T	T
4	9	T	F	F	F	F	F	F
5	10	T	F	F	T	F	T	T
6	11,12	T	F	T	-	F	T	T
7	13,14,15,16	T	T	-	-	T	-	T
Simple Condition		Com: (1, 16):Incom: (5,7,6,1)						
con/Dec coverg		Com: (1, 16):						
Minimal Multiple		Com:(1, 16) Incom: (5,7,6,1)						
Modified Condition		Com:(5,9,10,11,13) Incom(1,4,5,6,7)						

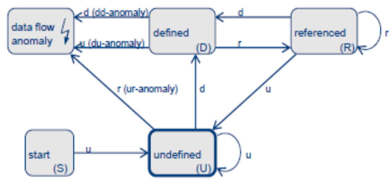
(A && B) && (C | | D)

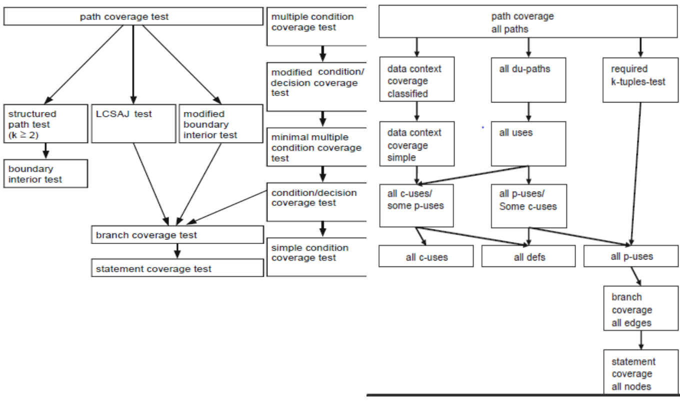
		A	B	C	D	A&&B	C||D	&&
1	1,2…..8	F	-	-	-	F	-	F
2	9,10,11,12	T	F	-	-	F	-	F
3	13	T	T	F	F	T	F	F
4	14	T	T	F	T	T	T	T
5	15,16	T	T	T	-	T	T	T
Simple Condition		Com: (1,16):Incom: (1,2,3,4,5)						
con/Dec coverg		Com:(1, 16) Incom: (1,2,3,4,5)						
Minimal Multiple		Com:(1, 16) Incom: (1,2,3,4,5)						
Modified Condition		Com:(9,6,13,14,15) Incom(1,2,3,4,5)						

Test Phases:



State machine for data flow anomaly:



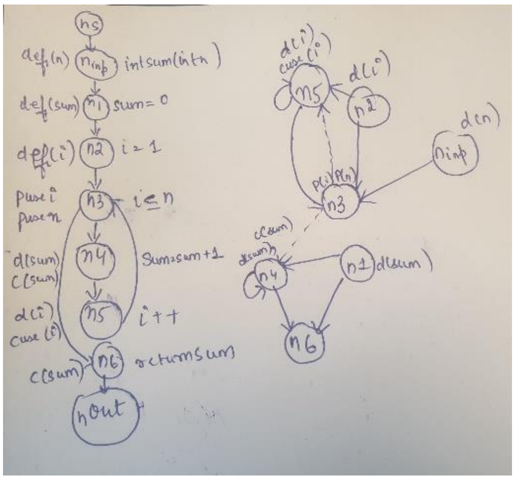


•The volumes L and Dare a measure for the problem adequacy of the used programming language and for the difficulty to implement a given algorithm in a particular language

•The Effort E necessary to code an algorithm is proportional to the program volume and to the difficulty of the coding. Difficulty D is the reciprocal of the program level L

•Effort E then can be defined to

E = V /L = V*V / V*



	A	B	C	D
1	F	F	F	F
2	F	F	F	T
3	F	F	T	F
4	F	F	T	T
5	F	T	F	F
6	F	T	F	T
7	F	T	T	F
8	F	T	T	T
9	T	F	F	F
10	T	F	F	T
11	T	F	T	F
12	T	F	T	T
13	T	T	F	F
14	T	T	F	T
15	T	T	T	F
16	T	T	T	T

Scale type	Operations	Transformations	Example
nominal	=, ≠	all unique	gender, job, ...
ordinal	=, ≠, <, >	strictly monotonic	marks, 1,2,3,4,5,6
interval	=, ≠, <, >, +, -	linear, y=ax+b	Temp in °C
rational	=, ≠, <, >, +, -, *, :	linear, y=ax	Distance in m
absolute	=, ≠, <, >, +, -, *, :	none	quantity