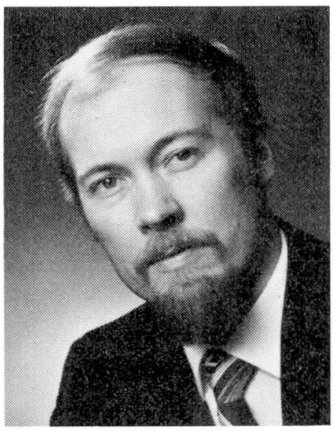
Software quality assurance

Software Quality Assurance

Assurance de qualité du logiciel

Qualitätssicherung in der Entwicklung von Software

Manfred KERSKEN
Electrical Engineer
Gesellschaft für Reaktorsicherheit
Garching, Fed. Rep. of Germany



Manfred Kersken, born 1944, got his Dipl. Ing. in Electrical Engineering from the Technical University of Berlin. He worked for the Lehrstuhl für Reaktordynamik und Reaktorsicherheit of the Technical University, München, afterwards he joined GRS, Garching. He is involved in the analysis and qualification of computer systems in safety related areas such as nuclear power plants and railway transportation systems.

SUMMARY
A short survey is given on the methods applied to avoid errors as far as possible during the construction of software. Likewise the fundamental analytical possibilities to remove remaining errors in programs are outlined.

RESUME
L'article présente les méthodes utilisées pour la prévention de fautes lors du développement du logiciel et sur les possibilités analytiques d'élimination de fautes restantes.

ZUSAMMENFASSUNG
Ein kurzer Überblick wird gegeben über Methoden, die verwendet werden, um Fehler während der Erstellung von Software weitgehend zu vermeiden sowie über die grundsätzlichen analytischen Möglichkeiten, um verbleibende Programmfehler zu entfernen.

1. NECESSITIY FOR QA IN SOFTWARE

The facts

- While the costs for hardware are decreasing, the costs for software are steadily increasing (Fig. 2).
- Up to 50 percent of the costs for software in its life cycle (Fig. 1) originate from verification and validation (V&V).
- In spite of V&V, a lot of errors which are implemented in all phases of software development, remain in the programs which are delivered to the customer.
- Specified properties of software, like reliability, effectiveness, maintainability, testability, readability, are insufficiently fulfilled.

The remedies

- Structuring the process of software development to prevent software errors (constructive approach).
- Structuring the process of software V&V to remove software errors (analytic approach) effectively.

2. CONSTRUCTIVE APPROACH

- Top-Down Development: The phases "Requirements", "Design" and "Coding" are developed from general towards specific aspects (Fig. 1), to avoid integration problems when developing a system bottom-up, from the component to system level.
- Levels of Refinement: Top-down development results in hierarchical levels within each of the development phases (Fig. 1). The steps between successive levels of refinement should be small to avoid errors in the transfer of a level to the next lower one. Each level of refinement must describe the whole system and must be fully documented.
- Development Tools: As far as possible computer assistance is involved in the development process. There are existing software systems which enforce a structured development of requirements-, design-, coding- and maintenance phase.
- Choice of Suitable Languages: Dependent on the problem to be solved, a suitable language is chosen, e.g. FORTRAN is good for numeric problems, ALGOL supports a good structure and the readability of programs, ATLAS has powerful features to implement test devices,...
- Restriction of Language Features: Theoretical work has shown that every problem (a computer can solve) can be solved with the three language constructs "Sequence", "Conditional Branch" and "Loop" (Fig. 3). With these constructs well structured programs can be written; however, every language contains additional constructs, some of which may contradict to the envisaged goal. E.g. a jump backward into a loop (see Fig. 3, dotted line) is a construct which is harmful for readability, reliability.

3. ANALYTICAL APPROACH

- Verification & Validation (V&V): The development of software is accompanied by V&V activities (Fig. 1). Verification is the process of demonstrating that all properties of a level of refinement have been translated in a correct manner to the next level of refinement. Validation comprises several levels of refinement, e.g. the coded software (a program written in a programming language) is compared against its specified requirements; it is shown whether the code fulfills the requirements or not.

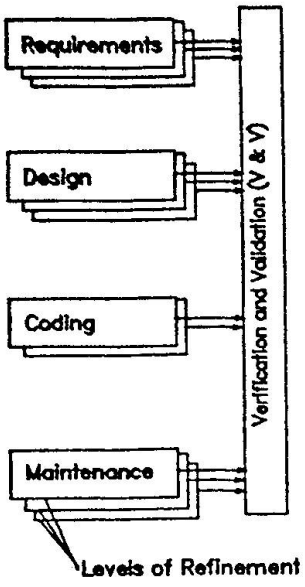


Fig. 1: Software Life Cycle

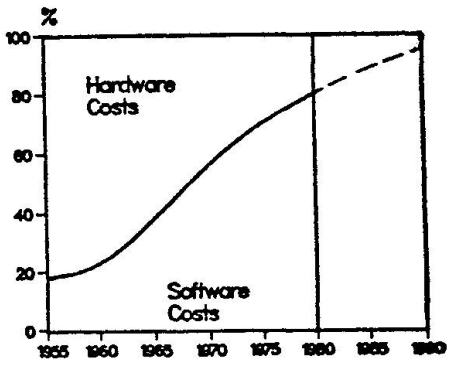


Fig. 2: Costs for Computer Systems

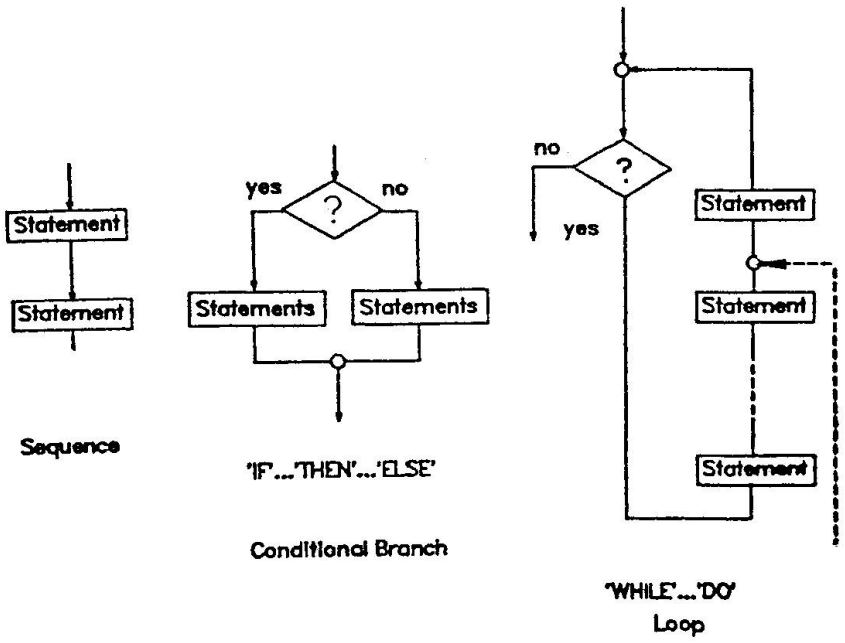


Fig. 3: Language Constructs

- Methods of V&V: These are static and dynamic analysis, testing, symbolic evaluation and proof of correctness.
- Proof of Correctness: Proving mathematical theorems about a program given its intended behaviour in the form of a set of assertions. These proofs are very difficult to realize, up to now no program of practical size could be proven with this method.
- Symbolic Execution: The program is executed as a sequence of symbolic formulas (with symbolic input data instead of numeric values). In the course of this method a set of symbolic expressions is produced, the solving of which may be very difficult - in a variety of situations even impossible.
- Program Analysis: The structure of a program is worked out, i.e. the control flow (all possible successions of statements which depend on jumps and branches) and the data flow within the program. With this knowledge the mappings of input data to output data by means of the program are demonstrated. All properties of the program are representable by these mappings.
- Testing: The program is executed with a predefined set of input data and it is observed whether the appropriate set of output data is produced. Difficulties arise in the determination of this set of output data; often it is determined from the requirements or by simulation. Testing is mostly preceded by program analysis to embed program structure in the tests.
- Verification Tools: As in development there exists a variety of software systems which are helpful in structuring the V&V procedure.
- Software Reliability Models: There are a variety of models which make the attempt to describe the activities of programming and removing of errors in a mathematical way and to give estimations on several figures of merit like "number of remaining errors after test" or "mean time to next error after test". As the above mentioned activities are very complex, the models can only be rather inaccurate as well as the resulting figures of merit.

4. CONCLUSIONS

The proof that a piece of software is free of errors cannot be given by a single of the constructive and analytic methods. The only way to give a certain amount of confidence into the software is to apply a meaningful combination of methods which are "taylored" to the software to be examined.