Validation of software for railway applications

B. Dejneka

Federal Research and Test Centre Arsenal, Faradaygasse 3, A-1030 Vienna, Austria

Abstract

Software in computer systems with safety responsibility (also in signalling and train control systems) has to be "safe software". It could be achieved by using different measures, for example measures to avoid faults in the design process of software (i.e. safety requirements) or to discover faults in the software (i.e. safety validation of software). This paper gives an overview about safety requirements and validation methods for software. They are discussed based on a case study with CASE-tools. Also, an economical aspect of the CASE-tools applications is presented.

1 Introduction

The complexity of control tasks in many systems has nowadays grown up so much, that realization of them without computer systems is hardly possible. Also the computer controlled systems with safety responsibility are applied today in almost all fields of modern technics also in traffic (e.g. in signalling and train control systems in the railway).

The author's institution, the Department "System Reliability and Traffic Electronics" of the Federal Research and Test Centre (BFPZ) ARSENAL in Vienna works already since a long time in reliability and safety validation of control electronic equipments and systems mainly for signalling applications in the railway traffic. During this work software is very often validated.

Besides the routine proving works the activities of the authors institution also contain applied research works. The fundamental and functional system considerations regarding the signalling safety on the field of the rail traffic have been done (Dejneka[1]). The signalling safety is understood here as correct, i.e. errorless function of e.g. signals and switches in the railway technology.

The relevant equipments with the signalling safety, e.g. warning signal equipments, interlockings, automatic train control systems, etc., include very of-
ten transmission or telecommunication parts. Therefore the transmission safety is also an important part of signalling safety and an object of validation work of the Department System Reliability and Traffic Electronics (Sethy[2]) too.

2 Safety requirements for the software

"Safety" in general is a state in which no danger exists for persons or for goods. In this sense safety related software is a software that ensures that a system does not endanger human life, limb and health, economics or environment. Such software must fulfill higher correctness expectations, with other words it shall have minimal software rest error ratio (expressed by software metrics).

Safety related software should be of a high quality to ensure reliable operation throughout its life. This high quality shall be arrived by fulfilling safety requirements. The requirements concern a big group of software attributes and can be split under different points of view, e.g. in the following way:
- development process of software (i.e. planning, documentation and verification of each phase),
- requirements for the software as a consequence of requirements for the complete computer system,
- structure of software,
- coding,
- protective measures against failures,
- documentation,
- software validation (with application of the methods and tools, they increased the quotient of detected errors),
- other aspects.

2.1 Requirements for the development process

The production of high quality software requires the strict control of the development process and its documentation. The necessary phases of this process will vary according to the system, but, typically, the following phases will be carried out (IEC[3]): system requirements, system design, module design, coding, testing and software/hardware integration.

Each phase shall be divided into elementary tasks with well-defined input, output and activities. For each phase the associated documentation shall be produced like design and test specification, verification plan and report, etc. In each development phase it shall be shown that the functional, reliability, performance and safety requirements are met. Therefore these phases must be ended by a verification, which should be carried out by an independent party.

For all the verification activities the verification plan shall be established, concurrently with the development and for each phase. It shall document all the criteria, techniques and tools to be used in the verification process for that phase.

After each verification activity a verification report shall be produced stating either that the software has passed the verification or the reason for the failures.

2.2 Requirements for the software as a consequence of requirements for the complete computer system.

The requirements for the complete computer system have a great influence on the safety concept of the software. For example the following requirements result from the structure of the whole system:
- It should be clear if the computer system has a redundant structure, or diversi-
yty, if the system concept is fault tolerance or fault avoidance concept (IEC[3]).
- The interfaces to other systems must be specified in a clear and simple way.
- The safety relevant part of the software shall be separated from the non critical software (ORE[5], ORE[6]).

2.3 Requirements for the structure of software
The proposed software structure shall be detailed in the software structure speci-
fication. It shall fulfill at least following requirements:
- The software shall have a modular structure with clearly defined inputs and
outputs of each module (UIC[4]).
- The safety-related part of the application shall be minimized (IEC[3], ORE[5]).
- Inside each program there shall be applied only following three structure ele-
ments and/or their combination: sequence, choice and loop (ORE[7]).

2.4 Requirements for the coding
Coding in general is understood to be a conversion of a logical sequence of
statements (showed as structure diagrams) in a code (by using of programming
language), which after compilation will be executed by the processor. At this
stage the errors that can occur would be systematic errors.

To avoid errors during the coding the following requirements shall be ful-
filled (ORE[7]):
- The listing should be divided according to the structure diagrams. The com-
ments should add information for comprehensibility of the program (not just
repeats the code statements in plain language).
- Dynamic modifications of instructions in operational programs shall not be
allowed.
- The variables must be clearly separated in output, input, output/input variable,
global and local variables.
- The addressing of variables and jumps must be clear. Complex calculating of
the conditions and addresses for branches and jumps shall be avoided.
- Constants, variables, flags, etc., must only be referenced by name.
- The meaning and use of each variable must be defined when the variable is
declared at the start of the program.
- Jumps (branches) shall only go to the beginning of a loop.
- Return from subroutines must always lead to the instruction immediately fol-
lowing the call.
- It must be assured, that used compilers or assemblers do not generate any er-
ors by converting the safety relevant programs into the machine code.
- If assembler language is being used, it is strongly recommended that indirect
jumps or branches are avoided, because they make the validation process ex-
tremely difficult, etc.

2.5 Requirements for the protective measures against failures
The protective measures against failures could help to reduce the error rate and
can be used to detect errors in an early process stage. They should be involved
in the programs itself. Following measures belong to these ones: failure detec-
tion mechanisms and times, measures against common mode failures, mecha-
nisms for detecting hardware faults or software corruption as well as measures
against using of variable data that is obsolescent or corrupt (ORE[5], ORE[6]).
It could be realized for example by diverse software, redundant bits for coding
or by mutual comparison of checksums from redundant channels, software mo-
dule shall run automatic tests in specified time interval.
2.6 Requirements for the documentation

The fulfillment of the requirements for the documentation is important for the understanding and it supports the testability of the program. For the documentation the most important requirements are the following ones (DB[8]):

- The minimum documentation for the code is the text of the equivalent step in the structure diagram.
- The documentation of each statement must be clear, redundant information shall be avoided.
- The use of each variable shall be described exactly, which procedures use these variables, and to which physical address do the variables correspond.
- The structure diagrams for each software module shall be clearly arranged, and the connection between the modules shall be obvious.
- The documentation shall correspond to the newest version of the code.

2.7 Other safety requirements

Other requirements, which do not belong to the above mentioned groups, could be the following ones:

- Requirements for the handling of interrupt: in general interrupts should be avoided (ORE[7]). If interrupts are considered to be necessary, it is preferred using only one level of interrupt and it is important to keep the use of interrupts as simple as possible. It must be ensured that no data corruptions can result from these interrupts.
- Requirements of the safe system generation and configuration after switch on or reset operation (ORE[6]).
- Requirement of possibility of software adaptation for a particular application by changes in the data base but not by changes in the program (ORE[7]).

3. Software validation

However, the fulfillment of the above mentioned requirements does not give enough guarantee, that there are no errors in the software. Therefore, each software product has to be validated. The software validation methods are widely independent from the field of application. They have already been described individually in detail several times in the literature (IEC[3], UIC[4], ORE[7], EWICS[9]).

The methods for the safety validation of software can be subdivided concerning different criteria. On the one hand they can be split according to the state of the proving object (i.e. "static" and "dynamic" methods) and on the other hand according to the validation level ("black box" and "white box") methods.

The distinctive criterions "static" or "dynamic" are derived in the literature from the state of the object to be proved in this sense, that in case of static methods the object needs not at all to be put in operation, whereas in case of dynamic methods the object will be proved especially together with its function.

In this manner a systematic of the validation methods shall be given here in a matrix structure. Following examples of validation methods belong to these main groups:

Group 1a (i.e. static "black box" methods): proving of redundancy aspects by Common Cause Failure Analysis, proving of performance by checking of Performance Requirements Lists, simulations/modeling methods (e.g. Performance Modelling, Boundary Value Analysis, Finite State Machines/State Transition Diagrams, Monte-Carlo Simulation, Cause Consequence Diagram), etc.
Group 1b (i.e. dynamic "black box" methods): functional testing (e.g. Probabilistic Testing, Interface Testing, Tests Case Execution from Boundary Value Analysis or from Cause Consequence Diagram), performance testing (e.g. Avalanche/Stress Testing, Response Timings and Memory Constraints), etc.

Group 2a (i.e. static "white box" methods): intuitive methods (e.g. Program Inspection, Program Walk-Through), semantic analyses (e.g. Symbolic Execution of Program), structural checks (e.g. Data Flow Analysis, Information Flow Analysis, Control Flow Analysis), simulation/modeling (e.g. Time Petri Nets, Markov Models, Structure Diagrams, Event Tree Analysis, Fault Tree Analysis), etc.

Group 2b (i.e. dynamic "white box" methods): data validation by testing in a simulated environment or by comparison with an already existing system, testing methods according to program structure (e.g. Structure Based Testing), Process Simulation, etc.

Which methods will be applied by the author's institution for the software validation depends on the one side on the function and structure of the software to be proved and on the other side on the agreement with the orderer (List[10]).

In the last time several software tools have been created, some of them are intended in principle for software development. Nevertheless for software safety validation it is possible to use such software tools, which have a powerful proving part. The application of such programs (for example MALPAS or SPADE) essentially decreases the rest error rate of the safety validation work, which is considered positive. However, the tools cannot replace the other validation methods without previous validation of the software tools themselves. Therefore for the safety validation a certain form of matrix structure of proving methods should be applied, which guarantees a maximum safety of proved software. It means, that the several proving methods, which can overlap partially, should be used simultaneously for software validation.

4 Case study with CASE-tools

4.1 Preparation phase
The used CASE-tools cannot interpret an assembler language. They use a special language. The source code has to be translated into the CASE-tool specific language. The translation process can be simplified by realizing a model of the processor (written in the CASE-tool specific language) that simulates the used instructions.

The input file needs some additional informations e.g. procedure specifications, main program specifications, function specifications, derived relationships, assert statements. The derived relationships simplify the analysis at a procedure call. The assert statements can be used for refining the analysis. These informations have great influence to the results of the analysis.

4.2 Analysis phase
The output of the CASE-tool depends on the specification in the command line, (which keywords were used). The CASE-tool that has been used, cover the control flow analysis, data use analysis, information flow analysis, semantic analysis and compliance analysis.

The control flow analysis is used to find out the structure of the code and to find out unreachable statements, multiple entries into loops. The control analyzer simplifies the graph. The stage of simplification (if only sequences of
nodes are removed, or also self loops) can be controlled by the used keywords.

The data use analysis shows how often a variable is read before written. Hence variables that are written more than once without reading could suggest omitted code. Also the data use analysis show if variables have been written and never read. This could indicate redundant code. As result also possible errors are stated, which have to be confirmed by the user of the CASE-tool.

The information flow analyzer delivers as result the dependency of the variables from the used variables and constants. For each variable the dependency from conditional nodes is stated. Also, a list of possible errors and redundant statements is given.

The semantic analysis generates the relation between input and output variables of each executable path. The user of the CASE-tool has to compare the results of the semantic analysis with the requirements specification.

At the compliance analysis the relations between input and output variables are calculated and compared with the specifications at the begin and the end of the program block. These specifications have to be inserted by the operator of the CASE-tool. The quality of the result from this analysis depends on these specification statements. By setting up more conditions in the code the result of the analysis can be simplified.

4.3 Evaluation phase
This is the most difficult section of the validation with CASE-tool. There the results of the different analysis methods have to be compared and conclusions must be made.

Some problems about evaluating are given in this chapter. For example the communication between the individual subroutines of the tested software is realized often by using the accumulator and flag register. In this case the CASE-tool can deliver an error statement that the register is not defined. This handover procedure was not done randomly, it was used systematically. Here a violation of the software requirements occurs. The main question here is now: Is this acceptable violation or shall it be treated as a safety critical violation? The decision whether the use of a register as variable is acceptable or not, has to be made by the proofing person and the orderer of the validation (contractor).

At a procedure the data use analyzer suggested, that a variable was written for sometimes with no intervening read. A review of the procedure showed that these writing actions to the variable were correct. The used processor model simulates the flag register by using boolean variables for each flag. According to the operation of the processor the flags have to be set. To avoid such results by the CASE-tool the variables that should be proved can be selected. This option simplifies the analysis results. It should be taken in account, that the selection of the variables is a critical decision done by the user of the CASE-tool.

The one of the software systems, which has been validated by the department of the author, has as documentation of the code only a structure diagram (Nassi-Shneiderman Diagram) and an incomplete variable list. It is clear that the requirements concerning documentation of the software were not fulfilled. The question is, shall the software be treated as a safe software or as unsafe ones. The incomplete documentation will increase the necessary time for the validation. The presentation of the validation report has been discussed by List [10].
4.4 Advantages for the use of CASE-tools

The use of CASE-tools for software validation supports a formalizing of the analysis results. This formalizing makes the analysis of the code easier. To utilize the simplification of the analysis the CASE-tool user has to investigate sometime into the preparation of the code before using the CASE-tool (see also 4.1 Preparation phase). The time profit $T_{pr}$ by using CASE-tool can be described mathematically (Rainer[11]) as time profit margin:

$$T_{pr} (t, t_{case}) = t - t_{case}$$

where $t$ is the required time for validation without CASE-tool and $t_{case}$ is the required time for validation with CASE-tool. The time profit depends mainly on the efficiency use of the CASE-tool.

The use of CASE-tool has not only an influence on the validation time, it has also influence on the rest error rate of the validation. The rest error ratio of the validation will be reduced by using CASE-tool. This quality improvement depends mainly on the person who carries out the preparation of the validation object and the assessment of the analysis results. The rest error ratio $r_r$ can be quantitatively described as:

$$r_r = \frac{N_u}{N}$$

where $N$ is the number of all items and $N_u$ is the number of all undetected errors. The quantitative view of the error ratio has been discussed in more detail by Sethy [E3]. The quality improvement can be described as the ratio of the rest error ratio without CASE-tool and the rest error ratio with CASE-tool:

$$V(r_r, r_{r-case}) = \frac{r_r}{r_{r-case}}$$

called also improvement factor $V$. The time profit and the quality improvement can also be seen in the economical view. The CASE-tool represents a bigger investment for firms. Therefore, an economical justification for such investment is required. In the application time of the CASE-tool the user has to do $M$ numbers of validation. The costs per validation $C_{val}$ are:

$$C_{val} = \frac{INV}{M}$$

where $INV$ is the investment for the CASE-tool (including costs for training and price for CASE-tool). This costs per validation can be transferred in a time equivalent $T_{val}$ as followed:

$$T_{val} = \frac{C_{val}}{C_{man}}$$

where $C_{man}$ is used for the cost manpower per hour.

The investment of the CASE-tool will be justified if following condition is fulfilled:

$$T_{pr} (t, t_{case}) > T_{val}$$

in words: the time profit has to be greater then the time equivalent $T_{val}$. The quality improvement of the rest error ratio can be quantified economically by using the mean costs of error consequences. The quantification of these costs depends on the user of the validation object and they can vary in a wide range. Discussing this problem would go far beyond this paper and therefore it cannot be done here.
5 Conclusions

The safety relevant software has to fulfill the safety requirements and has to be validated. These requirements and software validation methods are widely independent from the field of software application. As it has been shown here, the use of CASE-tools for software validation delivers among other advantages an improvement of the time- and quality aspect. It should be considered that the time for the analysis is reduced and the time for assessing of the results increase. The quality improvement of the validation results depends also on the person, who validates the software.

As it has been discussed above, the CASE-tools cover methods of the white box testing group. For a complete validation of a safety critical software some methods of the black box testing group also must be carried out. The CASE-tools will point at possible errors in the code. These errors have to be confirmed by other validation methods. The use of CASE-tools may replace some parts of the conventional test methods. However, the tools cannot replace the other validation methods without previous validation of the software tools themselves. Therefore for the safety validation a certain form of matrix structure of proving methods should be applied, which guarantees a maximum safety of proved software. It means, that several proving methods, which can overlap partially, should simultaneously be used for software validation.

6. References

8. DB Mü 8004. Grundsätze zur technischen Zulassung in der Signal- und Nachrichtentechnik, Deutsche Bundesbahn, Bundesbahn-Zentralamt München, Munich, Germany.