Avoiding Malfunctions Due To Software Failures by Automation of Software Production and Testing

Pannen wegen Software-Fehlern vermeiden durch Automatisierung von Software-Produktion und Test

Rainer Gerlich

BSSE System and Software Engineering

Auf dem Ruhbuehl 181
D-88090 Immenstaad, Germany

Phone +49/7545/91.12.58
Mobile: +49/171/80.20.659
Fax +49/7545/91.12.40
e-mail: gerlich@t-online.de
URL: http://home.t-online.de/home/gerlich/

Abstract: A number of reports about malfunctions of hight-tech components in vehicles appeared recently. To a major part, such malfunctions are related to software. As the amount of software will steadily increase during the next years, software will dramaticially impact the quality of service in future. The competence to produce robust and dependable software will be essential to keep a leading role on the market. Experience from the production of mechanical or electronic goods shows that only by automation productivity and quality can be sufficiently improved. So far software is developed in a handcrafted manner, the needed effort is high and the quality is poor. By the technology "Automated Software Production and Test" (ASaP) the productivity and quality is significantly increased as recent benchmarks do show. This technology introduces a repeatable production process which can continuously be maintained.

---

1. INTRODUCTION

During the last months a number of reports appeared in news papers, e.g. in Wirtschaftswoche [1,2], which complained about malfunctions in hight-tech components of vehicles. To a major part software caused such problems.

These problems reflect the state-of-the-art of software development methodology. Human involvement still dominates, although a number of software tools is available. Just recently Charles Simonyi, a co-founder of Microsoft Inc. complained [3] "today we can design and control production of jumbo-jets by computers, but programming still remains a handcrafted task".

The problems due to software failures will impact products more seriously in future, because software is considered as the means to provide sophisticated functionality. A car manufacturer which already increased the functionality by software, lost the leading position regarding quality due to serious malfunctions [2].

It is easier to produce software than to prove its correctness. And it is much more difficult to guarantee generation of correct software. First, engineers do not have sensors to recognise software bugs, they do need means for that. Second, engineers can only master correctly a limited degree of complexity. As there is lack of appropriate sensors, engineers soon exceed the limit they can reliably master, and together with limited verification capabilities, this explains the poor quality of software.

Handcrafted production of software is very expensive and whenever an engineer generates software the same bug rate applies. This way the quality of software never can be increased dramatically as it is needed. The time-to-market is high because any software production needs strong human involvement.

Outside the software domain such problems have been solved by automation. Automation reduces the development time and costs. By a precise production process the same quality can always be guaranteed, and the quality can continuously be improved by maintenance of the production process.

The technology "ASaP" (Automated Software Production and Test") applies the well-known idea of automation to software development . ASaP is an outcome of an initiative launched by ESA beginning of the 90's aiming to reduce the risks associated with software development. BSSE continued these activities, applied the approach to own projects, and improved the methodology due to the obtained feedback. In 1999 the first tool environment was ready to support fully automated production of software for a distributed real-time system. This system has been accepted at the customer's full satisfaction. In meantime, BSSE extended the automation approach to other areas. By benchmarking of the projects BSSE proved that productivity and quality can be easily increased by a factor of 100.

2. ASAP AND AUTOMOTIVE APPLICATIONS

In the space domain systems are decomposed into a hierarchy of intelligent, dependable items like sub-systems and equipment. Therefore the software architecture covers management of sub-components, fault recovery, system reconfiguration. "Fail-operational" and "fail-safe" capabilities are a "must".

In vehicles, components are becoming more and more intelligent. While in early days no software was included in servo-systems for breaks and steering, the amount of software is now more and more increasing, and its complexity, too. However, this also causes a number of new problems related to the poor quality of software., where the term "poor" compares quality of current software with what is state-of-the-art in other areas than the software domain.

The functionality of vehicles is a matter of competition, and it strongly depends on software, today and even more in future. We see the following main areas where software is or will become very important:

Cat1 software used for vehicle construction

e.g. software for analysis and design of the mechanical structure

Cat2 software residing in the components of a vehicle

-- aggreagates like engine, breaks and gear have to be coordinated in a dependable manner

-- control of a car by joy sticks ("drive-by-wire") requires highly dependable software

Cat3 software needed to support the driver ("infotainment")

-- display of status data

-- data filtering

-- warning/pre-caution and prediction systems

-- navigation, driver guidance

-- software supporting communication and entertainment

-- integrated display and control of communication and multi-media systems

Cat4 software controling vehicle production

-- robot software

-- status of production equipment

The problems reported in the past are related to category 3, driver support. They were mainly caused due to insufficient testing of exceptional situations like inadvertent operation by keyboard input. In such cases the system was not capable to handle the inproper input. Due to the high number of possible combinations such cases could not be tested.

Another problem is: how can all nominal and exceptional cases be handled and verified. The ASaP approach provides means to tackle such problems.

Similar problems exist for Cat2, especially when looking for "drive-by-wire". While malfunctions of Cat3 "only" cause user complains, in case of Cat2 safety of users is heavily impacted.

To solve such problems, first, ASaP gives engineers more time for principal investigation, because - due to automation - they don't need to spend much effort on implementation, but they can take the saved time to concentrate on system operations. Second, ASaP demands to cover exceptional situations, too. And it provides the means to stimulate the system by nominal and exceptional cases. Third, ASaP automatically inserts assertions and instrumentation, so that fault propagation is suppressed, failures can be detected when and where they occur, and a system's properties are visualised. Fourth, a lot of experience comes with ASaP which allows to identify optimum test and verification strategies.

E.g. instead of manually deriving test cases for a large input domain which is spawned up by a number of sub-domains, ASaP allows to concentrate on the sub-domains and to automatically derive the resulting product domain. Moreover, it guides towards an approach by which faults can be handled although not yet known. Finally, ASaP may report a problem, although engineers are not aware of this problem.

In case of Cat1 we believed that analysis and design of mechanical structures can well be handled by the existing simulation packages. But we were recently told by a customer, that this is not necessarily true. If the standard solutions as supported by such tools are insufficient, a lot of manual effort is needed for a tailored solution. Again, such effort can be covered by automation.

Similarly, it is for Cat4. Powerful CAD tools exist which provide automatically the positions for welding, riveting, glueing etc.. However, the robot programs are still manually generated, e.g. by teach-in, and it is an open question at the end, whether such programs really comply with the CAD data. Thousands of such positions exists per car. As the number of robots will increase, there is an emerging issue on verification of robot programs.

In a first step, the robot programs could be automatically evaluated by ASaP and checked for compliance with CAD data. In a later step, it should even be possible, to derive the robot programs automatically from the CAD data and to obtain immediately inherent correctness.

3. EXAMPLES

This chapter describes projects which were executed in the domain of distributed real-time systems. Currently, much more applications exist which are not listed here.

Fig. 1a shows the hardware architecture of a 2-processor real-time system as built for the "Material Science Laboratory" [4] to be used on-board of the International Space Station ISS. A number of complex peripheral hardware is attached to the two embedded Sparc processors: mass spectrometer, heaters, vacuum pumps, pyrometer etc. which have to be operated under µg (micro gravity) conditions. On-ground commands have to be processed, a large number of data items has to be acquired and maintained in the distributed, synchronised database, and telemetry frames have to be downlinked. Extensive exception handling and checkout is needed at run-time. The system definition lists about 225 internal commands, 150 external commands, about 40 states, about 280 state transitions, and more than 1000 functions. Asynchronous commands with timeout conditions are executed for commanding, supervision and reporting, and cyclic activities to control the peripherals. A MIL-bus is used for data distribution and for communication between both processors.

The system shown by Fig. 1b represents a distributed critical control system for commanding of a back-up Diesel power supply of a nuclear power plant. This system was defined in the course of the ESPRIT project "CRISYS" [5] to exploit the problems due to distribution. A number of data are acquired from the hardware and processed throug a number of data processing and control logic stages. To ensure high availability and dependability, the system is equipped by up to three redundant components. Each data processing computer is monitored by an independent computer. At the final stage, a decision is derived by voting. In total, the system includes 16 processors. A synchronous approach is applied to data processing. ASaP provided an asycnhronous execution environment to investigate the impact by data faults and time jitter.

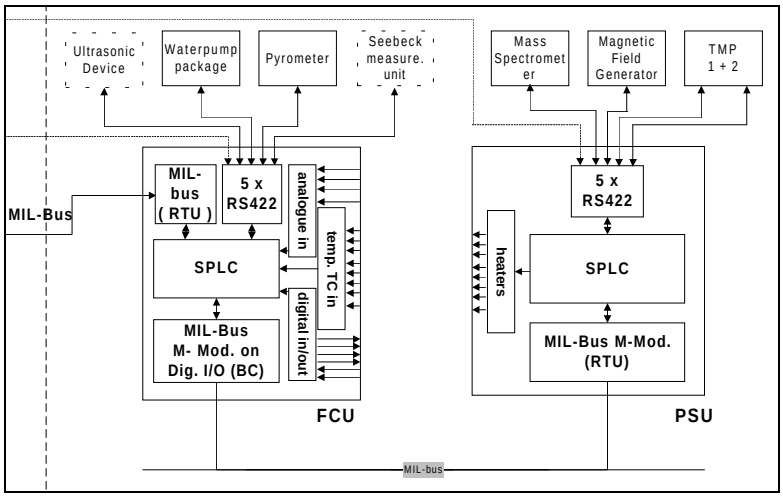

Fig. 1a: Distributed Real-Time Processing (asynchronous + synchronous processing)

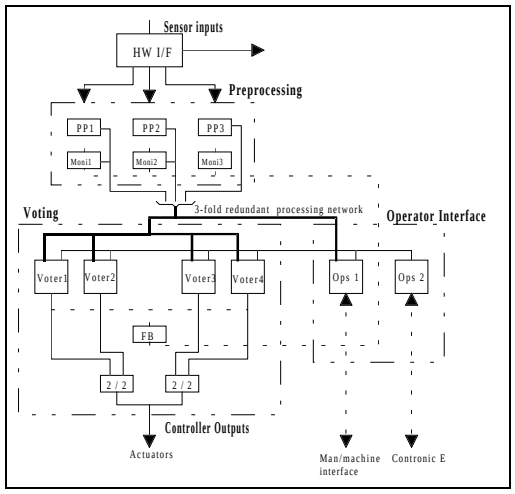

Fig. 1b: Highly Redundant Distributed Voting System (synchronous processing)

4. LESS HUMAN EFFORT, LESS BUGS

Whenever an engineer produces software, bugs are introduced. The bug rate depends on the engineer's education and tool support. Typically, such bug rates are in the range of 1 .. 0.1% per line-of-code (LOC). There is a principle limit, and better education cannot decrease the bug rate by a number of magnitudes as it would be required.

Therefore the reduction of human effort is the key point to solve the problems of software production. This requires rigorous analysis of the human activities and replacement of most of them by full automation. It is not sufficient just to automate some parts, e.g. to derive automatically code from inputs which an engineer provided via a graphical interface. What is applied today is nearly a 1:1 conversion of the engineer's inputs, and there is not really an added value regarding productivity, quality and time-to-market.

If the human effort is reduced from 100% down to 1% this immediately yields a decrease of the bug rate by a factor of 100, in fact it is more because the complexity is reduced which in turn lowers the individual bug rate.

Although an automated production process is also based on "handcrafting", it is subject of continuous improvement. In addition, there are further means to reduce the bug rate. First, due to the limited size of the process kernel all software related to automation can be intensively tested. Hence, it is better tested than large pieces of software which are generated from scratch. Second, the reuse of such automation software is much higher, which increases the test coverage even more and in consequence, the probability to detect a bug.

ASaP implements the strategy "less human effort, less bugs" by

- involving an engineer only in system definition instead of system implementation,
- reducing the dependency on the size of the system definition from higher orders to lower orders, preferably to first order or even constant effort,
- limiting the effort by appropriated organisation of the development approach e.g. to cover an infinite set of variations by finite and little human effort.

When an engineer is only involved in the definition instead of implementation, the complexity is decreased down to a level which can be better handled by a human being - apart from the reduction of effort.

So far, the principal approach is that engineers are always available to solve problems related to (bad) organisation of work. We have observed that the current development environments, like compilers or other tools, implicitly expect that an engineer is available to solve a problem which suddenly comes up. It is a real challenge to master such problems automatically and without human involvement, but we succeeded.

Moreover, it seems to be a standard procedure to execute work without thinking about if and how effort can be reduced. E.g. in case of a TCP/IP network engineers need to set up n(n-1)/2 point-to-point connections and to maintain them. Why shouldn't it be possible to involve engineers in the definition of the network nodes, only, and then to do the rest of the job automatically? In this case an engineer only has to define the nodes which drastically reduces the human effort., because the effort for implementation of all the point-to-point connections disappears completely.

Another example for saving of effort by an ASaP strategiy is instanciation of functions for user-defined types. The state-of-the-art approach as defined by the object-oriented paradigm is based on classes and inheritance which requires human involvement for each user-defined data type. Even if the human effort is small, it limits the potential saving significantly as manual intervention is needed for each new data type.

Therefore ASaP recommends a quite different approach which is highly efficient. It automates inheritance so that no human intervention is required. What remains is the definition of a very limited set of simple functions, less than 10 per new function, but not for each new data type. This way, an infinite set of types can be automatically derived with very little human effort.

The evaluation of ASaP projects shows that by automation about 5 .. 100 bytes code can be automatically derived from 1 byte manual input. The ratio is the higher the bigger the task is, i.e. for more complex tasks like generation of the infrastructure of a complex real-time system about 10 times more input bytes may be required than for generation of sequential code, while the output is 100 times higher. So in fact, less input is required for more complex tasks.

In case of testing, verification and validation ASaP applies the same organisation principles: not to involve engineers into a large amount of activities, e.g. which are a product of sets of sub-activities. It is much more efficient to involve the engineers only into the sub-activities (sub-spaces) which drastically reduces the effort. Instead of the product only the sum of activities has to be executed.

5. LESS HUMAN INVOLVEMENT, HIGHER FLEXIBILITY

The higher the degree of automation is, the shorter is the development time and the less are the costs. Consequently, maintenance is made easier and cheaper. In fact, automation allows to spent more time on system optimisation. Hence, development of a system by automated software production means incremental development by continuous maintenance. In case of manual development optimisation is nearly impossible: when the first representative version is available, there is not so much time and money left to change something.

As experts for implementation are not needed, a system engineer can easily generate a system which is immediately ready for use. And he can easily change it, if the current version does not meet his expectations.

Similarly, due to the short turn-around time, portation or migration of the system or parts of it between different platforms (processor, OS) is very easy and fast. It is even possible to simulate a distributed system consisting of n processors on a subset of m processors with $1 \leq m \leq n$.

As ASaP provides an executable system right from the beginning, integration can start immediately when the first version has been created. This allows to minimise the risk early enough when external software needs to be integrated.

6. EXECUTABLE SPECIFICATIONS

By ASaP representative versions of the full system can be early and easily generated. Even if functionality or hardware is still missing, the behaviour, interfaces and exception handling can be made representative by such a version. The major components of such a version, e.g. the processes in case of a real-time or client-server system, may be considered as "executable specifications".

In case of a subcontractor hierarchy such a representative version of the full system may be passed over from a customer to a contractor. The relevant components may be taken as contractual entities to which the contractor's deliverable needs to be compliant. This way the contractor can always integrate his components into a fully representative system environment and check it against such constraints, and the cusotmer can do the same when getting deliverables from contractors. This allows seemless and continuous integration of external components, and is considered as a major contribution to risk reduction and contractor management.

7. ASAP - AUTOMATED SOFTWARE PRODUCTION AND TEST

ASaP divides development, test, verification and validation of software into two principle classes: (1) to build new software from scratch, and (2) to test, verify and validate existing software. Case 1 can again be divided into two sub-categories: construction of a distributed / real-time infrastructure and generation of sequential software which can be plugged into such an infrastructure, or which may be completely independent and self-standing.

7.1 Generating Software From Scratch

Fig. 2 shows the principle development cycle of ASaP. The principal points of human interaction are at the left top box and at the bottom box: delivery of a minimum of information ("User Inputs") and reading of the report on the system properties ("Result Evaluation"). To change the properties or to add incrementally properties and functionality, the inputs are modified. In case of structural changes like number of processes, topology, platform changes, a "major cycle" has to be (re-)executed, in case of functional changes a "minor cycle" is sufficient - as it may occur during integration having received a new version of a deliverable.

The production process is based on experience and organisation and consists of two principal steps: the generation of an executable system for the host and/or target environment, and visualisation of the system's properties during development and delivery of the final version, respectively. By utilities and templates the ASaP software builds source code and what is needed to derive an executable, and it provides a run-time library.

Optionally, the code is instrumented by assertions and reporting capabilities, and by stimulation capabilities for nominal and stress testing and fault injection including test case generation.

To meet the goals for productivity and quality it is essential to fully automate all the steps between delivery of user inputs and delivery of the executable system and the evaluation report.

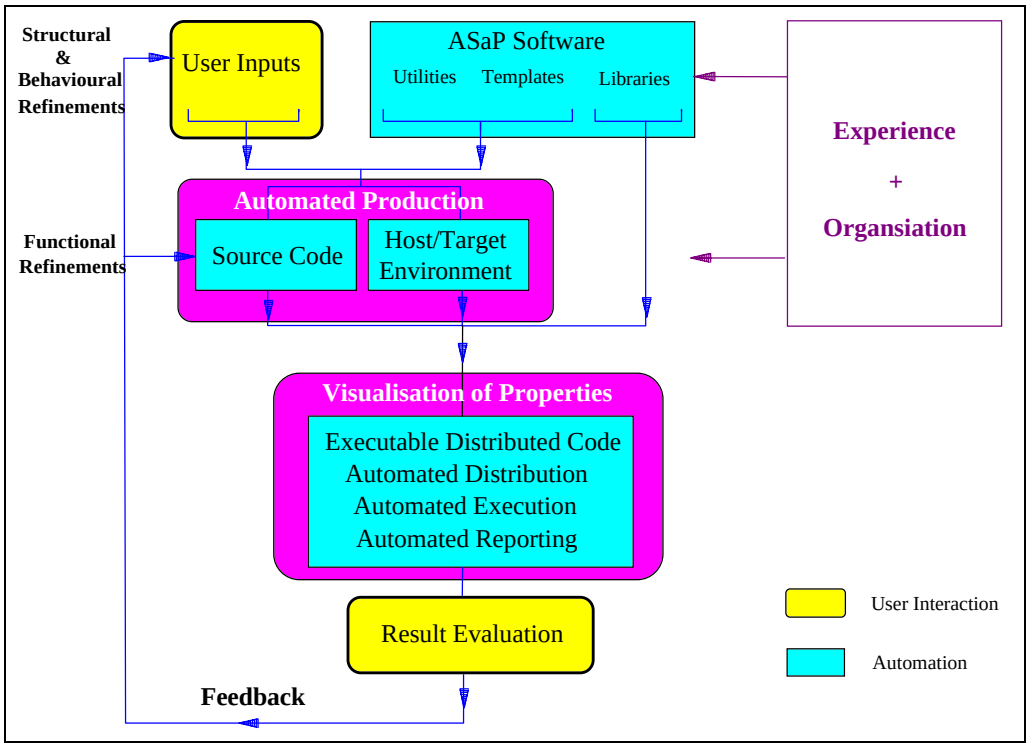

Fig. 2: The Principle ASaP Development Cycle

7.2 Test Automation

Fig. 3a and 3b show the ASaP approach for testing.verification and validation. While usual test automation starts after manual test case description, ASaP automates this task as well. This is an essential difference to existing approaches and complies with what was previously mentioned: do not involve engineers into a product space of activities.

Consider e.g a user who has to operate a keyboard. Applying the conventional approach means: an engineer thinks about what a user's inputs could be. Then he writes test procedures and such procedures will be forwarded to the test automation tool. Unfortuneately, a lot of effort has to be spent for definiing and writing of the test procedures.

By ASaP a model of the user is defined, and an "automated user" stimulates the system. Inputs and outputs are logged and presented, and the engineer can merge his data stream on result evaluation with the automatically generated data stream. What is left up to the engineer, is the definition of the user model, a simple and small input domain, the evalaution of the pre-processed results, and generation of a data stream on what is considered as correct out of the reported data stream. Apart from nominal inputs, this approach also covers stress testing and fault injection inherently.

The same is true for verification and validation. Existing source code can automatically be instrumented by assertions and instrumentation for reporting.

The procedure which is applied for automated testing is similar to test case generation when producing a new system. While in latter case the test cases are derived from the specification, in case of pure test automation this information has to be extracted from the existing source code or equivalent items.

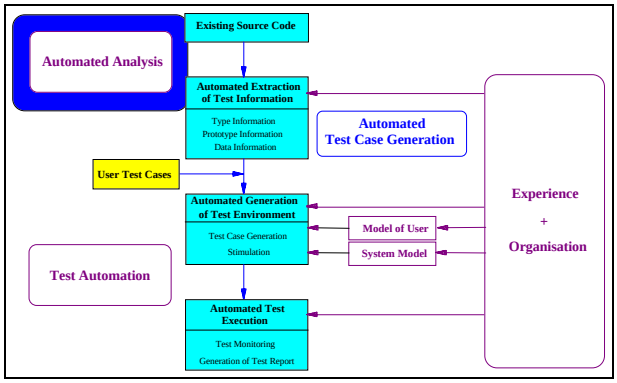

Fig. 3a Principles of Test Automation

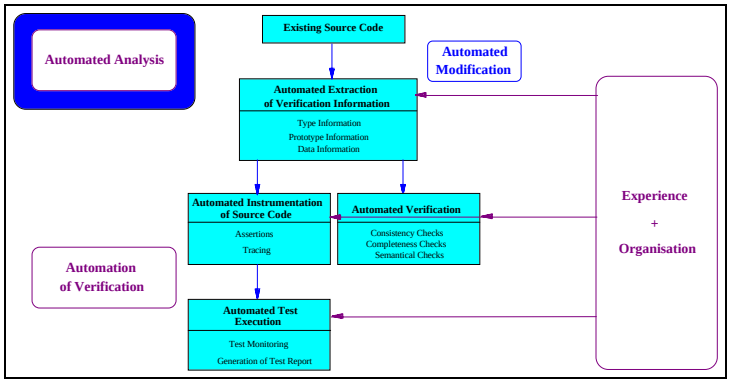

Fig. 3b: Principles of Automated Verification and Validation

8. RESULTS

Evaluation of recent projects [6] shows that a dramatic increase of productivity and quality can be achieved when applying ASaP:

- an equivalent of more than 5 man-years (up to 100 man-years) can be generated during 1 hour on a PC-800MHz

  remark: of course, such productivity can easily be achieved today during compilation etc., however we are talking about generation of executable systems from scratch, e.g. by taking only spreadsheet data.

- a bug rate in the range of 10^{-5} has been observed when generating a distributed real-time system

  For other projects a bug rate of 0 (zero) was observed, e.g. when generating a synchronised, distributed, memory-resident database

  More precisely, 2 bugs were reported for an amount of about 80,000 operational LOC, the complete environment, which was automatically generated within about 30 minutes, amounted to about 200,000 LOC.

  A bug rate of about 10^{-2} is usual, a bug rate < 10^{-3} is considered as very good [7].

  remark: these two bugs were related to the production process, which has been corrected.

The performance of the database could be improved by about 30 .. 40 % just be regrouping the data items in the spreadsheet. The distributed data base was integrated and tested on a 2-processor system which was simulated on one processor, only. When the full hardware was available again, the system run immediately and correctly on the 2-processor system without requiring any human intervention except to tell ASaP that 2 processors are available.

The structure of the MSL system was changed from 38 process types at the beginning to 32 at the end, just by modifying the spreadsheet.

In the CRISYS project the impact of data faults and time jitter could easily be evaluated, and the sensitivity of the control software in presence of such disturbences be demonstrated, although previous (theoretical) analyses concluded there wouldn't be any problem.

9. CONCLUSIONS

The current results as derived from ASaP projects show that a dramatic increase of productiivity and quality can be achieved when the concept of full automation is applied as supported by ASaP. Due to automation, testing, verification and validation is improved, the test coverage is increased at reduction of test effort. This way identification of malfunctions of software is possible before a system is released, even if a lot of tests are required.

REFERENCES

[1] Wirtschaftswoche, July 2002

[2] Wirtschaftswoche, September 2002

[3] VDI Nachrichten, September 17, .2002

[4] MSL - Material Science Laboratory on-board of the International Space Station ISS

[5] CRISYS - Critical Control System, ESPRIT Project EP 25.514, 1997-2001

[6] Eurospace Symposium DASIA'02 "Data Systems in Aerospace", May 13 - 16, 2002, Dublin, Ireland
R.Gerlich:"Benchmarks on Automated System and Software Generation -
Higher Flexibility, Increased Productivity and Shorter Time-To-Market by ScaPable Software"
"ScaPable" = scalable and portable

[7] N.E.Fenton, N.Ohlsson: Quantitative Analysis of Faults and Failures in a Complex Software System, IEEE TSE, Vol. 26, No. 8, August 2000