An Approach for Testing Safety-Critical Software

Weiwei Li, Zhongwei Xu, Yan Jin
Dept. of Telecommunication Engineering
450 Zhennan Rd, Shanghai Tie Dao Univ.
200331 Shanghai, P.R.China
Tel: 86-21-56220676, Fax: 86-21-56220676
E-mail: Liweiwei@263.net

Abstract

A novel approach for testing the effectiveness, efficiency, safety and relative appropriateness of Computer Interlocking Software (CIS) --a kind of safety-critical software is presented with a software platform developed to support this approach. A brief description of the proposed approach is also included.

Key Words: Safety-Critical Software, Failure Severity Level, Failure Frequency, Software Safety Integrity Level, Software Validation

1 Introduction

Safety-critical software is needed in such fields as missile control, nuclear power plant, railway signaling system or space shuttle system where the failures of the system will cause many deaths or tremendous financial losses.

Most of Chinese railway signaling systems are currently in the progress of carrying out the technology changing from relay-based to computer control interlocking, while some have already introduced Computer Interlocking System, but the users and the administrators always doubt whether the system can be safe, especially in some special situation.

The CIS aims at rendering the whole system safe or at least fail-safe. However, because of system complexity and configuration, administration errors, unauthorized use and misuse or abuse by "authorized" users, this goal is unlikely to be obtained for most systems. For this reason, the emphasis on system safety testing, especially on software safety testing is increasing. The test of hardware components in safety-critical system has become mature over the years and commonly followed techniques have emerged. However, methods and techniques for testing the reliability and safety of the software part of safety-critical system are relatively new and still maturing.

Software safety testing can increase trust in safety-critical software such as interlocking software.[1] This objective was achieved by using a novel approach integrated with some techniques such as fault injection, stress testing, software validation, etc.

2 Description of the proposed approach

The conventional approach to testing the CIS is mainly by manual testing. It is time-consuming, limited, ineffective and doubtable.

We advance a novel approach which is keen on the function robustness and the safety of the objective software. Human-operator components and operator-induced software failures in safety-critical system are also considered. The approach proposed is not restricted in the area of interlocking software testing. It is general and most of its operations are automatic.

0-7695-0104-4/99 $10.00

2.1 Black-box based testing

In view of copyright, security and so on, it is proper to choose black-box testing, in which the tester need not know what is inside the program, and what interests him is the relationship between the inputs and the outputs.

Figure 1 gives the rough structure of the test and assessment system. The approach uses comparison to get the test results. Firstly we upbuild a control software in the test platform, then input the software under test and the control software with the test information in the Expert System(ES), after that, their outputs are recorded and compared, at last, we obtain the test and assessment results by statistical analysis.

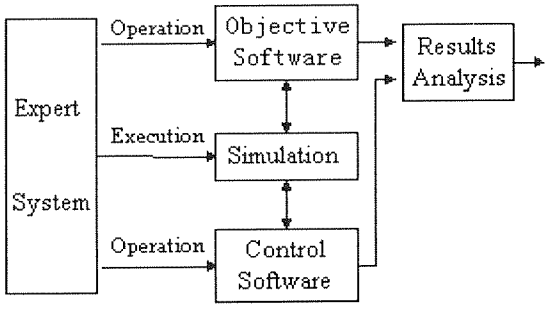

Figure 1 Structure of the system

2.2 Method of assessment

Failure Severity Level(FSL), Failure Frequency(FF) and Software Safety Integrity Level(SSIL) are important in evaluating whether the software safety meets required safety goal, if safety goal is defined as reducing severity and frequency of failure that would cause loss of life or abort of mission to an acceptable level of risk. The FSL are classified as catastrophic, critical, marginal and insignificant. Six kinds of the FF are known as frequent, probable, occasional, remote, improbable and incredible. The SSIL is formed by synthesizing the FF and the FSL and is divided into five levels as fail-safe, high-SSIL, mid-SSIL, low-SSIL and no safety[3].

Each input corresponds to a certain output. We believe that the control software totally differs with the software under test, so we omit the possibility of the same error output of the objective software and the control software under the same condition. The results of the two softwares are compared and the different results are sorted according to the SSIL.

2.3 Recording and replaying

Recording and replaying feature is a way to help create test cases. Although many test cases are generated by a certain graph-search algorithm, it is unwise to create test cases by searching or other mathematical methods when facing those cases with no clear logical relationship, such as those conducted by experienced experts at will. The proper method is to record and save them as special test cases. By using the recording and replaying feature, not only can we create a test case quickly and easily, but also create and replay concurrent scripts in synchronization.

2.4 Fault injection and stress testing

As to a complex software system, whatever approach is used, it is impossible to conduct complete test because of lots of the possible input data and the complex internal structure of the program. A good approach is to make the fewest tests and detect as many errors as possible.

Most software applications can operate well under normal circumstance, but in abnormal situation, they will expose many problems. It is required that safety-critical software should be fault-tolerant, i.e. when limited software or hardware faults emerge, the software should operate continuously and properly. Fault injection and stress testing, which are firmly related to the test cases, are powerful tools to make errors exposed quickly. According to the SSIL, we mainly address two kinds of test cases: function testing cases and safety testing cases.

Function testing cases focus on two aspects. One is regular inputs to verify the rudimentary functions that the software should meet according to technical specifications. The other is some allowable abnormal inputs within the specifications or the frequently happened fault inputs.

Safety testing cases are specially tailored for safety-critical software, and they are divided into two sets:

Set A: Stress testing cases. They use one or two abnormal inputs probably or occasionally happened, which may cause critical or catastrophic consequences, or some concurrent inputs which aim at checking how the CIS is affected by "stressful" conditions in the operating environment.

Set B: Worst testing cases. They are strong stress testing under severe circumstance, which means that the cases are impossible to happen or have never happened before, but they will cause catastrophic consequences in most cases if they happen.

2.5 Validation

Validation is an important phase in the life cycle of software development, especially for safety critical software. No validation means less dependability and, more seriously, severe injury. We develop a control software according to the software requirement documentation. Not only should this control software have all the functions of the real one, but also it should be more reliable and safer. Because the control software serves as a comparative standard for the objective software, people always expect it to be perfect.

In fact, to some degree, the software safety is determined by the complexity of the software itself under similar development circumstance (including the developing tools, quality of the developers, etc.), and the safety of the control software will directly influence the quality of the black-box based testing. Besides proper developing tools, methods, workgroup and so on, we should use some special techniques to decrease the complexity of the control software.

With the special information acquisition interface, the control software can know the fault information or the intention of the test cases; while the software under test should be smart enough to judge and handle those abnormal inputs. Figure 2 shows the inputs and outputs comparison of the software under test and the control software.

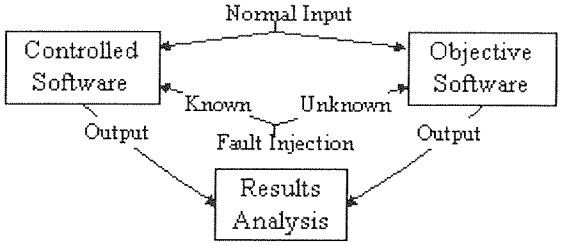

Figure 2 Comparison of two softwares

Studies[2] have shown that the developers of the control software and the software under test will make the same amounts of defects in the process of programming. It is no doubt that there must be some defects in the control software, however, as the control software has no real outputs, the defects in the control software will not cause serious consequence and more importantly, the defects can be corrected through the test processes. It seems that the testing of the objective and the control software is mutual. The more the objective softwares are tested, the more trust the control software can earn.

2.6 Result analysis

All the test results are recorded and analyzed mainly in two parts: function analysis and safety analysis. Function analysis focuses on function implementation and robustness and it is mainly based on the test results of function testing. The conclusion is that almost all the function testing results should be correct or proper. Safety analysis is mainly based on the safety testing results. Figure 3 shows the conclusion map of the safety analysis.

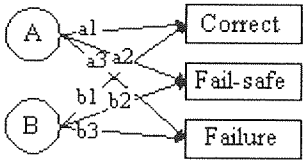

Figure 3 Conclusion map

In figure 3, "A" means stress testing, "B" means worst testing. "A" and "B" each has three kinds of outputs. It is obvious that the "Correct" and the "Fail-safe" results are preferred. We can draw the following conclusions:

- {a1} and {b1} are the most expected outputs;
- {a2} and {b2} are degraded outputs that can be

acceptable;

●{a3} and {b3} are not acceptable, they should be avoided in safety-critical software;

●An acceptable safety-critical software should at least meet $\{a3\}=\phi$;

●If $\{a3\}\cup\{b3\}=\phi$, it can be thought as a qualified safety-critical software.

All the detailed defects are reported to the administration agency and the developing workgroup. It is the responsibility of the administration agency and the users, not the testers, to decide whether the software should be applied, modified or taken a regression test.

3 The test tool: T&AP

Test and Assessment Platform(T&AP) is developed to support this testing approach, which mainly consists of test commander, field simulator, control system and results analysis. The test commander is an ES that contains many test cases. It can

● input the software system under test with typical, suspicious and completely wrong behaviors, which make up the operating sets, the running sets and the fault-injection sets, etc;

● record and replay commands from the knowledge of experts;

● inform the field simulator to inject fault to the software under test;

● conduct stress testing and worst testing.

The control software, which can receive the same inputs, is used to validate the software under test. The only difference between the software under test and the control software is that the latter can receive internal announcement in special situation such as faults created by field simulator. The outputs of the control software are an important basis for the results analysis to determine the relatively correct response by comparison. The field simulator, for testing purposes only, mimics the behaviors of the system control by the software under test. The results analysis draws inferences from the inputs of the test commander and the status of the field simulator. And it makes an impartial evaluation to such things as the effectiveness, efficiency, safety and relative appropriateness of different plans of action, factors that would strengthen or weaken the prospects of success for a proposed plan of action, the function implementation and the SSIL of the software under test.

The CIS of Fuzhou railway station, which hosts on the Programming Logical Controller (PLC) has been tested and evaluated. In function testing, we found that it has some problems in two significant functions and one of its components can not work. In safety testing, table 1 shows the results.

Table 1 Results

	a3	b3
Failures	5	9

Because of the failures in the testing, we recommend that it should be modified and taken a regression test.

4 Conclusion

The success in the first testing has greatly encouraged us to conduct further research and in the following testing we obtain more experience. Even though testing may not expose all potential defects in the objective software, the problems detected are analyzed to get an evaluation about the performance and safety of the software. The general conclusion is that, in principle, a combination of testing and validation will be successful in measuring safety-critical software.

References

[1] Norman F.Schneidewind. "Reliability Modeling for Safety-Critical Software", IEEE TRANSACTIONS ON RELIABILITY, Vol.46, No.1, MARCH,1997

[2] William Perry. Effective Methods for Software Testing. Wiley, 1995.

[3] Standard, EN 50128, Railway Application: Software for Railway Control and Protection Systems, European Committee for Electrotechnical Standardization, 1994.