Software Testing Applications Based on a Virtual Reality System

Xian-Yang Zhao, Li-Mei Xu, and Hui Li

***Abstract*—Software testing is an important part of software engineering and has been more and more popular as the rapid growth of the software products market. Good skills of communication with clients and programmers play a significant role for a tester during the test process. This paper presents some important and basic software testing applications (such as static testing, dynamic testing, black-box testing, white-box testing and their combinations) based on a virtual reality system, named as rocket digital simulation system (RDSS). Different testing methods are exercised during the software developing lifecycle and finally achieving significant quality improvement.**

***Index Terms*—Software engineering, software quality, software testing.**

1. Introduction

With the expansion of software system size and complexity, there is an ever-increasing demand for innovative testing schemes for software quality and reliability[1]. Large-scale software development normally experiences three major phases: requirements analysis, software design and coding[2]. In recent decades, software researchers have proposed a variety of methods, which can be used to guide developers to improve the software quality and avoid making mistakes during these three phases[3]-[8]. The characteristics of these methods is, international testing regulations are complied to insure the quality and management of the software during each testing methods. Unfortunately, complete avoidance of human mistakes during software development is not realistic. Therefore, systematic and effective software testing and maintenance are essential to ensure the quality of the software.

In this paper, software testing applications will be illustrated according to the concrete software project. A brief introduction is given in the following sections.

Manuscript received June 19, 2006; revised August 15, 2006.

X.-Y. Zhao is with School of Mechatronics Engineering, University of Electronic Science and Technology of China (UESTC), Chengdu, 610054, China (e-mail:zhao_xianyang@163.com).

L.-M. Xu and H. Li are with Institute of Astronautics & Aeronautics, UESTC, Chengdu, 610054, China.

2. Software Testing

Software testing is an indispensable phase in the modern software lifecycle. It is the process of revealing software defects and evaluating software quality by executing the software[9], [10].

Some basic software testing principles are listed as follows[11]:

First of all, software testing is impossible to test all the process. Because there are a large amounts of inputs and/or outputs for even a simple piece of program. Sometimes, the requirements are not integrated. And software logic paths usually have multi-combination.

In the next place, software testing is risky. A tester will take a risk to accept the bugs if he doesn't test all the potential faults of a program.

Thirdly, software testing could not find out some of the potential bugs. A tester could do some tests and find out some bugs, but no one could make sure that he could find out all the bugs. What he could do is to test again and again.

Fourthly, the more tested the more immune. As testing methods used in the software testing become more and more, it is more difficult to find out new bugs. Thus, it is better to try different testing approaches as more as possible.

Finally, not all the bugs could be repaired. Usually, there is not enough time to repair all the bugs. And some of them are not fatal and it seems difficult to find out the reasons, and some of them could be avoided if users are guided to use it in another way.

The goal of software testing is to find out bugs as early as possible, and make sure they could be repaired[11].

Four of the most important software testing techniques are commonly used in software testing fields, they are:

Static analysis: only examine and verify the unexecuted parts and analyze the characteristics.

Dynamic analysis: the process of executing and testing the software, analyze the corresponding relationship between the inputs and outputs.

Black-box testing: the tester views the program to be tested as a black box while no caring about its inner structure and characteristics. The goal is to examine whether the program has all the anticipated functionality and desired performance. It is also called functional testing.

White-box testing: it focuses on the inner structure of the software-under-test, so it is also called structural testing.

The four techniques mentioned above are not used

discretely, they are combined together to complete some kind of testing in this paper. Static black-box testing can be used to test the specification, dynamic black-box testing is the most commonly used testing method for the function, static white-box testing is quite important in testing the codes, and dynamic white-box testing is extraordinarily effective during structural testing.

3. Structure of the Virtual Reality System

The name of this virtual reality system is rocket digital simulation system (RDSS), which includes seven functional modules: control centre (CC), single equipment demonstration (SED), virtual roam (VR), virtual assembly and disassembly (VAD), aviation simulation (AS), numeric simulation (NS) and system management (SM). The function is implemented based on run-time infrastructure (RTI), which is the kernel of high level architecture (HLA) system, together with 3D model demonstration database and numeric drive, and the network is needed for better performance and demonstration. The software structure is shown in Fig. 1.

From Fig. 1, the structure of RDSS is shown clearly. Control center is the core of this system; it sends orders to the network, which is based on RTI, in order to control the actions of NS, AS, VAD and SM. When receiving corresponding orders, the NS, AS, VAD and SM modules could execute its actions. Compared with other modules, VR and SED are relatively independent, they could take good performance without the control of CC. 3D model database supplies all the 3D models needed in the background. So, this is a complicated software project composed of high technology and communication networks. The following part of the paper represents the software testing applications according to this project--RDSS.

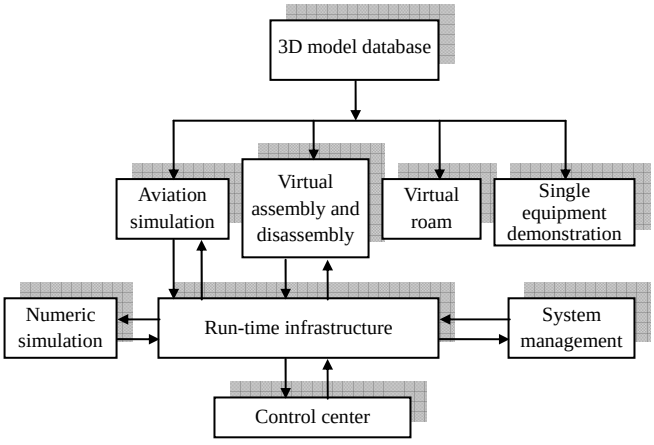

Fig. 1. The structure of the software.

4. Software Testing Applications

4.1 Specification Testing

It is much different from most testers' expectations that the first testing job is not unit testing or functional testing but specification testing [11]. Usually, the specification is a written document which defines the product from these aspects: the details, manipulations, functions, restrictions and so on. Most bugs come from the faulty specification. Because different programmers have different understanding about the appearance, functions of the product and the way to use it, the only way to make the ultimate product meet the clients' expectations is to assure that the description to the product is integrated in the specification.

The technique of static black-box testing is needed to test the specification. To RDSS, complicated software, the first step is to survey the specification at a high level in order to find out some serious problems or something forgotten. Standing at clients' point could help testers have a better communication with clients. Sometimes, much more requirements beyond the necessary function are total waste of time and energy, at this time, testers should express it to the clients tactfully and politely. In fact, it is quite useful to RDSS, the specification is coming clearer, and it is easy for programmers to code.

The second step is to inspect the details of the specification to make sure the goal is clear, the scheme is correct, the description has no antinomy, the function is not incompatible and it could be achieved on time at the state in existence.

There is still something testers should pay attention to, it is a long time to finish specification testing, good communication skills could play an important role and enough patience is needed.

4.2 Unit Testing

Unit testing is performed on individual code modules or logical errors, as well as determining whether or not the product matches its overall design as defined in the specifications [12].

The RDSS is written by the tool of visual C++ (VC), which uses the object-oriented (OO) computer language. The insertion of OO technology in the software industry has created new challenges: intricacy of inheritance, information hiding and encapsulation, as well as polymorphism, especially, making OO software testing much more difficult than the traditional software testing [13]-[16].

Anecdotal evidence from software developers suggests that testing is becoming an increasing percentage of the development budget [17].

In the RDSS, white-box testing method is used to do the test job: firstly, the debugger of VC is used to test logical and syntax errors. Then the automatic testing tools, PureCoverage and Purify are used for further testing. And the errors, such as memory leak and inadequacy of path coverage are cleared.

4.3 Integration Testing

Integration testing exercises the subsystem, which is made up of various modules after unit testing [2].

In the RDSS, static white-box testing and dynamic

white-box testing are two methods widely used in this period. And they use bottom-up integration testing approach.

Static white-box testing is to examine the design, system structure and code systematically when the software is not running, it is also called structure analysis. The advantage of static white-box testing is to find out the bugs as soon as possible when dynamic black-box testing can't find some errors. Examination between workmates is the simplest way of formal examination; public representation is the next step, in this period, codes are read out and running-effects are explained; verification is the last step and it is also the most formal examination, testers will be at different standpoints and point out different bugs.

After static white-box testing, some covert bugs are found out and it is time to do dynamic white-box testing.

Dynamic white-box testing focuses on the inner structure of the software-under-test, thus it is also called structural testing [2]. Testers should have knowledge of that structure in order to get information about where the testing is needed and how to test the codes, and then select test cases to exercise specific internal structural elements to determine if they are working properly.

Bottom-up integration testing approach, illustrated in Fig. 2, is the testing method taken in RDSS:

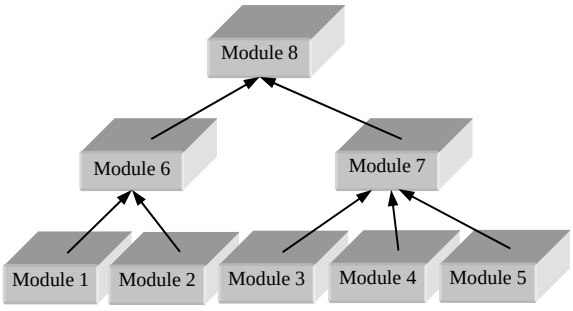

Fig. 2. Test sequence in bottom-up integration testing.

From Fig. 2, the low-level modules are first tested and the upper module is then integrated to form a new subsystem, this subsystem is then tested and the process repeats until the whole system is integrated and tested. The number of the layers and modules is convertible; it all depends on the actual requirement. The merit of this method is that stubs are not required and each module could be tested independently.

4.4 System Testing

The developed software is only a component of the overall computer-based system; it should be integrated together with hardware to constitute an integrated software. The objective of system testing is to find the discrepancies between the actual software performance and its expected performance by comparison with the requirements.

Here in RDSS, two methods, dynamic black-box testing and real-time testing, are commonly used during system testing. Dynamic black-box testing is a kind of functional testing, what it cares about is not the details of codes but the whole function after the developed software integrated with hardware. Several testing methods are as follows:

1) Equivalence partition: it is a process to reduce the test cases from mountains of cases to a rational range when the test cases are similar. That is to say, if there are errors in one of the test cases, there maybe the same errors in other equivalent test cases.

This method is quite useful in testing module SED, because different equipment has the same exhibiting methods in this module. Thus, testers only need to execute this module to test one of the cases.

2) Data analysis: data here includes keyboard inputs, mouse click, print outputs, etc.

In RDSS, some modules require users to input the name and numbers from the keyboard, and usually, 255 symbols, only including letters and numbers, is the maximum. So, three kinds of test cases are selected: firstly, 255 letters are input to do the boundary value analysis; secondly, invalid symbols, such as punctuations, "*", "%", "#", "~", "@" and "Tab", are the test cases needed to test the reaction of the software; thirdly, null symbol is input. If there are something happened beyond the expectation, testers should take them down and submit to the developers to solve the problem.

3) Comparison test: use the same test cases to test the software after some bugs are modified in order to make sure it meets the requirement. This is a useful testing skill in RDSS.

4) Repetition test: it means to repeat one same operation. In RDSS, the simplest way is to start and close the software, to assemble and disassemble the same equipment in the module of virtual assembly and disassembly, and to keep one model running time after time in the module of Numeric Simulation. Much errors and bugs are found out by this simple but bothering method.

5) Pressure test: keep the software-under-test running under worse condition, such as smaller memory, smaller disk space, slower CPU, to test the how the software require and depend on the external resource. It seems like boundary value analysis, and the necessary external condition could be tested.

Real-time testing is quite important in RDSS. Because the most important modules, control center, virtual assembly and disassembly, aviation simulation, numeric simulation and system management, are based on the network and a kind of real-time software, MakRTI. And MATLAB-RTI middleware is an important bridge to connect the background process, MATLAB workspace and VC workspace.

Three computers and a HUB could build up network environment to satisfy this software running condition. Data mapping testing, command alternation testing and synchronization testing are three important test emphases. The debugger of MakRTI is the simplest and most effective testing tool. After analyzing the structure of MATLAB-RTI middleware by static white-box approach, testers could set proper breakpoint and execute the program step by step to

test whether it satisfied the requirement. And the feedback of detected defects, such as control delay and packet loss, could be sent to corresponding developers for repair. Fig. 3 is a MATLAB-RTI testing picture:

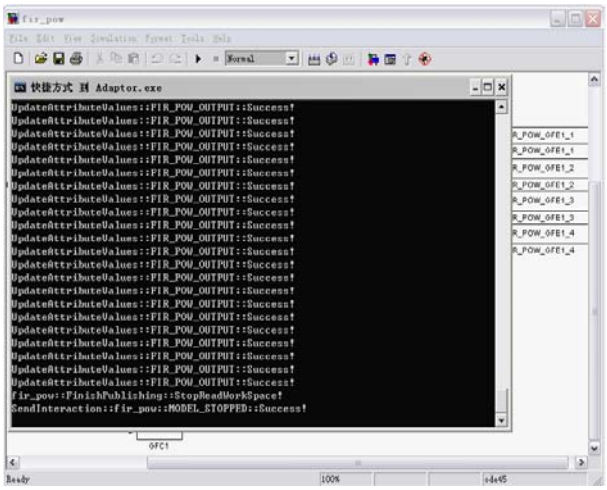

Fig. 3. MATLAB-RTI testing picture.

4.5 Performance testing

The performance testing is made up of testing for software availability, reliability, survivability, flexibility, durability, security, reusability and maintainability [2].

So many performance testing have only one objective: satisfy the captious clients' requirements. According to the feature of RDSS, performance testing is carried out from these two aspects: visual performance testing and numeric drive performance testing.

1) Visual performance testing:

Visual performance plays an important role in the scene simulation system of the product requirement; therefore, it is the primary testing task.

Firstly, in the early developing period, bad real-time performance is a big trouble, which is due to the complicated 3D models, and some suggestions are given: abnegate the sightless and quite small details, replace the unnecessary details with texture picture, and use the simple models instead of complicated models. Numerous facts have proven that these suggestions take positive effective.

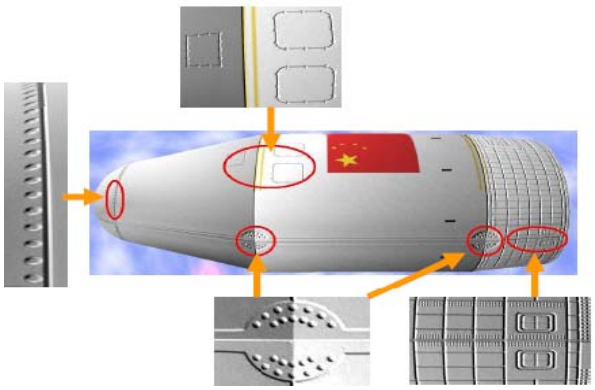

Fig. 4. A Bump-mapped Cowling of Rocket.

Secondly, a large amount of 3D models can't satisfy the third dimension requirement. Therefore, three kinds of important techniques are commonly used. 3D illumination technique is used to make the visual effect much more vivid; texture mapping technique including bump mapping and alpha mapping can help to represent the geometry details and illumination details; antialiasing technique can make up the defects of losing-shape picture. Fig. 4 is an example of using bump-mapped technique on the cowling of the rocket [18]:

In Fig. 4, the area enclosed by red ellipses shows the bump mapping effect. In these areas, the detail exhibition seems true to nature, but it is the function of bump mapping instead of the 3D model. Thus, the reduction of the faces in the 3Dmodel and the anti-anamorphic visual effect successfully improved the real-time simulation impacts.

2) Numeric drive performance testing:

Numeric drive is the mathematical foundation of the whole software. Correctness and expeditiousness are the basic request.

During this testing, algebraic loop (AL) is a serious problem detected by testers. AL consists modules of direct feedthrough, it will reduce the calculational speed or interrupt the simulation.

For example, the equation of $1-x=x$, which is built up in MATLAB/SIMULINK, is shown in Fig. 5:

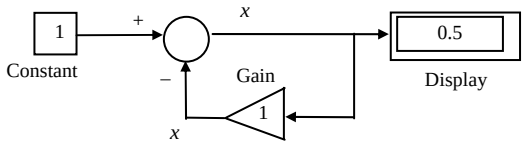

Fig. 5. An example of algebraic loop.

Several methods are given to solve this problem: firstly, to a simple system, we could work it out by human to remove AL, to the example above, we could work out $x=0.5$ and AL is removed; secondly, use optimized arithmetic, such as Newton method, to work out the AL; finally, we could add a delay module to avoid AL, but it will reduce the calculational speed.

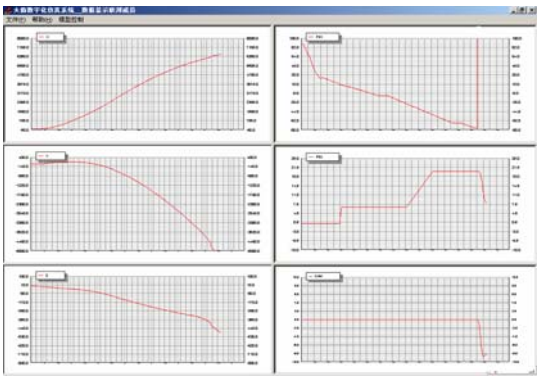

Fig. 6. Simulation result of six degrees of freedom.

Fig. 6 is the improved simulation result of the six degrees of freedom during the process of aviation after this testing:

The picture above demonstrates a much better

performance of six degrees of freedom, which includes the displacement of *X*, *Y* and *Z* direction (3 pictures on the left sides) and angle of pitching, yaw and roll (3 pictures on the right sides), compared with the former numeric models with no testing done.

5. Conclusions

In this paper, we present and analyze the applications of software testing based on the RDSS. Four basic testing methods and the combinations of them are applied during the testing process to test the specifications, software, codes and structures, etc. The testing process includes specification testing, unit testing, integration testing, system testing, and performance testing.

The concernful quality improvements are achieved by rationally using these testing methods. Nearly 1000 test cases are used to all the modules. The error rate (the number of errors / the number of test cases) after the stage of code is 12.7%, less than 22.4% of error rate in the stage of code. The running time is nearly 80% faster at an average level after the 3D models and numeric models are optimized. The reliability has also been firmly established to an average level of 98.1% after the testing, which is less than 44% before. It is difficult to detect all software faults with a single testing technique [1], thus the advantages of the combinations of the four basic testing methods is again proved to be much effective to this complicated software.

Testers are quite important factors concerned in this testing process, they play a significant role during the testing lifecycle, which could economize the precious time and reduce the charges. On the other hand, communication skills are also quite important. There is no doubt that, testing skills and experience are quite important for a tester, but it may be valuable that the active attitude and the agility of testers are more important.

References

[1] R. L. Zhao, M. R. Lyu, and Y. H. Min, "A new software testing approach based on domain analysis of specifications and programs," in *Proc. 14th International Symposium on Software Reliability Engineering*, Denver, Colorado, 2003, pp. 60-70.

[2] L.-F. Wang and K.-C. Tan, "Software testing for safety-critical applications," *IEEE Instrumentation & Measurement Magazine*, vol. 8, no. 2, pp.38-47, 2005.

[3] M. Norris and P. Rigby, *Software Engineering Explained*, Chichester, U.K.: Wiley, 1992, ch. 3.

[4] S. R. Schach, *Software Engineering*, 2nd ed. Homewood, IL: Irwin, 1993, ch. 2.

[5] S. Sigfried, *Understanding Object-Oriented Software Engineering*, New York: IEEE Press, 1996, ch. 2.

[6] I. Sommerville, *Software Engineering*, 6th ed. Reading, MA: Addision-Wesley, 2001, ch. 2.

[7] M. Weisfeld, *The Object-Oriented Thought Process*, 2nd ed. Indianapolis, IN: Sams, 2004, ch. 2.

[8] H. Younessi, *Object-Oriented Defect Management of Software*, Upper Saddle River, NJ: Prentice Hall, 2002, ch. 3.

[9] M. Ben-Menachem and G. S. Marliss, *Software Quality: Producing Practical, Consistent Software, Slaying the Software Dragon Series*, Boston, MA: International Thomson Computer Press, 1997, ch. 5.

[10] I. Burnstein, *Practical Software Testing: A Process-Oriented Approach*, New York: Springer, 2003, ch. 2.

[11]R. Patton, *Software Testing*, Beijing: Mechanical industry Press, 2002, ch. 2 (in Chinese).

[12] Y. Deng and Z.-T. He, *Batoom: A Practical Approach to Test Object-Oriented Software*, in *Proc. IEEE 1998 International Conference on Technology of Object-Oriented Languages*, Beijing, China, 1998, pp. 328-338.

[13] D. Tkach and R. Puttick, *Object Technology in Application Development*, 2nd ed. One Jacob Way Reading, M.A.: Addison-Wesley Publish Company, 1996, ch. 2.

[14] L.-Z. Jin, "Process in testing object-oriented software," *Computer Research & Development*, no. 1, pp. 23-26, Jan. 1985 (in Chinese).

[15] S. R. Chidamber and C. F. Kemerer, "A metric suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no.6, pp. 476-493, 1994.

[16] C. F. Mai, "The complexity of object modules in OO technology," *Computer Science*, vol. 23, no. 1, pp. 19-23, 1996.

[17] S. P. Ng, T. Murnane, K. Reed, *et al*., "A preliminary survey on software testing practices in australia," in *Proc. IEEE 2004 Australian Software Engineering Conference (ASWEC'04)*, Melbourne, Australia, 2004, pp. 360-370.

[18] J. Sun, L.-M. Xu, H. Li, *et al*., A practical bump mapping technique in scene simulation," in *Proc. IEEE 2005 International Conference on Intelligent Mechatronics and Automation*, Edmonton, Canada, 2005, pp. 537-542.

Xian-Yang Zhao was born in Jilin Province, China, in 1981. He is now studying for the master's degree and majoring in mechatronics engineering at school of Mechatronics Engineering, University of Electronic Science and Technology of China (UESTC). His research interests include virtual reality technology simulation and the applications of virtual manufacturing technology.

Li-Mei Xu was born in Sichuan Province, China, in 1969. She received B.S. and M.S. degree from UESTC, in 1991 and 1994, respectively, and Ph.D. degree from Nanyang Technological University, Singapore in 2001. She is currently an associated professor with UESTC. Her research interests are in the areas of structural vibration, Micro electro-mechanical system modeling and measurement.

Hui Li was born in Sichuan Province, China, in 1963. He received M.S. and Ph.D. degrees in mechatronics from the Northwest Polytechnic University Xi'an, China in 1987 and 1992, respectively. He joined UESTC in 1991 as a member of technical staff. He is currently a professor with UESTC. His current interests include information system development and industry engineering (IE).