ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 6, Issue 2, pp: (90-95), Month: April - June 2018, Available at: www.researchpublish.com

Verification and Validation of Component Based Software Systems

Seyfali Mahini

Islamic Azad University, Khoy branch, Khoy, Iran

Abstract: **One theory with component based software technology is that the software module consumers, that is the software system producers, can reduce their effort needed for, among other things, verification of their module based software systems compared to traditional custom developed systems. However, to achieve this goal, software module producers must ensure a high, documented and trusted quality of their commercial components. Otherwise the effect will be the opposite. It should be easier to put together ready-to-use software module in systems than to develop everything from basic on, but if a software component used is of poor quality, the system verification process can be very difficult, time consuming, and therefore expensive. Traditional verification and validation techniques are still important to ensure software quality, especially for software component producers. Detailed specifications of the software and thorough reviews of these specifications provide a way to confirm the functionality and quality of the components so that they can be trusted by the component consumers. So a defined software development process and corresponding development methods and tools guarantee a high software quality. For component consumers, software verification is not like a traditional confirmation. Software bugs can not be found and corrected by conventional debugging techniques because most of the source code is not available. Instead, a more research-like technique needs to be used to find bugs and workarounds to avoid them.**

Keywords: **software development, software module, verification, verification, validation, certification.**

1. INTRODUCTION

The emerging technology module-based software development places new demands on ensuring the required functionality and quality of the software. In classic or specially designed software, verification and validation could be done in close collaboration with the customer to meet the particular requirements from the customer [4]. When developing software module, or module based software systems, verification and validation becomes more complicated. Software module developed for reuse, and especially module developed for the open market, must be precisely specified and verified then most custom software for a variety of reasons. Other than those who developed them will use them. Such software will be used in many different situations and configurations, many of which were not planned at the time of development. They will not be delivered along with its source code, but as black boxes [7].

On the other hand, software systems developed with the help of components should require less verification than custom software systems; at least this is one of the premises with component-based software technology. But if the quality of the used modules is not enough, identifying of faults in a module based system can be both difficult and expensive [2]. Instead of systematically searching through the software code for faults, a more research like strategy has to be used to locate them [10]. Rather than correcting a mistake, a workaround must be developed to avoid this.

There are two strategy to ensure the quality of a software modules or software system. One strategy is to detect and eliminate errors that have occurred; this is done by means of various verification and validation activities. Another strategy is to prevent the introduction of errors; this happens through a corresponding development process. But it's not enough to guarantee functionality and quality, because for commercial components functionality and quality have to be certified as well, which is credible.

Section 2 briefly describes some traditional and some new verification and validation techniques, section 3 covers some aspects of error prevention, section 4 briefly describes some points on software certification, and section 5 discusses which verification and validation techniques are useful in the various components Software development processes.

2. VERIFICATION AND VALIDATION TECHNIQUES

The purpose of the verification is to ensure that the software modules or software systems meet their specified requirements [3]. Specified requirements are found not only in specification documents but also in functional specifications, architectural and design models, test cases and so on. Specified requirements can also be found in regulations, standards, instructions and guidelines and the like that impose restrictions on the software. The purpose of the validation is to demonstrate that a software module or system performs its intended purpose when placed in its intended environment [3]. There are two main techniques, inspections and tests, which may be used for software verification. Tests are also used for software validation [4].

The purpose of inspections and tests is to identify errors in the software. Inspections or static verification are used to review and analyze representations of the software module or software system such as specifications, models, and source code. Testing or dynamic verification and validation entail executing and examining an implementation of the software module or software system [4]. Verification and validation should be carried out continuously during the development of a software module or software system. Software specifications are decisive for verification and validation [4,6]. Specifications define the correct structure and correct behavior of the software so that incorrectness can be identified. Incorrectness is software faults called defects or bugs and can cause software failures. Software can also fail due to environmental constraints not caused by software faults.

2.1 Inspections:

A systematic software test requires a large number of tests that need to be developed, executed, and examined. This means that testing is time consuming and expensive. Inspections have proven to be a cost effective and successful method of fault detection for two reasons [4]. One reason is that many defects can be detected during an inspection session, while a test normally identifies only one or a few defects at the time. The other reason is that inspections reuse domain and implementation knowledge among the reviewers in a more efficient way than tests. Inspections can be done on different levels of formality and can either be done manually or by means of some analysis tools.

2.1.1 Manual Inspections Techniques:

Audits and assessments are formal inspection techniques based on interviews and reviews of work products such as documents, models and source code. Those responsible for quality outside responsible software development generally carry out audits and assessments and decide on what and how to control them. Audits and assessment results are documented and corrections based on these results should be re-verified. Formal reviews are usually conducted internally by the software development team itself. The work products are sent out the review team members and each reviewer report his/hers findings. The final result will be convened for a review meeting. The results are documented and corrections based on these findings should be reported [4].

Peer reviews are more informal inspections between colleagues. It is an exchange of services. Findings are usually not documented and corrections are not reported. The most extreme use of peer reviews is found in eXtreme Programming [9], which describes a pair programming strategy in which two or more colleagues work together in groups that are constantly engaged in analysis, design, coding and testing.

2.1.2 Tool-based inspections:

The relevant software inspection tool is the source code compiler. Many software bugs are detected by the compiler at a very early stage of development. Static source code analyzers are software tools that scan the source code and detect potential errors and anomalies by analyzing control flows, data usage, interfaces, information flow, and execution paths [4].

The static program analysis of executable software is done with tools that can analyze the content, structure, and logic of the software by examining of the executable file image. Binary editor tools show the binary code of the software and the corresponding ASCII representation and allows slight modifications of the code. Dis-assemblers can generate pseudo code representation of the binary code and de-compilers can even reconstruct the original source code [10].

2.2 Tests:

Testing of software module and module based software systems is possibly the single most demanding aspect of component technology [1]. When a software module is developed all coming usage of it cannot be known and by that means not tested, and when the software module is integrated into a system it should already be tested and ready for use. The ability to reuse pre-made and tested modules is the core of module technology. Software testing is the activity of executing software to control whether it matches its specifications and executes in its intended environment [6]. The fact that the software is running distinguishes test from code inspections in which the uncompiled source code is read and analyzed. Testing requires an executable file. Testing can be grouped according to the focus and level of details of the test. White box testing verifies the details of the software as it is designed and implemented. Black box testing verifies the functionality and quality of the software without regard to its implementation. Live tests validate the software behavior under operating conditions. White and black box tests are essentially done to identify faults while live tests are primarily done to demonstrate that the software works as intended during operation.

2.2.1 White Box Tests:

White-box tests called code-based testing or structure testing [6] are performed to verify that a piece of software has been implemented and is functioning as intended. Throughout structure testing all source code should be confirm, that means that every statement should be executed at least once and all paths through the code should be followed [5]. Structure testing can be done with a debugging tool, which allows evaluation and manipulation of the code during execution, and with trace logs.

2.2.2 Black Box Tests:

Black box testing called specification based testing or functional testing [6] is done to verify a software component or system's behavior without attention to how the software is realized. The black box test checks which output the software returns when it receives a certain data as input and starts in a particular state [5]. For most software, the combination of all possible inputs, states, and expected outputs is nearly infinite, making it virtually impossible to test each combination. A good approach to designing test cases for black box testing is to identify all equivalence partitions under test and then select at least one set of test data from each partition [4, 5].

Another good method is identify the operational form of the software. The operating mode of the software reflects how it is used in practice [4, 8] or in other words, the mode of operation is the distribution of use cases when the system is in normal operation [7]. The operational form is a specification of types of inputs and the probability of their incidence and it is found by tracing the usage of the software during normal operation. This means that it is quite difficult to identify the operational form for a completely new software, at least one version of the software, including trace loggers, has to be put in operation and be used for a while before enough data is collected to identify the operational form [8]. The operational form can be used to design test cases for statistical testing.

Probing can be used to observe the state of separate software modules within a module based system during execution, or postmortem. Probing is performed using tools that are typically specific to the actual operating system and distributed with it [10].

Snooping can be used to monitor the communication between two or more modules within a module based system during execution. Snooping is also performed using tools that are typically specific to the operating system [10].

Spoofing can also be used to monitor communication between modules, but in this case by interposing an observer between the modules. Several spoofing tools exist and they are not so dependent on the operating system as probing and snooping tools [10]. Spoofing can also be performed by developing specific test modules and inserting them between the components to be observed.

System level fault injectors test the system by generating errorsn in it [7]. An error injection technique is the so called Interface Propagation Analysis (IPA). IPA corrupt the states propagated through the module interfaces by means of for example a specific test module interposed between two system modules, the same technique as spoofing. The test module replaces the original state propagated between the system modules with a corrupt state during software execution.

2.2.3 Live Tests:

Statistical tests are used to investigate how the software works under operating conditions. The test cases are designed to simulate normal software usage based on the operating form and the goal is to obtain data to estimate software reliability and performance during normal system operation.

Operational tests of a software system and its integrated modules are performed during system operation. Modules should be used so that any errors leave traces that have been logged, so that errors can be detected in the operating software systems [1]. The system should be deployed in such a way that the usage of the system is traced and the operational form can be identified.

3. ERROR PREVENTION

To reduce the effort to detect software errors, the errors should not be introduced in the first place.

One way to prevent defects in the source code is to use the implementation languages and development tools appropriate for the purpose. A safe and meaningful language eliminates some of the potential for introducing errors, such as memory leaks, and allows compiler and source code analysis tools to detect errors early during implementation [1].

Another way to prevent the introduction of errors is a mature software development culture and a defined software development process. With defined software development process is meant rules, instructions, procedures, policies, good practices, checklists, patterns, templates, examples, methods and tools, etc. that guide software development. With mature software development culture means that the software development process expand as experience is achieve. For example, software inspections regiser knowledge and experience that is to be transferred back to the defined software development process [4]. This will prevent more defects from being introduced in the future.

One example of a defined software development process is the Clean room process [4]. The Clean room process uses incremental development practice in combination with a formal requirements change request method. All software is formally specified with for example state transition models, and then verified with formal inspections against these specifications. All software is formally specified with for example state transition models, and then verified with formal inspections against these specifications. The source code is written in a structured manner exactly as in the specifications with only a few valid constructions. Only statistical tests of the software, based on the operational form, are performed to confirm the system, no structure or specification testing is performed.

4. SOFTWARE CERTIFICATION

The certification has two goals: Firstly, it should establish facts about the software to be certified, and second, it should create confidence in the validity of these facts. Software certification is nothing new; It has a long history in the software industry. The general use of certification is that some authoritative organization attests to the fact that some software perform some criteria or conforms to some specification [2]. The certification of customized software systems is not that difficult, but the certification of software modules and module-based systems is a little more complicated [2]. Software modules need to be certified before going into one System and they must be certified for use in many different configurations. A module certificate says something about the quality of the modules, but nothing about the quality of the module based system it will be a part of. Even if a module vendor certifies a module, the system builder still has to certify it for their specific usage [7].

The certification of a software module or system can be done in two ways. One is to get a trusted independent organization or institute to evaluate the software to a certain standard. The result of the evaluation is then provided with the software. This approach is both time consuming and expensive, especially for modules. Another way to certify the software itself is the process industry. The verification and validation of the software could be done according to certain standards and the final verification and validation results are provided with the software. But there is no established practice for this.

One way to indirectly certify software from the trust perspective is to certify the software development process. ISO 9000 and TickIt assessments are well-known certification methods. The CMM rating (Capability Maturity Model) [3] includes another valuation method the development process, but usually it is not regarded as a certificate in the same way as ISO 9000. Whether certifying the development process will increases confidence in the software, itself depends on how dependent software quality is on the quality of the development process. If the dependency is high, the development

process can be a cost-effective alternative to software certification [2]. For example, if a software module is developed by an organization that is certified to follow the Clean Room Process completely the module will be implemented exactly as specified, that can be trusted, and the specifications can be read so also the facts about the module is available.

5. VERIFICATION & VALIDATION AND MODULE BASED SOFTWARE

All conventional verification and validation techniques are still valid in module-based software development, but the main focus depends on whether the task is to develop reusable modules or if the task is to use modules.

5.1 Verification and validation from the perspective of the Module Producer:

The production of reusable software modules for the module market requires a firmed verification and validation of the module. Implicitly then also firmed specifications is required. As a result, more effort is likely to be invested in the module software than if custom software is being developed. Module based software development activities does not differ much from custom software development activities.

Requirements must be analyzed, functionality and quality specified, and specifications reviewed with document and model checks. The architecture and detailed design has to be specified although architectural design may be limited for small modules. The design models should also be verified by

means of inspections. The code must be written, inspected and tested in a white box test to confirm that it conforms to the design. The functionality and quality of the module must be black box tested to verify compliance with the appropriate specifications. Finally, the modules should be certified and used together with their certificate and other externally interesting specifications.

5.2 Verification & Validation from the module Consumer perspective:

OTS modules (Off The Shelf) are often delivered in black boxes as executable files with licenses that prohibit decompilation into the source code [7]. Because of this, they can not be verified using white-box testing because the source code is not available. If an OTS module is not delivered with a module certificate or the certificate is untrustworthy, various black box tests must be performed to provide information useful for evaluating and selecting modules and to certify them for usage. Quality issues when using OTS modules are not just a question of the quality of each module, but also a matter of module integration compatibility.

The OTS Certification Process Module [7] proposes a combination of three verification and validation techniques to certify an OTS module for use; Black box testing based on system operating profile, system level fault injection and operational testing. After a module with the required functionality has been selected and evaluated, a black box test is performed to verify the quality of the module. If the identified defects are acceptable, the module is integrated into a system which is then tested with system level error injections to validate system behavior due to module failure. Finally, if the module appears to be acceptable to the system, a bump test is performed to confirm that the module and system behave correctly during normal system operation.

5.3 Verification & Validation of module based Software Systems:

Module based software system development activities are essentially the same as custom software development activities. Requirements must be analyzed, functionality and quality specified and specifications reviewed with document and model inspections. In addition, the requirements for functionality and quality of the required software modules must also be specified as input for the module selection and evaluation activities. The architecture and some detailed design also have to be specified. In particular, the architectural design model should be reviewed through inspections. It is likely that a code needs to be written to integrate the selected modules, and the code should be inspected and tested in a white-box test to ensure that it conforms to the architecture and detailed design. The functionality and quality of the integrated system must be black box tested to verify compliance with the appropriate specifications. Finally, the system should be certified or tested for acceptance as it will use the more common wording.

In custom software systems, errors can be found by performing white-box tests, but with module-based software systems, error identification can be much more difficult [10], especially if OTS modules are used. Instead of finding errors in the source code, the behavior of the modules must be observed and conclusions drawn from the observations. The approach to identifying errors in module-based software systems is essentially the classical scientific method of observation,

hypothesis, prediction, experiment and outcome [10]. Based on the observed result, a conclusion is drawn as to whether the hypothesis is correct or not. In order to be able to observe when the source code is not available, other means must be used to obtain visibility. Four general techniques can be used to gain insight into OTS modules: probing, snooping, spoofing, and static program analysis [10].

6. SUMMARY

The success of the module-based software technology depends on the fact that the effort required for building module based software systems can be considerably reduced compared to traditional customer specific software development. Verification and validation have always been important software development activities, and when at least the verification overhead can be reduced, much is gained. But this requires, among other things, that the modules used have the right Functionality and quality and are as well documented.

The development of software modules that should be reused and possibly also commercial requires a high focus on software quality. In mature software development organizations high quality can be ensured by means of an relevant defined development process and appropriate methods and tools, that is defect prevention. A high quality can be guaranteed by a thorough check and validation, also an error identification. The quality of the module should be certified in a, for the purchaser, trustworthy way.

There are two main techniques to identify defects in software, inspection and testing. Inspections can be performed on all software work products such as documents, models and source code. Tests can only be performed with executable code. Inspections can be more efficient and effective than tests because more errors can be found per session and the errors can be detected earlier in the development process. Specifications that define the correct structure and behavior of the software are needed to support verification and validation so that errors can be identified.

The only thing new in terms of verification and validation in software module development compared to custom software development is the proofing or certification of software functionality and quality. Verification and validation activities in the development of module-based software systems are more different, especially if the modules used are of poor quality, and verification of the software can be a very difficult task.

REFERENCES

[1] Ian Sommerville, *Software Engineering 6th Edition*, ISBN 0-201-39815-X, Addison-Wesley, 2001

[2] Jeffery M. Voas, *Certifying Off-the-Shelf Software Components*. IEEE Computer, June 1998.

[3] CMU/SEI-2000-TR-008, *Volume II: Technical Concepts of Component-Based SoftwareEngineering*, Software Engineering Institute, May 2000

[4] Kent Beck. *Extreme Programming Explained*. ISBN 0-201-61641-6, Addison-Wesley,2000

[5] Wallnau, Hissam, Seacord. *Building Systems from Commercial Components*. ISBN 0-201-70064-6, Addison-Wesley, 2001

[6] James A. Whittaker. *What is Software Testing? And Why Is It So Hard?* IEEE Software,January/February 2000

[7] Jeffery M. Voas, *Will the Real Operational Profile Please Stand Up?* IEEE Software,March/April 2000

[8] Jacobson, Booch, Rambaugh. *The Unified Software Development Process*. ISBN 0-201-57169-2, Addison-Wesley, 1999

[9] 7CMU/SEI-2000-TR-028, *CMMISM for Systems Engineering/Software Engineering,*

[10] Clemens Szyperski, *Component Software Beyond Object-Oriented Programming*, ISBN 0-201-17888-5, Addison-Wesley, 1998*Version 1.02*, Software Engineering Institute, November 2000