Verification and Validation for Safety Critical Real Time Computers

D.Thirugnana Murthy ,T.Sridevi, A.Shanmugam and P.Swaminathan
Electronics and Instrumentation Group
Indira Gandhi Centre for Atomic Research
Kalpakkam, Tamilnadu, India E-mail : dtm@igcar.gov.in

Abstract

Assurance of safety of public, occupational workers and protection of the environment are important needs to be met in the pursuance of activities for economic and social progress. These activities include the establishment and utilization of nuclear facilities and use of radioactive sources and they have to be carried out in accordance with relevant provisions in the Atomic Energy act. Increasing use of computer based system necessitated to deploy in Nuclear reactors also. Since inception of Nuclear power development in the country, maintaining high safety standards has been of prime importance. Although today computer systems are more matured, but when it comes to use in Safety Critical Systems (SCS) it forces lots of challenges. These are systems important to safety, provided to assure under anticipated operational occurrences and accident conditions, the safe shutdown of the Nuclear reactor, the heat removal from the core and containment of any radioactivity. These systems are called SCS. These systems need to have high reliability and availability. The cost and consequences of critical system failure are potentially much greater than for non-critical systems. So SCS need an augment normal analysis and testing with additional processes that are designed to produce evidence that the systems are trustworthy. This paper discusses the aspects of Verification & Validation procedure to qualify the computer system for SCS. This paper elaborates program inspections, static & dynamic analysis and V&V techniques

Key words: Safety critical system, Verification, Validation, Walkthrough, Inspection, Static and dynamic analysis, Formal verification, Clean room, V&V techniques

I. INTRODUCTION

Verification and Validation (V & V) of the software systems are checking and analysis processes that ensure that software conforms to its specification and meets the needs of the customer. V &V is a whole life cycle process. It starts with requirement reviews and continues through design reviews and code inspections to product testing. V & V procedure for Real Time Computers used for Safety Critical Systems (SCS) forces lots of challenges. Software V&V processes "determine whether development products of a given activity conform to the requirements of that activity, and whether the software satisfies its intended use". This determination may include analysis, evaluation, review, inspection, assessment, and testing of software products and processes. V&V processes assess the software in the context of the system, including the operational environment, hardware, interfacing software, operators and users.

II. SAFETY CRITICAL SYSTEMS

These are systems important to safety, provided to assure under anticipated operational occurrences and accident conditions, the safe shutdown of the Nuclear reactor (Shutdown systems), the heat removal from the core (Emergency core cooling systems) and containment of any radioactivity (Containment Isolation Systems) [1]. These systems are called SCS, which plays a principle role in achievement or maintenance of Nuclear power plant safety[2]. These systems need to have high reliability and availability.

III. VERIFICATION &VALIDATION

Verification is the process of determining whether or not the product of each phase of computer based systems development process fulfills all the requirements imposed by the previous phase. Validation is the process of testing and evaluating the integrated computer based system (hardware and software) to ensure the compliance with the functional, performance and interface requirements. Demonstration to safety committee for correctness and safety of the system requires a variety of detailed V &V activities.

IV. V &V PLANNING FOR SOFTWARE V&V

Systematic planning is needed to get the most out of V&V and to control. V & V are required for each stages produced at the end of the following development activities[3].

1.Computer system requirements

2.System Architectural design

3.Software requirements

4. Software design

5.Software coding and Implementation

6.Software unit testing

7.Hardware and Software Integrated testing

8.Acceptance Testing both at factory & site.

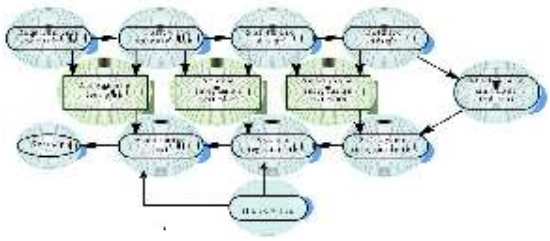

Fig. 1. V model development

Fig. 1. shows how the test plans are derived from the system specification and design [4].

V. V&V OF SOFTWARE

The V & V of the software are carried out in all phases of life cycle.

1.Concept 2.Requirements

3.Design 4.Implementation

5.Test 6.Installation

7.Operation and maintenance

It is recommended that V&V activities to be planned based on V&V standards/guides such as IEC 880[5]. The Software V&V Plan should then be evolved conforming to the selected standard e.g. IEEE std. 1012.

Software Verification

Verification should be applied to following software development phases.

Software Requirements Specification (SRS)

Software Design (architectural and detailed design)

Code (Programs)

Number of techniques and tools can be used for verification. The capabilities of different tools & techniques for verification differ in terms of aspects they cover. It is therefore essential to evolve a mix of verification techniques which will have adequate depth and which will together ensure widest possible verification coverage. Some of the techniques that should be considered for verification are walkthrough and Inspection, program analysis and formal verification techniques.

Walkthroughs and inspections

A walkthrough is the process of reviewing a development product manually, by a set of trained people. i.e somebody literally going through it. It can be formal or informal. Informal walkthroughs are very useful during the development activities but is certainly not meant for approving products, as there is no formal commitment to it and the goals are not formally defined. Walkthroughs are made effective, with a formal structure for SCS. The participants are assigned roles and evaluate the product being verified from different points of views. The important goal of a walkthrough is to cover as many errors as possible.

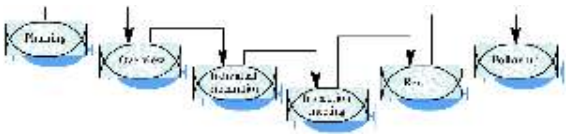

Fig. 2. Inspection process

Inspections are similar to walkthroughs except that the initiative comes from the inspection team, which studies the product and documentation independently for finding out problems and seeks clarifications from the designers. Use of checklists can be very effective if check lists are designed carefully and the responses to check list are direct, brief and not open to interpretations. The inspection process adopted for SCS is shown in the fig 2. The walkthroughs and inspection techniques are applied to almost all work products of the software development cycle for SCS.

Program analysis (static & dynamic)

Program analysis techniques fall in to two categories - Static and Dynamic. In static analysis the programs are analyzed without executing them. It results in Language, standard/constraints checking, Control flow analysis, Data flow analysis, Information flow analysis and Semantic analysis. This analysis results help to establish correspondence between programs and the designs to decode potential problems. These tools can be used to verify that 'as-built' conforms to 'as-designed'. For example call graph of a program can be generated and compared with the calls, called-by information in software design document.

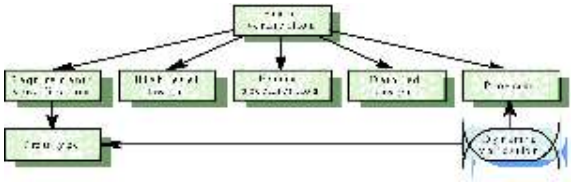

Fig. 3. Static and dynamic V & V

Dynamic analysis consists of instrumenting code and executing test cases to measure the structural coverage achieved with the test cases. Verification plan should set coverage targets so that testing can be designed to meet

those targets. Fig. 3. shows the static and dynamic V&V.

Formal verification techniques

Formal techniques are gaining acceptance in V&V activities. The effort required is high and may be at present justifiable in safety critical applications only. There are number of Formal Techniques which can be applied at different stages of software development. This is basically mathematical representation. For example software specifications could be developed using, Z or VDM, so that it would be possible to prove correctness and completeness of specifications.

The languages like ESTEREL, PROMELLA or graphical notations like STATECHARTS enable one to capture reactive behavior of the software in the 'models of the software' built using these languages & notations. These models of software can then be used to verify rigorously the required properties of the software. This technique can be used during software specification or design.

VI. SOFTWARE V&V TECHNIQUES

Software V&V tasks to fulfil the requirements of the V&V activities generally involve techniques selected from three major classes: static, dynamic, and formal analysis.

Strategies for choosing techniques

Software V&V techniques used during software requirements stage are control flow analysis, data flow analysis, algorithm analysis, and simulation. Control and data flow analysis is most applicable for real time and data driven systems. These flow analyses transform logic and data requirements text into graphic flows which are easier to analyze than the text. State transition and transaction diagrams are examples of control flow diagrams. Algorithm analysis involves re-derivation of equations or evaluation of the suitability of specific numerical techniques. Simulation is used to evaluate the interactions of large, complex systems with many hardware, user, and other interfacing software units.

V&V techniques used during software design include algorithm analysis, database analysis, sizing and timing analysis, and simulation. Algorithm analysis examines the correctness of the equations, but also examines truncation and round-off effects, numerical precision of word storage and variables (e.g., single-vs. extended precision arithmetic), and data typing influences. Database analysis is particularly useful for programs that store program logic in data parameters. A logic analysis of these data values is required to determine the effect these parameters have on program control. Sizing and timing analysis is useful for real-time programs having response time requirements and constrained memory execution space requirements.

V&V techniques used during code are control flow analysis, database analysis, regression analysis, sizing and timing analysis. For large code developments, control flow diagrams showing the hierarchy of main routines and their sub-functions are useful in understanding the flow of program control. Database analysis is performed on programs with significant data storage to ensure common data and variable regions are used consistently between all call routines. Data integrity is enforced and overflowing data tables can accidentally overwrite no data or variable. Data typing and use are consistent throughout all program elements. Regression analysis is used to revaluate software requirements and software design issues whenever any significant code change is made. This technique ensures project awareness of the original system requirements. Sizing and timing analysis is done during incremental code development and compared against predicted values. Significant deviations between actual and predicted values are a possible indication of problems or the need for additional examination.

Another area of concern is the ability of compilers to generate object code that is functionally equivalent to the source code, that is, reliance on the correctness of the language compiler to make data dependent decisions about abstract programmer coded information. For critical applications, it is solved by validating the compiler or by validating that the object code produced by the compiler is functionally equivalent to the source.

Code reading is another technique that may be used for source code verification. An expert reads through another programmer's code to detect errors. Reverse Engineering tools are used to Re-build source code from object code which in-turn in verified.

A comprehensive test management approach to recognize the differences in strategies and in objectives for unit, software integration, and software system test. Unit test verifies the design and implementation of software units. Software integration test verifies functional requirements as the software units are integrated. Special attention is focused on software, hardware, and operator interfaces. Software system test validates the entire software program against system requirements and software performance objectives. Software system tests validate that the software executes correctly within its stated operating environment. The software ability to handle with anomalies and stress conditions is emphasized. These tests are not intended to duplicate or replace the user and development groups test responsibilities, but instead supplement the development

testing to test behavior not normally tested by the user or developer.

Functional test cases execute part or all of the system to validate that the user requirement is satisfied; these test cases cannot always detect internal errors that will occur under special circumstances. Another software V&V test technique is to develop test cases that violate software requirements. This approach is effective at uncovering basic design assumption errors and unusual operational use errors. The process of planning functional test cases requires a thorough examination of the functional requirements. An analyst who carefully develops those test cases is likely to detect errors and omissions in the software requirements. In this sense test planning can be effective in detecting errors and can contribute to uncovering some errors before test execution.

Descriptions of techniques

The following are summary descriptions of the commonly used V&V techniques.

Algorithm analysis examines the logic and accuracy of the software requirements by translating algorithms into some language or structured format. The analysis involves re-deriving equations or evaluating the suitability of specific numerical techniques. It checks that algorithms are correct, appropriate, stable, and meet all accuracy, timing, and sizing.

Analytic modelling provides performance evaluation and capacity planning information on software design. It represents the program logic and processing of some kind of model and analyzes it for sufficiency.

Boundary value analysis involves tests which cover the boundaries and extremes of the function. The value zero, whether used directly or indirectly, should be used with special attention (e.g., division by zero, null matrix, zero table entry). It should also be designed to force the output to its extreme values. If the output is a sequence of data, special attention should be given to the first and last elements and to lists containing zero, one, and two elements.

Code reading involves expert reading through another programmer's code to detect errors. The individual is likely to perform a pseudo-execution (mentally) of the code to pick up errors before compilation.

Coverage analysis shows how much of the structure of unit system has been exercised by a given set of tests. System level coverage measures how many of the unit parts of the system have been called by a test set. Code coverage measures the percentage of statements, branches, or lines of code exercised by a test set.

Critical timing/flow analysis checks that the process and control timing requirements are satisfied by modelling those aspects of the software design.

Database analysis ensures that the database structure and access methods are compatible with the logical design. The data integrity is enforced and no data or variable can be accidentally overwritten by overflowing data tables and that data typing and use are consistent throughout the program..

Data flow analysis is important for designing the high level (process) architecture of applications. It can check for variables that are read before they are written, written more than once without being read, and written but never read.

Desk checking involves the examination of the software design or code by an individual, usually an expert other than the author, for obvious errors. It can include looking over the code for obvious defects, checking for correct procedure interfaces, reading the comments to develop a sense of what the code does and then comparing it to its external specifications, comparing comments to software design documentation, stepping through with input conditions contrived to "exercise" all paths including those not directly related to the external specifications, and checking for compliance with programming standards and conventions.

Error inserting (seeding) involves inserting known error types into the program and executing it with the test cases. If only some of the seeded errors are found, the test case set is not adequate. The ratio of found seeded errors to the total number of seeded errors is an estimation of the ratio of found real errors to total number of errors, or Seeded Errors Found / Seeded Errors = Real Errors Found / Total Real Errors. Then, one can estimate the number of errors remaining by subtracting the number of real errors found from the total number of real errors. If all the seeded errors are found, it indicates that either the test case set is adequate, or that the seeded errors were too easy to find.

Functional testing executes part or all of the system to validate that the user requirement is satisfied. Interface analysis is a static analysis technique. It is used to demonstrate that the interfaces of sub programs do not contain any errors that lead to failures in a particular application of the software. The types of interfaces that are analyzed include external, internal, hardware / hardware, software / software, software / hardware, and software / database. Interface testing is a dynamic analysis technique. Similar to interface analysis, except test cases are built with data that tests all interfaces.

Performance testing measures how well the software system executes according to its required response times, processor usage, and other quantified features in operation.

Prototyping helps to examine the probable results of implementing software requirements. Examination of a prototype may help to identify incomplete or incorrect software requirements and may also reveal if any software requirements will not result in desired system behavior.

Regression analysis and testing is used to re-evaluate software requirements and software design issues whenever any significant code change is made. It involves re-testing to verify that the modified software still meets its specified requirements. This analysis ensures awareness of the original system requirements. It is performed when any changes to the product are made during installation to verify that the basic software requirements and software design assumptions affecting other areas of the program have not been violated.

Requirements parsing involves examination to ensure that each software requirement is defined unambiguously by a complete set of attributes.

Reviews are meetings at which the software requirements, software design, code, or other products are presented to the user, sponsor, or other interested parties for comment and approval, often as a prerequisite for concluding a given activity of the software development process. Reviews check the adequacy of the software requirements and software design according to a set of criteria and procedures.

Simulation is used to evaluate the interactions of large, complex systems with many hardware, user, and other interfacing software units. Simulation uses an executable model to examine the behaviour of the software. Simulation is used to test operator procedures and to isolate installation problems.

Sizing and timing analysis is useful for determining that allocations for hardware and software are made appropriately for the software design architecture. It is performed during incremental code development by obtaining program sizing and execution timing values to determine if the program will satisfy processor size and performance requirements allocated to the software.

Stress testing tests the response of the system to extreme conditions to identify vulnerable points within the software, and to show that the system can withstand normal workloads.

Structural testing examines the logic of the units and may be used to support software requirements for test coverage, i.e., how much of the program has been executed.

Test certification ensures that reported test results are the actual finding of the tests. Test related tools, media, and documentation are certified to ensure maintainability and repeatability of tests. This technique is also used to show that the delivered software product is identical to the software product that was subjected to V&. It is used, particularly in critical software systems, to verify that the required tests have been executed and that the delivered software product is identical to the product subjected to software V&V.

Techniques used during the lifecycle

Criteria for a selection of techniques for any document include the amount of information available on them, their citation in standards and guidelines. The table 1. provides a mapping of the error detection techniques to software lifecycle phases.

Coding phase

Use of static analysis techniques helps to ensure that the implementation phase products (e.g., code and related documentation) are of the proper form. Static analysis involves checking that the system adheres to coding and documentation standards or conventions, and data types are correct. This analysis can be performed either manually or with automated tools. One category of static analysis techniques performed on code is complexity analysis. Complexity analysis measures the complexity of code based on specific measurements.

In-house developed “Static Analyser” of “C” programs is used to evaluate the quality attributes of the Code without executing. The following are the quailty attributes reported by the Static Analyser[6]

1.Comment to Code Ratio

2. Cyclomatic Complexity

3. “Goto” statement

4.Ternary Operator

5. Nesting Level

6. Dynamic Memory

7.Unused Functions

8. Assembly Code

9. Unused Variables

10. Un-initialsed variables

Robustness testing: The software is tested against inputs that fall outside the range of expected inputs to

make sure that the program can handle unexpected. If the program is expected to handle input streams arriving at unpredictable times, test the program for inputs arriving faster than planned. The purpose of this type of testing is to avoid catastrophic failure when most needed. The software is also tested for reversal of input signal and spikes / noise in the input signal.

Statistical testing: The program is run for many thousands of randomly selected inputs in order to derive a program reliability value. If any failure is detected during statistical testing, the program should be repaired. The most useful aspect of random testing is to locate failures could not be found during the other forms of testing.

VII. REVIEW TECHNIQUES

Many review techniques exist for V&V of software systems. The V&V Plan should specify the techniques to be used, and the time within the software development life cycle when the various techniques will be applied. These techniques include various forms of analysis, formal & informal reviews, product & process assessments and software testing.

VIII. PROCEDURES

Procedures require a description of the software review and testing strategy. The plan should describe the management of the software V&V activities. It should specify the V&V tasks, which will be carried out, including the planning assumptions for each task. It should establish the procedures and methods by which each V&V task will be performed, including the activities required to evaluate each software design output and each development activity in order to demonstrate that the system and software requirements have been met. It should establish procedures to ensure that systems in which errors are detected are appropriately analyzed, reported, corrected and reassessed. Anomaly reports should be generated and disseminated. The plan should describe V&V reporting requirements. This includes review documentation requirements, evaluation criteria, error reporting, and anomaly resolution procedures.

IX. CLEAN ROOM SOFTWARE DEVELOPMENT

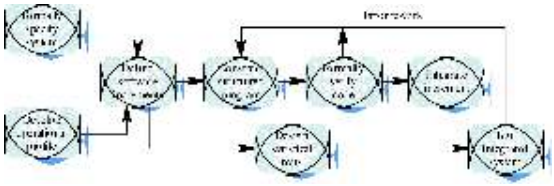

Fig 4. The clean room process

The name is derived from the semiconductor fabrication. The philosophy is defect avoidance by mathematical approach rather than defect removal as shown in fig 4. Software development process is based on incremental development, Formal specification, Static verification using correctness arguments, Statistical testing to determine reliability.

Cleanroom process teams

Three teams involved when the clean room process is used for large system development.

Specification team:Responsible for developing and maintaining the system specification

Development team:Responsible for developing and verifying the software. The software is NOT executed or even compiled during this process

Certification team:Responsible for developing a set of statistical tests to exercise the software after development. Reliability growth models used to determine the reliability.

X. SOFTWARE QUALITY ASSURANCE (SQA)

The SQA activities arise due to need to enforce conformance to selected standard and procedures in a given project. Therefore it is necessary for the software developer to identify governing standards and procedures for all software development activities and deliverables (including documentation) and then evolve a QA plan to ensure compliance[7]. The plan should address, Identification of all governing standards and procedures, Identification of authority for review and approval of each deliverable, Procedures for reporting and correcting non-conformances to standards and procedures, Identification of development, verification & configuration management plan., List of all the audits to be performed, Organizational structure and reporting lines for QA teams

The SQA plan should be available prior to start of the software development activities. A SQA Plan should be evolved based on an appropriate standard.

XI. CONCLUSION

V & V for Safety critical systems are discussed by which the high availability and reliability requirement is achieved. Verification and validation should establish confidence that the software is fit for purpose. This doesn't mean completely free of defects. Rather, it must be good enough for its intended use and the type of use will determine the degree of confidence that is needed and is the step towards Zero Deftect Programming. V&V process should be extended during operation & maintenance phase of Real Time Computer Systems. For any change in the requirement in O&M phase, impact analysis shall be carried out. The modified software shall be tested & verified in representative system before porting to target system.

Table 1. Error detection and related techniques

Techniques	Req.	Design	Coding	Test	Code	Maintenance
Algorithm analysis	✔	✔	✔	✔		✔
Boundary value analysis				✔		
Control flow analysis	✔	✔	✔			✔
Database analysis	✔	✔	✔	✔		✔
Data flow analysis	✔	✔	✔			✔
Data flow diagrams	✔					
Desk checking (code reading)			✔			
Error seeding				✔		
Formal methods	✔	✔				
Inspections	✔	✔	✔			
Interface analysis	✔	✔	✔			
Interface testing				✔		
Performance testing				✔		
Prototyping	✔	✔	✔			
Regression analysis and testing	✔	✔	✔	✔	✔	✔
Requirements parsing	✔					
Reviews	✔	✔	✔	✔	✔	✔
Simulation	✔	✔	✔	✔	✔	✔
Sizing and timing analysis		✔	✔	✔		✔
Stress testing				✔		
Tracing (traceability analys s)	✔	✔	✔	✔		
Walkthroughs	✔	✔	✔	✔	✔	✔

ACKNOWLEDGEMENT

The authors thank members of Electronics and Instrumentation Division of Indira Gandhi Centre for Atomic Research, Kalpakkam for their constant encouragement in V & V.

REFERRENCES

[1]. Design safety guide on computer based systems, Atomic Energy Regulatory Board /SG-D25

[2]. Software reliability and safety in Nuclear reactor protection systems by J. Dennis Lawrence for U.S Nuclear regulatory commission. UCRL-ID-114839

[3]. Pressman, R.S., 2001, Software Engineering, McGraw Hill, 5th Ed.,

[4]. Sommerville, I., 2001, Software Engineering, Pearson Education, 6th Ed.,

[5]. IEC 880, 1986, 'Software for computers in the Safety Systems of Nuclear Power Stations'. IEC

[6]. T. Sridevi, A. Shanmugam, D. Thirugnana Murthy, S. Ilango Sambasivan and P.Swaminathan, March 2007, Static Analyzer for Computer Based Safety Systems Journal of the Instrument Society of India (ISOI), India 37(1) 40-48.

[7]. Watts S.Humphery, Managing the software process. SEI series in software Engineering.