Journal of the Korean Nuclear Society
Volume 28, Number 6, pp. 594~601, December 1996

≪Technical Note≫

Verification of Safety Critical Software

Ki Chang Son, Chong Son Chun, Byeong Joo Lee, Soon Sung Lee, and Byung Chai Lee
Korea Atomic Energy Research Institute
150 Dukjin-dong, Yusung-gu, Taejon 305-353, Korea

(Received December 18, 1995)

Abstract

To assure quality of safety critical software, software should be developed in accordance with software development procedures and rigorous software verification and validation should be performed. Software verification is the formal act of reviewing, testing or checking, and documenting whether software components comply with the specified requirements for a particular stage of the development phase[1]. New software verification methodology was developed and was applied to the Shutdown System No. 1 and 2(SDS1, 2) for Wolsong 2, 3 and 4 nuclear power plants by Korea Atomic Energy Research Institute(KAERI) and Atomic Energy of Canada Limited(AECL) in order to satisfy new regulation requirements of Atomic Energy Control Board(AECB). Software verification methodology applied to SDS1 for Wolsong 2, 3 and 4 project will be described in this paper. Some errors were found by this methodology during the software development for SDS1 and were corrected by software designer. Outputs from Wolsong 2, 3 and 4 project have demonstrated that the use of this methodology results in a high quality, cost-effective product.

1. Introduction

The hardware platform for SDS1 for Wolsong 2, 3 and 4 was replaced with ABB(Asea Brown Boveri) Procontrol-PS system due to hardware obsolescence of SDS1 for Wolsong 1. The regulation requirements were changed compared with those of the shutdown system of Wolsong 1. The changed regulation requirements consist of diversity design for shutdown system and the use of high level language for mathematical specifications, designs and verifications among the design phases.

For these reasons, new software development methodology was developed by KAERI and AECL. The main items are as follows;

- The use of P10 Function Block Diagram(FBD) language to develop application software for SDS1
- Automatic generation of execution code from Graphical Design Definition
- Simplification of software verification process

Software verification process is closely integrated with software development process. Therefore, first of all this paper introduces software development process applied to this software verification process briefly(See Figure 1).

In the first stage, the software designer creates Software Requirements Specification(SRS) based on preliminary Hardware Design Manual(HDM) and De-

sign Input Document(DID). Functional requirements, computer input/output information, input/output transformation and performance requirements are described in the SRS[9].

In the next stage, the software designer creates Software Design Description(SDD) from SRS and self-checks identified in the HDM. The functional requirements in the SRS are duplicated in the SDD and performance, reliability, and accuracy targets are replaced in the SDD by measures to achieve these requirements[8].

In the final stage, software designer translates the graphical diagrams in the SDD into the high-level instruction set of the platform. Since the graphical notation is formal it is not necessary to go through an additional coding step. The notation is translatable directly to executable code. A data-flow editor and translator named FUP which accompanies the Procontrol-PS system is used to perform this translation.

The formal notation of the SRS combined with the systematic design process(including verification) ensures that a highly reliable system will be produced. The streamlining of process where requirements, once specified formally, are never re-specified in another notation significantly simplifies the verification process.

Software verification is performed according to the four verification phases: Requirements Review, Design Verification, Code Verification, and Testing.

These verification phases will be presented in the next section and software verification methodology applied to SDS1 for Wolsong 2,3 and 4 was developed according to "Standard for Software Engineering of Safety Critical Software"[2] which is prepared based on "IEC 880"[3] and "CAN/CSA-Q396. 1. 1-89"[1].

2. Requirements Review[4]

The objective of requirements review is to identify any ambiguity and incompleteness in the requirements specified in the Design Input Document(DID)

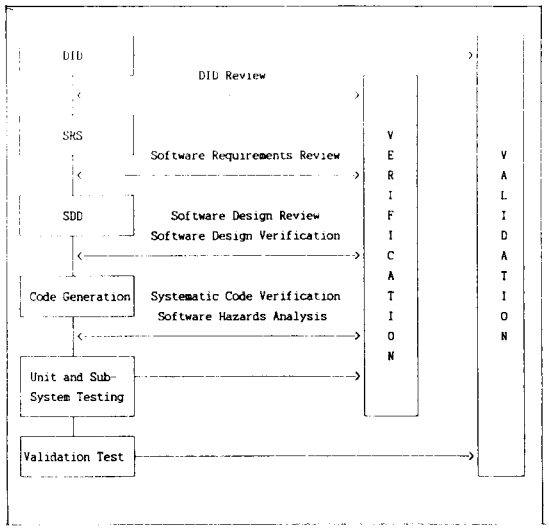

Fig. 1. Software Verification and Validation Activities Applied to SDS1 Software Development Process

[5], to verify that the Software Requirements Specification(SRS) meets the requirements of the DID that pertain to the application software, to verify the justification for including any requirements and design constraints in the SRS which were not derived from the DID.

Prior to conducting the actual review of the SRS, each reviewer should understand the DID in sufficient detail to be able to describe the required behavior of the software. By having a thorough understanding of the DID prior to reviewing the SRS, a reviewer will be less likely to be biased by the interpretation presented in the SRS.

In addition, the reviewers must also be familiar with the notation of the SRS and methods of analysing the SRS.

In the actual review process, the lead reviewer performs a comprehensive review of all topics while the reviewers from the functional design team and hardware design team review topics applicable to their areas of expertise. These sets of topics are assigned in SRS review procedure as shown on Figure 2.

The lead reviewer shall collate the findings from each reviewer and then pass these findings onto the

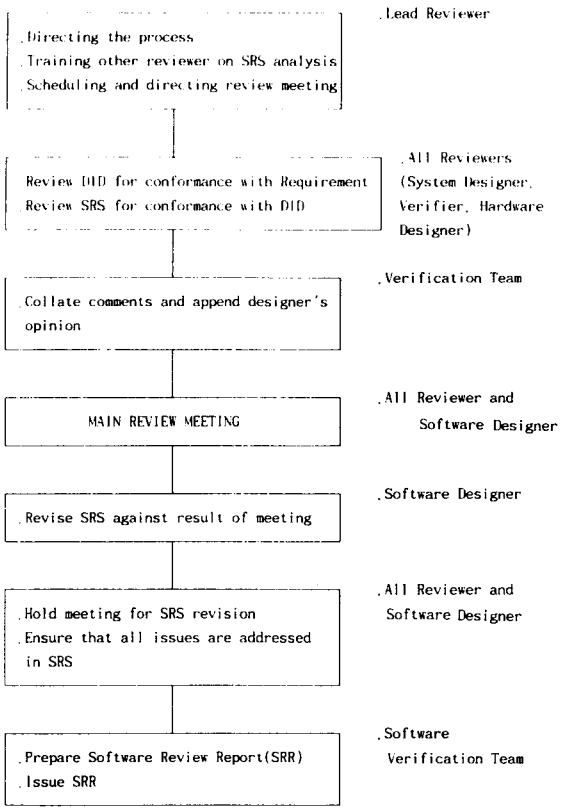

Fig. 2. SRS Review Procedure

lead software designer. The designer shall prepare a response for each issue identified. A copy of these responses shall be attended to by all reviewers. Each issue presented during the individual reviews shall be addressed in the review meeting. All issues and their resolution shall be documented in the Software Review Report(SRR)[6].

Once the SRS has been revised to incorporate each issue, the lead reviewer shall review the revised SRS to ensure that each issue has been adequately resolved.

The entire review procedure shall be followed for initial issues and major revisions. Minor revisions may be reviewed solely by the lead reviewer provided that the extent of the review is identified in the SRR.

The SRS review process is divided into the following tasks :

- diagram review,
- input/output interface review,
- requirements mapping,
- timing requirements review,
- design constraints and other requirements, and
- SRS format and index review.

3. Design Verification[7]

Design Verification consists of Software Design Review and the Systematic Design Verification.

3.1. Software Design Review

The Software Design Review ensures that in creating the Software Design Description(SDD) from the SRS, the SRS requirements other than the functional requirements have been implemented appropriately. Also, any self-checks added will be reviewed against the preliminary Hardware Design Manual(HDM) to ensure that they have been implemented properly. Also, the justification for the addition of any functionality not in the SRS or the HDM will be checked.

3.2. Systematic Design Verification

The Systematic Design Verification verifies, using mathematical verification techniques and rigorous justifications, that the functional requirements of the SRS have been transferred correctly to the SDD without the introduction of any errors during the implementation of the other SRS requirements or self-checks. The estimated computational error for each output variable is one of the principal check items in the systematic design verification stage.

3.2.1. Systematic Design Verification Process

The systematic design verification process is as follows ;

A. Map the pages of SDD logic to the corresponding SRS logic based on signal names which are

common. Identify any missing or additional logic.

B. Apply the required arithmetic conversions to constants specified in the SRS and compare with the values in the SDD.

C. Verify by inspection, or if necessary by algebraic transformations and/or truth tables that the SDD logic for each output on a page, together with the defined I/O transformations, is mathematically equivalent to the corresponding SRS logic.

D. For each output variable review the analysis of computational accuracy presented in the SDD and compare with the required accuracy stated in the SRS.

E. Compare the accuracy of all timing functions in the SDD logic with the functional timing requirements stated in the SRS.

F. Review the analysis of performance times presented in the SDD, and compare with the required performance times stated in the SRS.

G. Review the analysis of initialization behaviour presented in the SDD.

H. Review the analysis of potential race conditions presented in the SDD.

4. Code Verification[10]

Code Verification consists of Systematic Code Verification and Software Hazards Analysis.

4.1. Systematic Code Verification

The Systematic Code Verification verifies, using mathematical verification techniques and rigorous justifications, that the executable code is correctly generated from the SDD. To perform it, an assembly code listing generated from the executable code by a disassembler is manually compared against the SDD. Figure 3 shows the overview of code verification.

A disassembler is prepared and tested according to

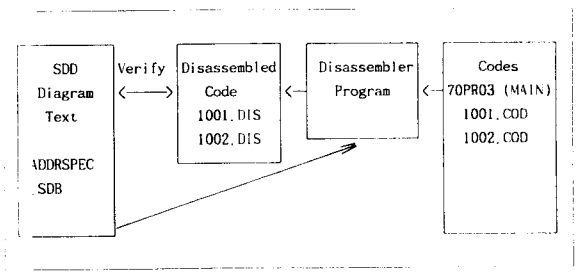

Fig. 3. Overview of Code Verification

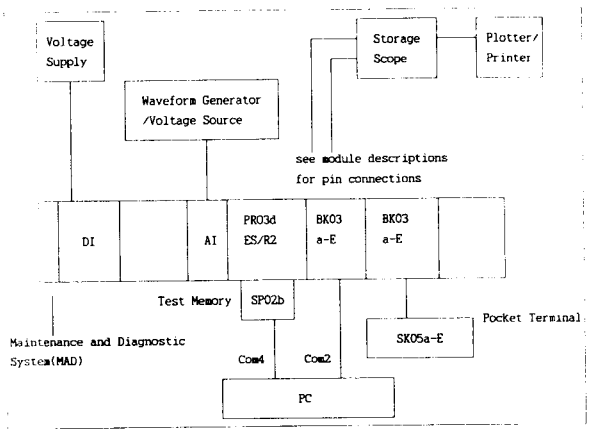

Fig. 4. Hardware Configuration for Unit and Subsystem Testing

Reference 15 to prevent a fault of code verification from a fault of a disassembler.

For safety critical applications with the Procontrol-PS system of ABB, the need to make arguments for the quality of the FUP translator of Procontrol-PS is obviated by verifying the object code against the SDD directly.

This is an one to one mapping between the primitive blocks in the graphical language and executable instructions generated.

For each primitive block with n inputs and m outputs there is exactly one instruction, n input addresses, and m output addresses generated. With the aid of a disassembler and a map file mapping physical addresses onto logical variable names, the disassembled code is visually compared with the data-flow diagrams. A check-list is given to the verifier with items to check on each drawing.

```
*  Log Power
**************
* $Header$
**************
1 * Log Power Cases
DEFINE TOLERANCE_% = 0.06 %
DEFINE HLim = 81.76 %
DEFINE LLim = 4.0 %
DEFINE LCalibLim = 10.0 %
DEFINE Hys = 1.0 %
DEFINE M = 0.07 %
**************
 TEST_TABLE
**************
* Conditions tested:
*       - rational (and greater than LCalibLim)
*       - irrational high
*       - rational and  <= LCalibLim
*       - irrational and <= LCalibLim
*       - irrational low
*
* Note: HCalib limit corresponds to irrational HLim, however the LCalib limit does not
*       correspond to the irrational LLim
*       <<LogPower>> = (50/71.76) (<LogPower> - 10.0 ) %
**************
      SET   <LogPower>     TEST   <<LogIrr_I>>  <<ValidityErr_I>>    <<LogPower>>
1           50%                      1 B            1 B              27.871 %
2           HLim - M                 1 B            1 B              49.951 %
3           HLim + M                 0 B            0 B              50 %
4           100 %                    0 B            0 B              50 %
5           199 %                    0 B            0 B              50 %
6           HLim                     0 B            0 B              50 %
7           HLim - Hys + M           0 B            0 B              49.352 %
8           HLim - Hys - M           1 B            1 B              49.254 %
9           LCalibLim + 1 %          1 B            1 B              0.697 %
10          LCalibLim - 1 %          1 B            1 B              0 %
11          LLim + M                 1 B            1 B              0 %
12          LLim - M                 0 B            0 B              0 %
13          0 %                      0 B            0 B              0 %
14          -0.006 %                 0 B            0 B              0 %
15          -50 %                    0 B            0 B              0 %
16          -199 %                   0 B            0 B              0 %
17          LLim                     0 B            0 B              0 %
18          LLim + Hys - M           0 B            0 B              0 %
19          LLim + Hys + M           1 B            1 B              0 %
20          LCalibLim - 1 %          1 B            1 B              0 %
21          LCalibLim + 1 %          1 B            1 B              0.697 %
END_TABLE
```

Fig. 5. The Example of Test Case for Unit and Subsystem Testing

4.2. Software Hazards Analysis[11]

The Software Hazards Analysis determines failure modes and sequences of inputs that can lead to an unsafe state in the computer system. Based on this analysis, changes or additional self-checks may be recommended.

Preliminary Hazards Analysis(PHA) is the first step described in the procedure and provides the analyst and the reviewer with the rationale for the identification of the software related hazards. A functional Failure Mode and Effects Analysis(FMEA) has been utilized for the PHA to identify the consequences of failure for each shutdown system function. Following the completion of the PHA, the System Fault Tree was created to identify hardware, software and human failure modes in the integrated system. The interface between the system and the software is defined in this fault tree. The detailed Software Fault Trees were created using the high level source code as described in the SDD.

The Software Fault Trees are qualitative, and the probability of any of the identified failure modes are not quantified. During the process of constructing and analyzing the software fault trees, possible improvements to the structure of the code were identified. Recommendations were made to improve the fault tolerance of the software. These suggestions were discussed with the designers and review team, and where benefit was identified the changes were implemented. The Fault Tree Analysis(FTA) technique enforces rigor and structure to the software analysis, and provides reviewable documentation which details the analysis procedure.

5. Testing

Three phases of testing shall be required for the application software: Unit and Subsystem Testing, *Validation Testing, and Reliability Testing.* The Validation Testing and Reliability Testing are performed at the Validation stage for SDS1 application software.

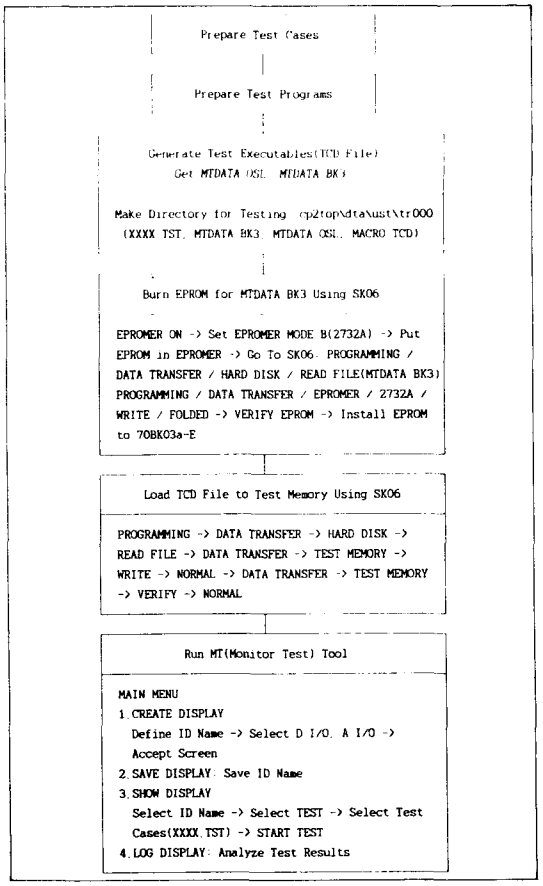

Fig. 6. Test Procedures for Unit and Subsystem Testing

5.1. Unit and Subsystem Tests

The Unit and Subsystem Tests will test the code against the graphical data-flow notation of the SRS. Tests will be performed against macros, drawings, and combinations of drawings in the SRS. Features that will be tested include testing each instruction, testing each data-flow path, and testing on and around each boundary. Any self-checks identified in the SRS and preliminary Hardware Design Manual(HDM) will also be tested where possible[13]. Unit and Subsystem Tests will be conducted on the PDC hardware using input signals simulated on the local bus by the Monitor/Test software running on the development computer. Figure 4 shows the hardware configuration for unit and subsystem testing.

The bus coupler modules (BK03a-E) are added to the Maintenance and Diagnostic System(MAD) to facilitate reading from bus, internal and parameter addresses and writing to bus addresses.

To perform timing tests, a 48 VDC voltage source will be connected to the appropriate digital input module to simulate a binary input, and a waveform generator(capable of providing waveforms and manually selected voltages from 0 to 5 VDC) will be connected to the appropriate analog input(AI) module. The digital storage scope will be used to measure time differences between input and output signals. The appropriate connection points can be found in the module descriptions.

The digital storage scope shall meet the following minimum specifications :

- 2 channels
- 10^6 samples/second
- voltage resolution 5 mV/div
- time base range 10 ms/div to 1 s/div

Figure 5 shows the example of Test Case for Unit and Subsystem Testing. Test cases shall be identified based on each path from a set of monitored variables to a controlled variable in the SRS, each design feature in the SDD which is not fully specified in the SRS. For each input variable which represents an analog input, the boundaries between different input domains shall be identified. The set of test cases shall include cases in which these variables are set to values at the boundaries of each domain[14].

Each test executable is created on the PC and then transferred to the test memory(SP02b) which is attached to the processor(PR03d-ES/R2). The detailed procedures for loading an executable are described in Figure 6.

6. Conclusions

Software verification methodology applied to SDS1 for Wolsong 2,3 and 4 was described in this paper.

The review of the SRS was highly successful. Software verifiers, system designers, safety analysts, hardware designers and software designers all participated in the review. The SRS requirements correctly and unambiguously described all the requirements of the DID which pertain to the application software.

The software design was prepared by the software designer and then reviewed by a team of software verifiers and hardware designers. A few changes to the software design were made at this stage. After the corrections were made, the design verification stage detected no error.

The executable code verification was performed on the whole application and no errors were found except for a minor discrepancy in the self-check code. A few changes to the executable code were made by software designer. The verification was later repeated and no errors were found. For future applications, consideration should therefore be given to eliminate the systematic check of every line of code for code verification.

Unit and subsystem testing were executed in accordance with the relevant test procedures. No errors were found in the code files, so no testing iterations were required. The system was finally submitted to the validation team for black box system testing.

Outputs from Wolsong 2,3 and 4 project have demonstrated that the use of this methodology results in a high quality, cost-effective product.

References

1. CAN/CSA-Q396. 1. 1-89, "Quality Assurance Program for the Development of Software Used in Critical Application"
2. AECL/Ontario Hydro, "Standard for Software Engineering of Safety Critical Software", Dec. (1990)
3. IEC 880, "Software for Computers in Safety Systems of Nuclear Power Station", (1986)
4. "Procedure for the Specification of Software Req-

uirements Using the Integrated Approach", 00-68000-SWP-012, Rev. 01, Jul. (1993)
5. "SDS1 PDC Functional Specification", 86-68200-PFS-000, Rev. 02, Apr. (1994)
6. "Requirements Review SDS1 Programmable Digital Comparators", 86-68250-SRR-001, Rev. 01, Sep. (1994)
7. "Procedure for Software Design Using the Integrated Approach", 00-68000-SWP-013, Rev. 00, Aug. (1993)
8. "Software Design Description for SDS1 PDC", 86-68250-SDD-001, Rev. 01, Jun. (1994)
9. "Software Requirements Specification for SDS1 PDC", 86-68250-SRS-001, Rev. 01, May (1994)
10. "Procedure for Systematic Code Verification Using the Integrated Approach", 00-68000-SWP-016, Rev. 00, Jan. (1993)
11. "Procedure for Software Hazards Analysis of Safety Critical Software", 00-68000-SWP-006, Rev. 00, Oct. (1993)
12. "Method for Unit and Subsystem Testing Using the Integrated Approach", 00-68000-SWP-017, Rev. 00, Nov. (1994)
13. "SDS1 Part7-PDC Hardware", 86-68200-DM-007, Rev. 00, Nov. (1994)
14. Glenford J. Myers, "The Art of Software Testing", John Wiley & Sons, (1979)
15. "P10 DISASSEMBLER software user's and design manual", 00-68000-MAN-007, Rev. 0, Feb. (1994)