Design and Implementation for Safety Critical Software Systems

Ehab Shafei*, Ibrahim F. Moawad# and Mostafa Aref+

Abstract: In recent years software has become a key element of safety critical systems. The use of Safety Critical Software Systems (SCSSs) has been increased in many critical systems such as Nuclear Power Plants (NPPs), radiotherapy, aircrafts, and many medical devices. These systems rely on the use of safety critical software in controlling and monitoring critical devices. SCSSs are developed in accordance with a set of guidelines and standards produced by the industry, or imposed by a regulator. Despite of the vital role of SCSSs in saving human life, the environment, and properties, there is no generic methodology for developing such systems based on standards and guidelines. This methodology sets an integrated model that concerned about the safety of critical software systems as a component of the critical systems. It consists of the needed processes required for developing SCSSs free of faults. This methodology ensures that SCSSs are developed using processes based on appropriate standards and guidelines and can be certified accordingly. The objective of this methodology is to produce certified critical software systems that conform to standards and guidelines. The methodology consists of three phases (safety planning and requirements phase, safety analysis phase, and design, implementation, and operation phase). This paper is going to focus on design and implementation phase. The insulin pump system is applied as a case study on the design and implementation phase.

KyeWords: Safety Critical Software Systems, Safety Analysis, Insulin Pump System

1. INTRODUCTION

The importance of the use of critical software systems in being an important factor in assuring the safe operation of many critical systems. Safety must always be considered throughout the

*Assistant Lecturer, Operation Safety and Human Factors Department, Nuclear and Radiological Regulatory Authority/ ehab_vip@yahoo.com.

#Associate Professor, Information Systems Department, Faculty of computer and information sciences, Ain Shams University/ ibrahim_moawad@cis.asu.edu.eg.

+Professor, Computer Science Department, Faculty of computer and information sciences, Ain Shams University/ aref_99@yahoo.com.

overall critical systems not limited to software, but extended to include computer hardware, electronics/electrical hardware, and operators.

Certification of SCSSs is the hot issue in many industries. SCSSs must be certified before installation and operation phase. Certification ensures that the system will not fail consequently will not cause harm to human beings or the environment and if it fails, it will fail safely. Considering existing models for developing SCSSs, most of existing methodology did not follow standards, guidelines and related documents for developing SCSSs. The methodology consists of three phases: Phase1, safety planning and requirements which are discussed in [1] that consists of four processes (describe the critical system, identify critical system functions, determine the SCSSs safety plan, and identify the functions of SCSSs). Phase2, safety analysis which is discussed in [2] that consists of (analyze and identify the hazards, apply risk management process, specify safety requirements of critical software systems). Phase3, design and implementation that consists of (design and implement SCSSs, verify and validate SCSSs, certify SCSSs, operate, and maintain SCSSs). This paper is going to focus on design and implementation phase.

Literatures showed that safety issue should be considered in the whole critical software systems and from the beginning. Already there are three methodologies for modeling software safety in safety critical computing systems. Some of them share common processes and differ in other processes. Also the implementation and the sequence of such common processes are different. Although of already exciting models follow standards in developing some process and neglect standards in developing other processes, beside they did not based on standards from early process. The first methodology [3] is a primitive one, it's based on four essential processes for developing safety software. These processes are software safety planning, safety critical computer system function identification, software and computing system hazard analyses, and finally validation and verification. The second methodology [4] is a more complex model with more seven processes than the first one. The new processes are software safety requirements analysis, software safety architecture design analysis, software safety detailed design analysis, software safety code analysis, software safety test analysis, software safety evaluation, and software safety process review and documentation. The third methodology [5] nearly is the same process as the second methodology. These models neglecting describing the critical system in which the software is a subcomponent of the whole system. They are missing the importance of developing critical software systems, according to standards and guidelines. They did not give attention to the process of software certification. The missing processes could produce deficient critical software systems which might cause failures during the operation. For these reasons the proposed methodology for developing SCSSs based on standards and guidelines is presented to overcome the weak points of exciting models.

The paper is organized as follows: section 2 describes SCSSs. Section 3 presents the methodology for SCSSs design and implementation for developing certified SCSSs based on standards and guidelines. Section 4 presents a case study performed according to our methodology. The last section concludes the discussion, and explores trends for future research work.

2. SAFETY CRITICAL SOFTWARE SYSTEMS

SCSSs can be defined as software that monitors, exercises direct command and controls over the condition or state of hardware components. And if not performed, performed out-of-sequence, or performed incorrectly could result in improper control functions, which could cause a hazard or

allow a hazardous condition to exist. Software systems are considered as safety critical if it perform one of the following functions [6]:

1) Implement a critical decision making process.
2) Control or monitor safety critical functions of software or hardware.
3) Cause or contribute to hazards.
4) Intervene when an unsafe condition is present or imminent.
5) Execute on the same target system as safety critical software.
6) Mitigate damage if the risk occurs.

3. DESIGN, and IMPLEMENTATION for SAFETY CRITICAL SOFTWARE SYSTEMS

The proposed methodology describes the development of certified SCSSs based on standards and guidelines. The methodology consists of three phases (safety planning and requirements phase, safety analysis phase, and design and implementation, phase) as shown in Figure 1. The design and implementation phase consists of four processes as shown in Figure 2. It starts from design and implementation, verification and validation, certification and finally the process of operation and maintenance.

A. Design and Implement Safety Critical Software Systems

This step is aiming to design and implement the SCSSs. The outputs of the second phase (safety analysis) which is discussed in [2] are considered to be the inputs to the design phase. These inputs are system hazard analysis, risk analysis, safety concept, software safety requirements, and safety design recommendations. In this phase also, the safety components and functions are identified for implementation, and should be analyzed to assure that the design satisfies the safety requirements and objectives.

According to [7], and DO-178B [8], coding standards should be employed and must specify good programming practice, prohibit unsafe language features, and specify procedures for source code documentation including:

1) Legal entity 2) Description 3) Inputs and outputs 4) Configuration management history

In addition to the software code must be:

1) Readable, understandable, and testable.
2) Able to satisfy the specified requirements.
3) Reviewed.
4) Tested as specified during software design.

The software design description (SDD) depicting the logical structure, information flow, logical processing steps, and data structures are defined and documented. The criteria of SDD are to assure that 1) Application software related requirements are implemented in the design,

2) All design elements are traceable to the requirements,
3) The design is correct, consistent, clearly presented, and feasible.

The SDD should contain the information listed below [9]:

1) A description of the major safety components of the software design as they relate to the application software requirements and any interactions with non-safety components.
2) A technical description of the software with respect to control flow, control logic, mathematical model, and data structure and integrity.

3) A description of the allowable or prescribed ranges of inputs and outputs.

4) A description of error handling strategy and use of interrupt protocols.

5) The design described in a manner suitable for translating into computer codes.

According to standard [10] Safety Detailed Design Analysis (SDDA) should be performed and documented. The software detailed design process develops the low-level design for the software units that will implement the software requirements. In addition, high-level design for component level of SCSSs, including the safety design features and methods, should be developed into a low level unit design. The detailed design and SDDA should examine the software requirements specification, test procedures, and the ongoing detailed design to [10]:

a. Refine the identification of Safety-Critical Computer Software Components (SCCSCs) to the unit level software components that implement the software safety requirements identified by the SSRA. Those unit-level software components that are found to affect the output of SCCSCs should also be identified as SCCSCs.

b. Ensure the correctness and completeness of the detailed design as related to the software safety requirements, architectural design, and safety-related design recommendations.

The outputs from this phase include identifying safety critical unit level software component and functions, implementation of the designed system after integrating the safety features, and unit testing phase.

The software implementation translates the detailed design into code in the selected programming language. The code should implement the safety design features and methods developed during the design process.

B. Verify and Validate Safety Critical Software Systems

Software verification is a procedure to decide if the development of software phases implemented according to requirements. While software validation is to determine that the correct requirements are full field. Validation activities assure that each software requirement is clear, unambiguous, correct, complete, and testable [11].

For verification activities, methods for verification include [7]:

1) Software analysis at the level of logic, data, and interface.
2) Inspections include software review.

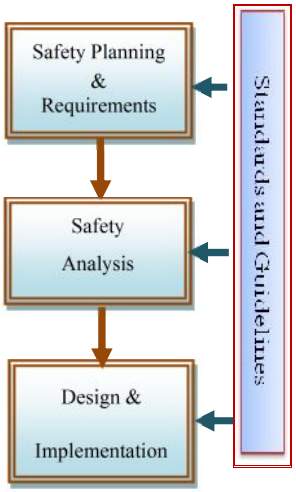

Fig.1 SCSSs development methodology

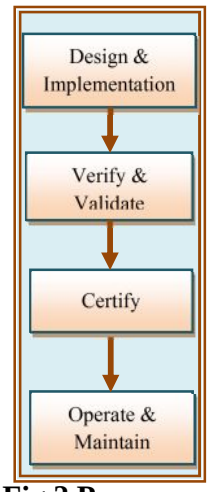

Fig.2 Processes of Design and implementation Phase

3) Testing should be performed at the level of the unit, interface, software integration, software/hardware integration, and system.

When applying the test cases we should consider that test cases covered all requirements, including full ranges values and out of range values for input signals, related trimming, all modes of operation, recovery after power supply failure, and finally stress and load test. According to [7] (Clause 7.7), testing process must be the primary method of validation with analysis used only to supplement. Testing process must be the primary method of validation with analysis used only to supplement process should be conformed to IEEE 829-1998 [12] standard for software test documentation, and IEEE 1008-1993 [13] standard for software unit testing.

All tools used in the validation must be calibrated and an approved quality system must be in place. If validation is done separately from the software, the validation must follow the software safety validation plan. For each safety function, the validation effort should document:

1) A record of the validation activities
2) The version of the software safety validation plan
3) The safety function being validated with reference to planned test
4) Test environment (tools and equipment)

In general verification and validation (V&V) activities that are carried out during the development of software phases include [14]:

1) User requirements analysis traceability
2) Design and code reviews
3) Software fault analysis
4) Documented test programs
5) Test planning and execution
6) Audits and assessment

According to [7] (Clause 7.9) verification process must be performed on:

1) Software safety requirements, 2) Software system design, 3) Software module design, 5) Software source code, 5) Data, 6) Software module testing , 7) Software integration testing, 8) Hardware integration testing 9) Software safety requirements testing (software validation).

Although it is recommended to carry out independent V&V by independent organizations to assure that the safety requirements are met and implemented according to defined requirements. This process conforms to IEEE 1012-1986 [15] for software V&V plans, IEEE/EIA 12207.0 [16], IEEE 1012-1998 [17] for software V&V, ANS 10.4 [18]. In addition to conformance with PRD-5092 [19] which identifies the requirements and responsibilities for controlling the quality of computer software, section 4.1.2 describes the requirements for V&V of computer software. A graded approach is allowed based on the complexity of the software, the degree of standardization, similarity with previously proved software, and importance to safety.

C. Certify Safety Critical Software Systems

Certification can be defined as “the process of issuing a certificate to indicate conformance with a standard, a set of guidelines, or some similar document.” [20]. Certification in critical software systems is used to certify product, process, or personnel. Product and process certification are the most challenging in developing SCSSs for critical systems such as NPPs, radiation therapy, medical devices, flight control, etc. [21]. These critical systems may cause significant damage or loss of life, if not operating properly. To avoid catastrophic damage or disaster that may caused

by failure in the safety critical software system, and to protect the society, the governments and engineering organization established standards and guidelines for developing SCSSs for different critical systems application area.
Standards and guidelines define the rules and constraints for each process. The developed SCSSs should be checked for compliance with standards and guidelines based on generic standards with the focus on the specific needs of different application area exist. Software certification conforms to the guidelines in [22], and [23]. In the industrial area, [7] has realized the need to certify the software included in their products. In fact, it has been used as the foundation for several domain-specific standards: IEC 61513 [24] for nuclear domain, [25] for automotive, IEC 62304 [26] for medical field, CENELEC EN 50128 [27] for railway and IEC 61511 [28] in the process.

D. Operate, and Maintain Safety Critical Software Systems

Operation and maintenance of SCSSs are very important processes. The operation test should be carried out to ensure that the critical software has been installed and operated correctly and to conform the correct functioning of the system. System during operation and maintenance may be subjected to perform the following [10]:

1) Operation test,
2) Periodic tests to verify that the system not degrading,
3) System modification to enhance or change functionality or to correct defects,
4) System test after hardware component replacement,
5) Adaptation for hardware changes,
6) Adding or removing features and capabilities,
7) Compensating for changes in other software components.

The installation and operation test cases must be used to confirm proper functioning of the system in the facility environment. The generated records and reports which describe the results of operation should be checked.
According to [7] (Clauses 7.6 and 7.8) the modification process starts with an analysis on the impact of the proposed software modification on functional safety. In addition to [10] state number of activities should be applied in case of modification such as: performing a hazard analysis, updating the software safety requirements, identifying new safety critical software components, updating the specification, design, and operator documentation for critical software, updating and adding comments for safety critical code, and testing the software with its components. Testing includes regression testing to verify correct implementation of related software safety requirements.

4. CASE STUDY: AN INSULIN PUMP CONTROL SYSTEM

The insulin pump is a medical system that mimics the operation of the pancreas. It's used by people who cannot make their insulin and suffering from diabetes. It helps to achieve rapid, precise, and control varying glycemic. The critical insulin delivery system is used to monitor the blood sugar levels and deliver an appropriate dose of insulin when required. Moderninsulin pumps depend increasingly on embedded control software. Software is responsible for safety functions such as sensing blood data, providing display output, dosage control, and mitigating certain hazards through alarms and alerts. To develop an insulin pump system with a high assurance of its reliability and safety, we should follow IEC 61508 [29]. The insulin pump system includes eleven components as shown in Figure 3. The entire system is controlled by the controller component. The measuring time and compared blood sugar level data are stored in the

data stores component. These data are used to analyze the diabetic's health condition. The insulin is stored in the reservoir component and pumped from the reservoir to needle assembly by using the pump component. The insulin devise communicates to the doctor's computer through communication component to download system setup parameters and upload the data stored. Alarm component using to alert the user and set status to warning.

The insulin dose computed according to sugar levels (unsafe, safe, and undesirable). The computed dose of insulin injection depends on cumulative reading at interval times not on the absolute level of glucose. There are four scenarios for injection 1) level of sugar in safe band. 2) The level of sugar is failing. 3) The level of sugar is stable. 4) Level of sugar is increasing.

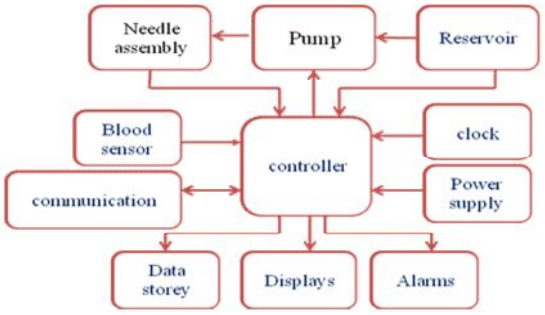

Fig.3Insulin Pump System Structure

The embedded critical control software is responsible for reading blood parameters, measuring level of blood sugar at periodic intervals, comparing between the consecutive readings, computing required insulin dose, actuating insulin pump signal, issuing alerts to the user, coordinate the functions of the various components within the pump to ensure safe operation of the pump.

The approach of implementation is aiming to produce the insulin pump software was to emulate the hardware organization by producing separate software objects (classes) for each distinguishable hardware object. These objects such as controller, display, clock, sensor, power supply, insulin pump, and insulin reservoir.

For insulin dose computing algorithm and when the insulin pump is in automatic mode, the software periodically determines (using the blood sugar level readings) the dose of insulin that should be administered to the user. This is the functionality provided by the insulin pump software that ensures the system is considered to be a safety critical system. Therefore, it is advantageous to more closely examine the algorithm that performs this task and produce a safety argument, in order to determine it is adequately safe.

Before the blood sugar level is read in, a safety check is performed to determine that the system status is correct and that the mode is set to Auto:

```
if (manDeliveryButPressed == false)
{
If (status != ERROR &&remainingInsulin>=
maxSingleDose&&cumulativeDose<maxDailyDose)
{
```

If these requirements are met then the blood sugar level can be read in:

```
reading2 = Sensor.getReading();
```

Once the blood sugar level has been read in, the system must determine whether this level represents that the sugar level is too low, at a safe level or is too high. In order to do this the current reading is checked against the safe minimum and safe maximum blood levels defined.
Check if sugar level is low:

if (reading2 <safeMin)
if sugar level is safe:
else if (reading2 >= safeMin && reading2 <= safeMax)
or if sugar level is too high:
else if (reading2 >safeMax)

Depending on the outcome of these tests, one of three execution paths is taken such as:i.e.Inthe case of sugar level is low the execution path will be:

```
compDose = 0;
alarm_ON = true;
status = WARNING;
pumpTimer.displayOutput("Sugar Low");
```

To maintain the safety of the algorithm and once the dose has been computed, the system must determine whether the dose calculated is safe. This is essential due to the critical nature of the software. In an attempt to prove to a satisfactory level that the insulin pump software is safe we can use the safety argument method of validation, in order to make certain that the insulin pump system cannot deliver an unsafe quantity of insulin. Therefore the section of code we must produce a safety argument for is the following:

```
//IF STATEMENT: 1
//If max daily dose will be exceeded by the dose calculated
if ((compDose + cumulativeDose) >maxDailyDose)
{
//Alert the user and set status to Warning
alarm_ON = true;
status = WARNING;
dose = maxDailyDose - cumulativeDose;
InsulinPump.deliverInsulin(dose);
pumpTimer.displayOutput("Close to Max Daily Dose");
}
//Normal situation
else if ((compDose + cumulativeDose) <maxDailyDose)
{
//IF STATEMENT: 2
if (compDose<= maxSingleDose)
{
dose = compDose;
InsulinPump.deliverInsulin(dose);
}
//If the single dose computed is too high
else
{
dose = maxSingleDose;
```

```
InsulinPump.deliverInsulin(dose);
}
}
```

There are two pre-conditions that would make the system unsafe:

1) If the dose calculated to be delivered is greater than the maximum single dose value.
2) If the maximum daily dose was exceeded.

Using the safety argument we must prove that the all the if-statement post conditions contradict the unsafe pre-condition [30].

The safety argument shows that the potentially unsafe state could be reached and potentially a user could be administered the maximum daily amount of insulin in one go.

The insulin pump system Java code can be found in [31].

The test of insulin pump software is depending on the following:

1) Simulators for the sensor and the insulin delivery components.
2) Test for normal operation using an operational profile. Can be constructed using data gathered from existing diabetics.
3) Testing has to include situations where the rate of change of glucose is very fast and very slow.
4) Test for exceptions using the simulator.

5. CONCLUSION

In this paper design and implementation phase of our methodology for developing certified SCSSs is presented. This methodology, which is commonly used for developing certified SCSSs consisting of three phases. This paper discussed deeply the third phase and related processes. These processes inspired on many standards, guidelines, and related documents for developing software systems generally and SCSSs specifically. This phase of methodology relies on a generic and domain specific standards. The proposed design and implementation phase of methodology integrates and complements these standards. This phase of the methodology is applied to the insulin pump system. The main advantage and contribution of this methodology is that the proposed methodology helps the practitioners to develop certified SCSSs beside other advantages such as overcoming the weak points of the existing methodologies for developing SCSSs. This paper represents a milestone for developing certified critical software. It sheds the light on the importance of embedding standards and guidelines in the analysis, design, implementation and testing processes of SCSSs. Our future work is dedicated to applying this methodology for developing NPPs system.

6. REFERENCES

[1] Ehab Shafei, Ibrahim F. Moawad, HanySallam,ZakiTaha, MostafaAref, "A Methodology for Safety Critical Software Systems Planning", *in the proceeding of 7 th WSEAS European computing conference (ECC'13), Dubrovnik, Croatia*, pp. 173-178, 2013.

[2] Ehab Shafei, Hany Sallam, Mostafa Aref, "Safety Analysis for Safety Critical Software Systems ", The Sixth International Conference on Intelligent computing and Information systems ICICIS, Egypt,2013.

[3] Daniel P. Murray, Terry L. Hardy, ''Development Safety Critical Software Requirements for Commercial Reusable Launch Vehicles'', *National Technical Information Science*, 2009.

[*4*] S.Phani Kumar, P. SeethaRamaiah, V.Khanaa, ''Identification of Software Safety Metrics for Building Safer Software based Critical Computing Systems'', *International Journal of*

Computer Science and Application, ISSN 0974-0767, 2010.
[5] SrinivasAcharyulu, P.V., P. seethe ramaiah, ‘‘A Methodological Framework for Software Safety in Safety Critical Computer Systems’’, *Journal of Computer Science* 8 (9), ISSN 1549-3636, Science Publications, 2012, pp. 1564-1575.
[6] NASA STD-8719.13. NASA software safety guidebook: NASA technical standard. Department of Defense, NASA-GB-8719.13A, 2004.
[7] IEC 61508, International Standard, Functional Safety of Electrical /Electronic / Programmable Electronic Safety- Related Systems, 1998.
[8] RTCA. DO-178B: Software Considerations in Airborne Systems and Equipment Certification, December, 1st 1992.
[9] QSD 2004-87, Assessment of Safety System Software and Firmware at Idaho National Engineering and Environmental Laboratory, 2004.
[10] NASA Software Safety Guidebook, NASA Technical Standard, NASA-GB-8719.13A, 2004.
[11] Daniel P. Murray, Terry L. Hardy, ‘‘Development Safety Critical Software Requirements for Commercial Reusable Launch Vehicles’’, National Technical Information Science, 2009.
[12] IEEE 829-1998, IEEE Standard for Software Test Documentation.
[13] IEEE 1008-1993, IEEE Standard for Software Unit Testing.
[14] AnkurPandit, ‘’A Framework-Based Approach for Reliability & Quality Assurance of Safety Critical Software’’, International Journal on Computer Science and Engineering, Vol. 02, No. 09, Pp. 2874-2879, 2010.
[15] IEEE 1012-1986, IEEE Standard for Software Verification and Validation Plans.
[16] IEEE/EIA 12207.0-1996, IEEE/EIA Standard- Industry Implementation of ISO/IEC 12207:1995, Standard for Information Technology- Software life cycle process.
[17] IEEE 1012-1998, IEEE Standard for Software Verification and Validation.
[18] ANS 10.4-1987, "Guidelines for the Verification and Validation of Scientific and Engineering Computer programs for the Nuclear Industry," 1987.
[19] RD-5092, "Software Quality Assurance," Rev. 4, November 2002.
[20] Neil Storey, Safety-Critical Computer Systems. Addison-Wesley, 1996.
[21] Andrew Kornecki, Janusz Zalewski, “Certification of software for real-time safety-critical systems: state of the art”, Innovations in Systems and Software Engineering, Springer, Vol. 5, Iss. 2, pp 149-161, 2009.
[22] RTCA. DO-178B: Software Considerations in Airborne Systems and Equipment Certification, December, 1st 1992.
[23] D. Alberico, J. Bozarth, M. Brown, et. Al. JSSC Software System Safety Handbook; A Technical and Managerial Team Approach December 1999.
[24] BS IEC 61513, International standard for Nuclear power plants. Instrumentation and control for systems important to safety. General requirements for systems, 2011.
[25] International Organisation for Standardization. ISO 26262: Road Vehicles-Functional Safety, Draft International Standard (DIS), 2009.
[26] IEC 62304, International standard for medical device software, software lifecycle processes, 2006.
[27] CENELEC EN 50128, European standard for Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems,

2001.
[28] IEC 61511, International standard for Functional safety –Safety instrumented systems for the process industry sector, 2003.
[29] IEC 61508, International Standard, Functional Safety of Electrical /Electronic / Programmable Electronic Safety- Related Systems, 1998.
[30] I. Sommerville. Software Engineering 7. Addison Wesley, 2004.
[31] www.lancs.ac.uk/ug/mccarthb/insulinpump.zip