A Framework for Hazard Analysis of Safety-Critical Computer Controlled Systems

Kadupukotla Satish Kumar
Dept of Computer Science and Engineering
JNTU Kakinada
satishkmca@gmail.com

Panchumarthy Seetha Ramaiah
Dept of Computer Science and Systems Engineering
Andhra University, Visakhapatnam
psrama@gmail.com

Abstract
Safety-Critical Computer Controlled Systems (SCCCS) are those systems whose failure could result in loss of life, significant property damage, or damage to the environment. There are many well known examples in application areas such as medical devices, aircraft flight control, weapons, and nuclear systems. These systems consist of a set of functional elements, controlled by embedded processors that together achieve a common objective. Most software-related accidents occurred in SCCCS by executing flawed programming and instructions. The research literature currently lacks an appropriate safety analysis and is fragmented among activities. The aim of this paper is to reduce the probability of unsafe system conditions using a variety of management, organization and technical measures. The approach begins with conducting three different types of hazard analysis techniques to SCCCS such as Software Failure Mode and Effects Analysis (SFMEA), Software Fault Tree Analysis (SFTA) and Systems-Theoretic Process Analysis (STPA). In this paper, the application of systems-theoretic approach is implemented on Ball Position Control System (BPCS) and performed the comparison between traditional methods and systemic methods for analysis and design. The proposed systems-theoretic approach can be applied to SCCCS in diverse sectors to identify and control the identified hazards. There is clear value in developing a systems-theoretic approach to safety analysis in SCCCS. Development of a SCCCS based on our proposed software safety approach significantly enhanced the safe operation of the overall system.

1. INTRODUCTION

Safety-Critical Computer Controlled System is safety-critical, integration of computation, social networking, and physical processes. Safety critical systems are used in multiple areas such as medicine or healthcare, aerospace, automotive, chemical processes, civil infrastructure, energy, manufacturing, transportation, entertainment, and consumer appliances traffic management and safety, automotive engineering, industrial and process management, avionics and space equipment, industrial robots, technical infrastructure management, distributed robotic systems and biological systems technology [1]. SCCCS involve trans-disciplinary techniques, combining concept of cybernetics, mechanic design, and process science [2, 3, and 4]. The emerging smart technologies, such as computers and software, are changing the types of accidents in these days. At the same time, traditional hazard analysis techniques assume accidents are caused by component failures or faults and over simplify the role of humans [5, 6]. In these days, most software-related accidents can be monitored to partial or faulty program requirements [7, 8]; however current hazard analysis methods like Fault Tree Analysis (FTA), Failure Mode Effect Analysis (FMEA) analyze component failures and easily overlook unsafe requirements.

Safety critical systems have been involved in several accidents. Some well-known software-related accidents in key industrial sectors are described below, in brief. Some of the most generally described software-related accidents in Safety Critical Systems involved a computerized radiation therapy machine called the Therac-25.

On June 4th, 1996 an unmanned Ariane 5 rocket launched by the European Space Agency exploded just forty seconds after its lift-off from Kourou, French Guiana. In February 2010, Toyota recalled its flagship high-technology hybrid car, the Prius, due to a brake software problem. The largest single American casualty of the Gulf War occurred on February 25th 1991, when an Iraqi Scud missile struck a barrack near Dhahran, Saudi Arabia, killing 28 soldiers.

Table 1: examples of accidents caused by software failure

Industry	Unsafe situation	Year
Biomedical	Therac-25 Cancer Radiation Accident,	June 1985,
	Software miss-configuration in CT scanner used for brain perfusion scanning.	August 2009-February 2008
Automotive	Toyota Prius Car Recall	Feb. 2010
Aviation	Korean Airways Flight 801 Accident	Oct.1997
Defence	USS Yorktown Breakdown	Sep. 1997
Aerospace	Ariane 5 Rocket Failure, Malaysia flight 370 crash	June 1996, March2014

1.1 Implications for SCCCS

For safety critical systems, the implications are clear - a single software-related failure leading to an accident may have extremely severe consequences. As computers are deployed in more and more safety critical applications, it is becoming obvious that there are still many significant problems to be solved. The real-life software failure accidents we have described earlier emphasize the need for improvement in both the management and technical aspects of safety critical systems development and deployment. The diversity of current standards and practices, and disagreement over concepts such as the use of Safety Integrity Levels (SILs) illustrate the absence of common understanding and agreement within the industry on the best route to improved products.

Software engineering of a safety-critical system requires a clear understanding of the software‘s role in, and interactions with, the system [9], [10]. The development of safety-critical systems demands a different, more rigorous approach than most other computer applications. As safety-critical systems are often real-time control systems they require the utmost care in their specification, design, implementation, operation and maintenance, as they could lead to injuries or loss of lives and in-turn result in financial loss [11], [12].

They require several disciplines that are still unfamiliar to many programmers and technology managers like safety engineering, software engineering of critical systems, and formal methods. Safety engineering teaches how to design systems that remain safe even when hardware or software fails. Software engineering provides methods for developing complex programs systematically. Formal methods are mathematically based techniques for increasing product reliability that overcome some of the limitations of trial-and-error testing and subjective reviews.

1.2 Safety-related terms:

Accident: An unwanted and surprising event that outcomes in an [unacceptable] level of loss [13].

Hazard: A system state or set of conditions that, together with a particular set of worst-case environmental conditions, will lead to an accident (loss) [14].

Safety: The freedom from accidents [13].

Software Safety: Software safety is a component of overall system safety. Software safety can be defined as:

- features and procedures which ensure that
  - a product performs predictably under normal and abnormal conditions, and

- the likelihood of an unplanned event occurring is minimized and its consequences controlled and contained
- thereby preventing accidental injury or death, whether intentional or unintentional

The notion of software safety was first mentioned in the MIL-STD-1574A (1979) which required analysis of software to identify and eliminate software errors relating to safety critical commands and control functions of space and missile systems. Since then, the role of software has becoming increasingly important and is being used in many critical applications, such as avionics, vehicle control systems, medical systems, manufacturing, power systems, and sensor networks [15, 16].

Safety: Freedom from those conditions that can cause mishaps [14].

Component failure accidents: An accident that results from component failures, including the possibility of multiple and cascading failures [14].

Component interaction accident: An accident that arises in the interactions among system components (electromechanical, digital, human, and social) rather than in the failure of individual components [14].

Goal for the paper is need to expand our view of safety such as technical, human, organizational and need to understand the whole system of interactions and need to build in safety from the start. Technical factors focus on independent random failures, but other technical problems pose growing challenge there are design errors, incomplete requirements, incorrect assumptions. The ultimate goal is to develop foundations and techniques for building safe and effective Safety critical computer controlled system.

This paper proposes a hazard-based safety-driven model-based system engineering methodology on SCCCS. The approaches described in this paper are focused on achieving and improving safety analysis process in SCCCS. This paper identifies the methodology for achieving the quality safety analysis of SCCCS. It gives a framework for integrating this methodology in the software development process. This paper does not seek to present metrics for each of the proposed quality of software safety. Identification and evaluation of metrics for each of the safety analysis is not within the scope of this research. Practical validation of the proposed model is demonstrated by the implementation of case study Ball Position Control system (BPCS).

2. MATERIALS AND METHODS

Safety-critical system failures may lead to catastrophic accidents, which are dangerous to the environment and to the people around. The methods used to analyze the failures of SCS are Failure Mode and Effects Analysis (FMEA), Failure Modes Effects and Criticality Analysis (FMECA), Fault Tree Analysis (FTA) [17], Event Tree Analysis (ETA) and Failure Mode Factors and Effects Analysis (FMFEA) [18]. The features of FMEA and FTA methods are described briefly followed by various analysis techniques like Graphical Requirement Analysis (GRA), Deductive Cause-Consequence Analysis (DCCA) and language like Unified Modeling Language (UML) etc [19].

To model a process the most critical facet is to catch the dynamic behavior. To clarify a bit in detail, dynamic behavior means the behavior of the system when it is running. So only static behavior is not sufficient to model a system rather dynamic behavior is more important than static behavior. Over the years several approaches for verification of software of a system has been undertaken as shown in following figure 1, which has been also applied to SCS safety analysis, but with little success. The inaccuracy in experimental safety evaluation is evident from the large number of recalls faced by several manufacturers of safety critical systems.

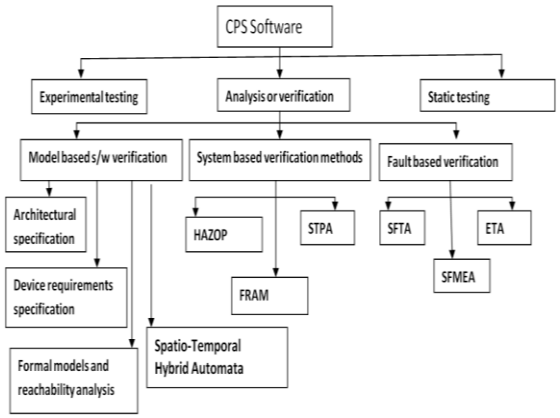

Fig. 1 Safety assurance approaches for software

a) Failure Mode and Effects Analysis (FMEA)

Failure Mode and Effects Analysis (FMEA) [18] is a method used for analysis to assure quality. It is used for analyzing the failures of a system that may lead to hazards. It is used to find their potential failure of a product or a process, to identify and estimate its importance and to recognize appropriate actions to prevent the potential failure of the system. FMEA is used for analyzing the individual risks of a system. The risks one-by-one are checked against each other to recognize the failures. FMEA does not provide a report on the total failure risk. For the analysis of failures, fault-tree analysis is more appropriate. The disadvantages of the FMEA technique are they can identify only the major failures of the system and the data flow of the system cannot be represented.

b) Fault Tree Analysis (FTA)

Fault Trees [5] [18] are used for the analysis of a system and to find the probability of failures. A Fault Tree has its representation in the form of a tree. It is used for analyzing the failures that may occur due to various conditions. A deductive analysis is used to find the failure that is placed at the top of the tree and it provides alternatives for the occurrences of failures. The limitations of FTA are it is a complicated process and the data flow is not represented.

FTA and FMEA are used for the analysis of a system for enhancing system reliability during the design but does not relate to a system undergoing maintenance. The limitations of FTA are it is a complicated process and the data flow is not represented.

Systems-Theoretic Process Analysis (STPA)provides an algorithmic and well-guided analysis process that identifies the causes of system hazards, including hardware component failures, software errors, complex system interactions, human errors, and inadequate organization management, policy, and procedures. The research approach used in this paper in respect of software safety analysis of SCCCS including hardware or software and integration of system are practically implemented for case study Ball Position Control System (BPCS) which are developed in the embedded system laboratory.

c) Systems-Theoretic Process Analysis (STPA)

STPA is a new hazard research technique, based on STAMP for SCS. It uses a collection of interacting loops of control to evaluate SCS. It can be used at any stage of the system life cycle, from before designing to after implementation. STPA technique is dependent on the following ways: define system hazards and related safety constraints, develop safety control structure for closed-loop system, recognize possibly unsafe control actions, determine how potentially insufficient control activities could happen.

There are several limitations of these approaches. The majority of software related injuries have involved errors in the requirements, not problems of the software to properly apply the requirements in SCCCS. A second significant issue is that most typical hazard research techniques such as FTA and FMEA work on a preexisting design in SCS. But systems and system styles have become so complicated that patiently waiting until a design is finished carrying out safety research on it is impractical. The only hope for practical and cost-effective safe design techniques in methods is to develop safety in from the beginning.

This paper makes the claim that safety analysis involves performing additional specialized activities beyond basic good software engineering practices. These involve additional design, analysis, measurement and verification

activities that occur concurrently with the software development activities. The paper validates the proposed safety analysis approach by presenting case study of laboratory prototypes of Ball position control system (BPCS). In addition, this paper will show that traditional system safety and reliability analysis techniques such as FMEA (Failure Mode and Effect Analysis), FTA (Fault tree Analysis) and Systems-Theoretic Process Analysis (STPA) can be successfully applied to systems with significant software content to complement the dynamic verification techniques.

3. PROPOSED FRAMEWORK FOR HAZARD ANALYSIS IN SCCCS

The development of safety-related systems relies heavily on the identification and subsequent analysis of system hazards. These hazards are identified in the context of the operation of the overall system in its operating environment. This means that modifying the operational environment may alter the hazards associated with the system and that the safety of the system components will depend directly on the top level hazards [20]. This in turn implies that software safety must be considered in the context of the overall system in its operating environment, and that the hazards applicable to the software must be related to those hazards identified at the top level. Therefore an important task of system safety analysis is to associate, where possible, potential hazards identified at the top level with the system components.

3.1 System Safety Analysis Lifecycle

Systems are comprised of many components such as electrical, software, and mechanical. Development of such systems follow a lifecycle paradigm with many phases [21, 22], this lifecycle is generally divided into system requirements analysis and specification, system design, system implementation, integration and test. System design phase can spawn many other lifecycles depending on the various system components [23]. Similarly electrical and mechanical lifecycles are spawned for their respective subsystems.

3.2 Software Safety Analysis Lifecycle

Figure 2 illustrates the software safety analysis lifecycle, which is integrated with the software development lifecycle. The software development lifecycle is divided into requirements analysis and specification, design, implementation, and test phases. Similarly, we divide the software safety analysis lifecycle into requirements safety analysis, design safety analysis, code safety analysis, and test safety analysis. Each software safety analysis phase is a sub-activity of the corresponding development phase. The input to each phase of the software safety analysis lifecycle is the software decomposition from the corresponding development phase, and the safety faults identified at the previous safety analysis phase. Except for the test safety analysis, the output of each phase of the software safety analysis lifecycle is the safety requirements, fault tree, and hazard dictionary for that phase. The output of test hazard analysis is the outcome of the execution of safety test cases. Each phase of the software safety analysis lifecycle is performed iteratively and concurrently within the corresponding development lifecycle.

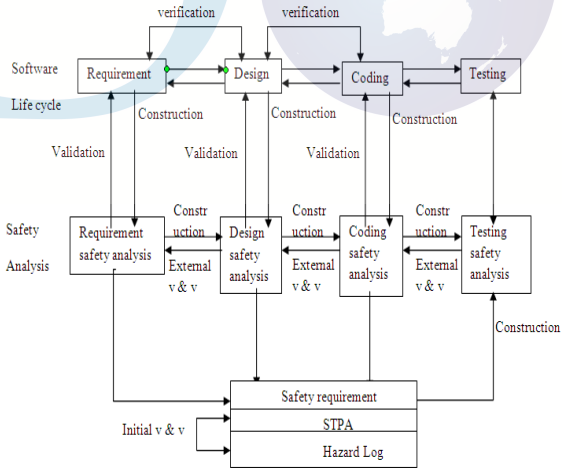

Fig. 2 Software safety analysis lifecycle integrated with software development lifecycle

Good requirements have several useful properties, such as being consistent, necessary, and unambiguous. The basis of sound design for a SCCCS is the identification, through systematic analysis, of the hazards that the system might encounter in operation. Traditional safety analysis techniques such as SFTA and SFMEA can indeed be successfully applied to

systems with significant software content. For SCCCS, software design must enforce safety constraints using STPA technique. Testing safety analysis is intended to supplement the existing requirements-based testing. The analysis of each hazard may involve use of common hazard analysis techniques such as Fault Tree Analysis (FTA) and Failure Modes and Effects Analysis (FMEA). FTA is a top down technique, beginning with the hazard as the top event as the result of a cause-effect relationship. By tracing backwards from effects to causes, we discover intermediate causes of the hazard. These intermediate causes are combined in the FTA using logical relationships, namely 'AND' and '0R'. Each branch of the FTA terminates with an internal or external condition. FMEA is a bottom up technique, typically used to examine the consequences of failures in system components. STPA, or Systems-Theoretic Process Analysis, is a new hazard analysis technique with the same goals as any other hazard analysis technique, that is, to identify scenarios leading to identified hazards and thus to losses so they can be eliminated or controlled. STPA is based on systems theory while traditional hazard analysis techniques have reliability theory at their foundation. STPA was designed to also address increasingly common component interaction accidents, which can result from design flaws or unsafe interactions among non-failing (operational) components. In fact, the causes identified using STPA are a superset of those identified by other techniques.

4. STPA ANALYSIS OF SAFETY-CRITICAL COMPUTER CONTROLLED SYSTEMS: BALL POSITION CONTROLL SYSTEM (BPCS)

The central objective of the BPCS system is to regulate the flow of air into a plastic tube so as to keep a small light weight ball suspended at a predetermined height called the set-point. Increasing the flow raises the ball and decreasing the flow lowers it. The BPCS experiment consists of: 3-foot long white plastic tube, light weight ball, DC motor fan, and ultrasonic sensor circuit and 89S52 micro controller. The proposed framework for hazard analysis of SCCCS in the BPCS consists of the following criteria:

- Design of control structure.
- Identification of unsafe control actions and causal factors.
- Creation of hazard log.
- Apply analysis to sub system components.
- Perform safety-guided design process.

Each criteria of the model is integrated into BPCS development process as described below. Here the system objectives are defined as Allow system to reduce the probability of unsafe system conditions through using a variety of physical, organization, cyber measures.

A) System and Software Hazard identification

A safety-driven design should start with identifying accidents and then defining the system hazards which would cause these accidents to occur. The hazards here can be defined as system states or a set of conditions that, together with a particular set of hazardous conditions, will lead to an accident [24]. Hazard is a state of system that leads to accidents [25]. The system-level hazard relevant to an accident includes:

H1	Ball Position Controller's Output signal is too high
H2	Ball Position Controller's Output signal too low
H3	Ball Position Controller's Loss of output signal to drive circuit
H4	Analog to Digital converter's failure to convert position (in hardware)

b) Identification of safety constraints

After the system hazards are defined, they should be translated into the corresponding safety constraints, which are restrictions on how the system can achieve its purpose.

hazards	Safety constraints (SC)
H1	SC1: Ball Position Controller's the output signal stays in between the boundary, duty cycle always in below the 100% when fan is moving

H2	SC2: Ball Position Controller's the output signal stays in between the boundary, duty cycle always in above the 0% when fan is moving
H3	SC3: Ball Position Controller's the output signal stays in between the boundary, duty cycle always in above the 0% when fan is moving

c) Identify unsafe control actions for BPCS

Once the hazards and related safety constraints have been defined, a typical socio-technical hierarchical structure with safety control processes, which is called hierarchical safety control structure, should be described. The next step is to develop the safety control structure for the system. The operational objective of the system is to maintain the ball at a predefined height in the tube. The software in the microprocessor provides the control mechanism; it implements a proportional controller (P-controller)-that is, the output signal is proportional to the amount of error in the ball's position relative to the set-point. A schematic diagram of the system is shown in Figure 3.

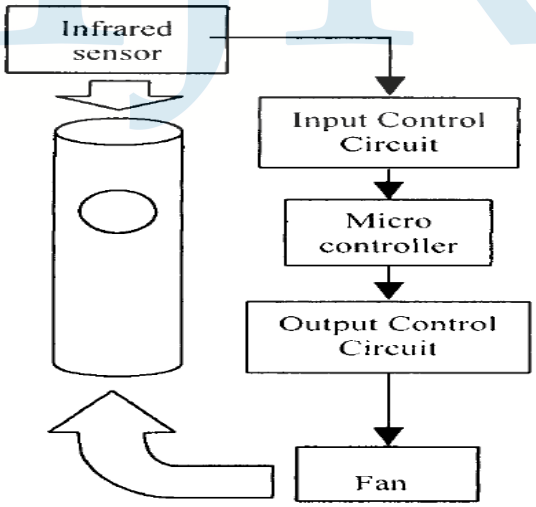

Fig. 3 Block diagram of the closed loop control system

d) Identify unsafe control actions for BPCS

The STPA process is used to analyze each of the high-level hazards. The two steps of STPA include identifying unsafe control of the system and determining how these control action could occur. A controller can provide unsafe control in the following four ways:

- A control action is not provided, missing or not followed.
- A control action is provided, but is wrongly provided.
- A control action is provided at the wrong timing, earlier or later than the required timing, or out of sequence with other control actions.
- The control action is stopped too early or applied too long.

After the safety control structure in system-level has been defined, the next step is to identify the potential inadequate control, which may drive the system into a hazardous state. STPA is a systemic method used for hazard analysis. This model considers hazards and causes in a systemic way rather than just based on component failures or failure events. At this level, Micro controller becomes a controller for the two lower controlled processes: Analog to Digital converter, Motor driver. Micro controller maintains the overall system, A to D converter and Motor driver processing.

Table 3: Unsafe control actions for BPCS

Control Action	**Not providing cause of hazards**	**providing cause of hazards**	**Incorrect timing/ order**	**Stopped too soon**
Output signal	Ball controller not provides Output signal within boundary command when the duty cycle goes to 100%. Ball controller not provides Output signal within boundary	Ball controller provides Output signal command when the duty cycle goes to 100%. Ball controller provides	Ball controller provides Output signal command too late or too early	Ball controller provides Output signal command too soon or too long

	command when the fan blows at the full speed.	Output signal too high command when the fan blows at the full speed		

e) Identify how the safety constraints could be violated for BPCS

After hazards have been identified, the following step should identify causal factors, which are very useful to figure out mitigating features against the hazard. Because hazards result from inadequate control and enforcement of safety constraints, the causal factors can be understood in terms of control flaws.

- Ball at top of the tube / Sensor damaged
- Fan runs too fast
- Output signal too high
- D4 pin output high
- Input duty cycle to ISR stuck high
- Ball at bottom of the tube
- System does not run
- Fan runs too slow
- Fan does not run
- Loss of output signal to drive circuit
- Output too low
- Connector
- Failure to initialize input register
- Failure to initialize output register
- Failure to initialize ISR
- Loss of output signal to drive circuit
- Output signal too high
- Output signal too low
- Loss of output signal to drive circuit
- Incorrect output data
- Response data too slow
- Dc motor fan runs too fast
- Dc motor fan does not run or runs too slow
- Dc motor fan runs too fast
- Dc motor fan does not run or run too slow
-

f) Hazard list and hazard log

Hazard
H1. Ball Position Controller's Output signal is too high.
System Element
Micro controller, Input control circuit, Output control circuit
Causal Factors
• CF1: Ball at top of the tube / Sensor damaged • CF2: Fan runs too fast • CF3: Output signal too high • CF4: D4 pin output high • CF5: Input duty cycle to ISR stuck high • CF6: Ball at bottom of the tube • CF7: System does not run Etc…..
Safety constraints
SC1- Ball Position Controller's the output signal stays in between the boundary, duty cycle always in below the 100% when fan is moving SC2-Ball Position Controller's the output signal stays in between the boundary, duty cycle always in above the 0% when fan is moving

5. HAZARD ANALYSIS USING FMEA, FTA, STPA ON SCCCS

The STPA process aims to eliminate or control the system hazards through eliminating the unsafe control actions, which is reached by examining the control loop and the process models. The system as a whole has to be examined. By using STPA in our case, more hazards, failure modes and causal factors are identified, By using a safety control structure and a general control flaws classification to analyze causal factors of each identified hazard, STPA may help the analyst to find more failure modes and causal factors. This paper identifies 65 scenarios found by FMEA where as 134 identified by STPA. FMEA results show that it identified only single fault cause of hazard where as STPA identified complex causes of hazards, multiple failures, and no component failures that lead to a hazard.

Table 4: hazard analysis methods comparison

Attributes	Methodology		
	FTA	**FMEA**	**STPA**
Single failure event	Yes	Yes	Yes
Multiple failure events more than one event	Yes	No	Yes
System approach model (organizational-environment-technical)	No	No	Yes
Able to address system interaction accidents	No	No	Yes
Applicable in design phase	Yes	Yes	Yes
Applied with limited system information	No	No	Yes
Ease of application	No	No	Yes
Suitability for large objects	No	Yes	Yes
Criticality analysis	Yes	No	Yes
Failure mode identification	Yes	No	Yes

6. CONCLUSION

This approach is derived from research in the fields of systems engineering, software engineering, and safety engineering. The purpose of the research in this paper was directed according to two themes. It is hoped that the results presented will help and provide new insights for anyone working on safety analysis and implementation. The approaches described in this paper are focused on safety analysis in Safety Critical Computer Controlled System, and thus their practical use, it is hoped, will help to produce safer software. While the soundness of the concepts within the proposed Systems-theoretic approach has been demonstrated, further validation of the practicality of applying the concepts in an industrial setting would be beneficial. The most suitable approach for performing this additional validation would be to apply the approach on an industry-strength project. However, the intention was to define a starting point to approach for Safety-Critical Systems in a more systematic, harmonized and practical way than what was found in the literature and standards. It is hoped that this paper has, at least in a small way, contributed to the foundations of a scientific discipline of software engineering of safety critical systems theory, as well as adding some practical new approaches to the specific problems of safety analysis in Safety Critical Computer Controlled Systems.

REFERENCES

[1] Khaitan et al., "Design Techniques and Applications of Cyber Physical Systems: A Survey", IEEE Systems Journal, 2014.

[2] Hancu, O; Maties, V.; Balan, R.; Stan, S. (2007), Mechatronic approach for design and control of a hydraulic 3-dof parallel robot, The 18th International DAAAM Symposium, "Intelligent Manufacturing & Automation: Focus on Creativity, Responsibility and Ethics of Engineers".

[3] F, E.A., Seshia, S.A.: Introduction to Embedded Systems - A Cyber-Physical Systems Approach, LeeSeshia.org, 2011.

[4] Suh, S.C., Carbone, J.N., Eroglu, A.E.: Applied Cyber-Physical Systems, Springer 2014, Rad, Ciprian-Radu; Hancu, Olimpiu; Takacs, Ioana-Alexandra; Olteanu, Gheorghe (2015).

[5] Dekker S., Ten questions about human error: a new view of human factors and system safety. Human factors in transportation 2005, Mahwah, N.J: Lawrence Erlbaum Associates Xix, pp. 230.

[6] Dekker, S., The field guide to understanding human error 2006, Aldershot, England; Burlington, VT: Ashgate xv, pp. 236.

[7] Lutz, R.R. Analyzing software requirements errors in safety-critical, embedded systems in IEEE International Conference on Software Requirements, 1992.

[8] Leveson, N., SafeWare: system safety and computers. 1995, Reading, Mass.: Addison-Wesley, Xvii, pp. 680.

[9] Robyn R. Lutz, Software Engineering for Safety: A Roadmap, Proceedings of the Conference on The Future of Software Engineering, June 04-11, 2000, Limerick, Ireland, pp.213-226.

[10] John C. Knight. Safety Critical Systems: Challenges and Directions Proceedings of the

24th International Conference on Software Engineering (ICSE), Orlando, Florida, 2002.
[11] Debra S. Herman, Software Safety and Reliability Basics (ch.2), Software Safety and Reliability: Techniques, Approaches, and Standards of Key Industrial Sectors Wiley-IEEE Computer Society Press, 2000, pp.13-31.
[12] Douglas C. Schmid, Adaptive Middleware: Middleware for Real-time and Embedded Systems Communications of the ACM, Volume 45 Issue 6, June 2002.
[13] Leveson, N., SafeWare: system safety and computers. 1995, Reading, Mass.: Addison-Wesley, xvii, pp. 680.
[14] Leveson, N., Engineering a safer world: systems thinking applied to safety. Engineering systems 2012, Cambridge, Mass.: MIT Press.
[15] P. V. Bhansali, Software Safety: Current Status and Future Directions, ACM SIGSOFT Software Engineering Notes, Volume 30 Number 1, page 1, January 2005
[16] Dongfeng Wang, Farokh B. Bastani, and I-Ling Yen, Automated Aspect-Oriented Decomposition of Process-Control Systems for Ultra-High Depend-243 ability Assurance, IEEE Transactions on Software Engineering, Vol. 31, No. 9, September 2005.
[17] G Thangamani, Generalized Stochastic Petri Nets for Reliability Analysis of Lube Oil System with Common –Cause Failures, American Journal of Computational and Applied Mathematics, Scientific and Academic Publishing, USA, 2012, pp. 152-158 [doi: 10.5923/j.ajcam.20120204.03] .
[18] N.G. Leveson and C. S. Turner, an Investigation of the Therac-25 Accidents. IEEE Computer, pp. 18- 41, March 1987.
[19]. Perneger, T.V., the Swiss cheese model of safety incidents: are there holes in the metaphor? BMC health services research, 2005. Pp.71.
[20] Hazard Analysis, Hazardous Industry Planning Advisory, Paper No 6, Planning, NSW Government, Chapter 2, January 2011.
[21] J.D Sailor, "System Engineering: An Introduction", in System and Software Requirements Engineering, edited R.H.Thayar and M.Dorfman, pp.35-47,1990.
[22] R.H.Thayar and W.W.Royce, " Software System Engineering" , in System and Software Requirement Engineering, edited R.H. Thayar and M.Dorfman, pp. 77-116, 1990
[23] M.Dorfman, " System and Software Requirements Engineering", in System and Software Requirements Engineering, edited R.H. Thayar and M.Dorfman, 1990.
[24] Lee, E.A., Seshia, S.A.: Introduction to Embedded Systems - A Cyber-Physical Systems Approach, LeeSeshia.org, 2011.
[25] Suh, S.C., Carbone, J.N., Eroglu, A.E.: Applied Cyber-Physical Systems, Springer 2014, Rad, Ciprian-Radu; Hancu, Olimpiu; Takacs, Ioana-Alexandra; Olteanu, Gheorghe (2015).