A Review of Applications of Formal Specification in Safety-Critical System Development

Emanuel S. Grant
School of Electrical Engineering and Computer Science
University of North Dakota
North Dakota, USA
1.701.777.4133
emanuel.grant@und.edu

Smruti Priyambada Nanda
School of Electrical Engineering and Computer Science
University of North Dakota
North Dakota, USA
1.701.777.7331
smruti.nanda@und.edu

ABSTRACT
Since the advent of the computer and computer programming there have been many attempts to improve the quality of the software systems developed. At various stages in this evolution of development techniques, processes, and methodologies, a review of the current trend in software development is conducted. One such current trend is in the realm of safety-critical system development. Safety-critical systems are characterized by the resulting potential of harm to or loss of life if such systems should fail during operation. A strategy applied in developing such systems is the use of formal specification techniques. Formal specification techniques are the application of rigorous techniques to assess the correctness of system design. The use of formal specification techniques in safety-critical system development has been in place for a number of decades and there have been multiple reviews and comparisons of the successful and failed application of formal specification techniques. This report reviews examples of the application of formal specification techniques in a number of application domains, with a focus on the types of error detection and correction associated with the particular technique. The benefit of this work is towards the assessment of the suitable of a specific formal specification technique with a particular problem domain.

CCS Concepts
• Software and its Engineering → Software Notations and Tools→ Formal Language Definitions

Keywords
Computer programming; Formal notations; Avionics; Requirement specification; Formal specification; Methodology; System life-cycle

1. INTRODUCTION
In the past decades, there have been a number of methodologies and models for software development have been introduced [1]. Some software development methodologies improved on the ideas of past ones and new ones have been introduced. Multiple formal specification methodologies, and notations that are used at various phases of the software engineering cycle; each with their own pros and cons, but there are currently no single formally established methodologies to deliver a correct software system. Figure 1 illustrates the evolution of the most popular software development methodologies and notation over the early decades. All the methodologies of Figure 1 are insufficient to satisfy the complete requirements for developing safety-critical systems.

Safety-critical software systems are a subset from problem domains in which humans interface and are directed affected by the operations of such systems. The failure of safety-critical systems may result in harm or loss of life to the humans that either operate or interface with such systems. Because of this, it is necessary for additional tools and techniques to satisfy the high demand for system correctness of safety-critical software development. Formal specification techniques/methodologies are strategies for presenting a rigorous view of a software system in a mathematical notation. The purpose for applying formal specification techniques is to facilitate a rigorous analysis of the software system. The benefit of applying formal specification techniques in software development is that the analysis of a rigorous system representation provides a more reliable system. This level of system reliability is required in the domain of safety critical software development. Safety critical software systems do not have the option of restarting at a point of failure, as is acceptable during the operation of non-safety critical systems. The application of formal specification techniques has its greatest benefit at the early stages of software development, i.e. during the system requirement analysis phase, where the goal is to verify and validate a complete, correct, and unambiguous set of system requirements [2]. This need for a rigorous development environment for safety-critical systems leads to the use of formal methods in safety-critical systems development. According to Jeannette M. Wing in [3], the undesired features of inconsistence, incompleteness, and ambiguity are mitigated with the application of formal specification techniques. Thus, it can be asserted that the use of formal specification techniques, in software development, is a great aide to the production of reliable software systems, specifically in the domain of safety critical systems.

The application of formal specification techniques realizes the greatest benefits when it is applied at the earliest phase of the development cycle. This is self-apparent, as any error will persist from the point of its introduction in the development cycle, through to the end of the cycle. There is also greater cost to correct and eliminate an error; the later it is identified in the development life cycle. While the cost of conducting formal

ICCDA 2020, March 9–12, 2020, Silicon Valley, CA, USA

ACM ISBN 978-1-4503-7644-0/20/03…$15.00
https://doi.org/10.1145/3388142.3388175

specification techniques is high the reduction in cost from the elimination of errors from the application of formal specification techniques, justifies its cost.

Written and spoken communications are two of the most used form of documenting the specification of a software system. The use of written system specification, in natural language, is the genesis of the undesired features of ambiguity, incompleteness, and inconsistency in documentation of system requirements. These issues are of the highest concerns in the domain of safety critical systems. This accounts for the use of specification languages for documenting the requirements of software systems; specifically safety-critical software systems. In the application of formal specification techniques/methodologies, a specification language refers to the use of a formal syntactic notation, associated semantics, and precise rules for the notation use in representing aspects of the system [3]. The Z notation [4] is an example of a formal specification language that is the use in software development. The undesirable system features of ambiguity, incompleteness, and inconsistency are addressed with the use of a formal specification language. Ambiguity occurs in the system requirements when multiple meanings or interpretations can be applied to a single or group of concepts in the domain of consideration. Incompleteness, occurs when aspects of the description of a system requirement are absent. Inconsistency is evident when two or more requirements are not disjoint and their intersection results in conflicts. The review, reported on in this paper, looks at a number of cases where formal specification techniques were applied in the development of safety critical software systems, in a number of industrial fields to address the issues of ambiguity, incompleteness, and inconsistency at development cycle.

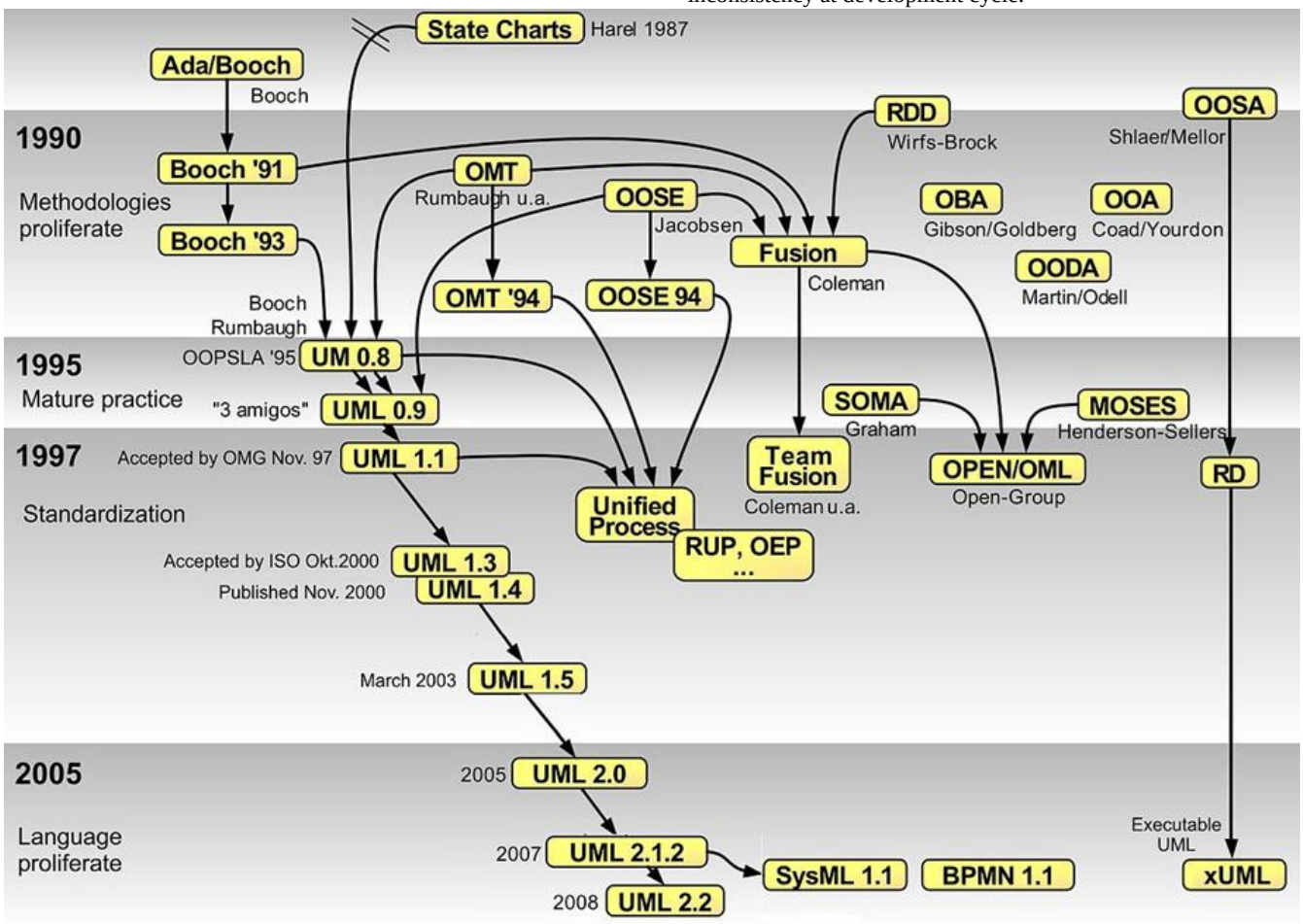

Figure 1. The evolution of object-oriented methods and notations 1980s – mid 2000s

The structure of this report is as follows: Section 2 is the background section that discusses the on-going problems in applying formal methods for safety-critical systems development. Section 3 is a comparative analysis of various domain applications of safety-critical system. Section 4 specifies inferences drawn from the application reviews of the previous section, and Section 5 provides a conclusion.

2. BACKGROUND

Formal specification techniques offer the greatest benefits throughout the software development life cycle and beyond, as during the evolution of the system. Formal specifications of the system may be developed at the early stages of the development cycle and is referenced in later stages of the development cycle. The use of formal specification techniques is aimed at the verification and validation of the system under development. Research as determined that benefits from the application of verification and validation strategies are realized when they identify errors during the early and initial stages of system development. Requirement engineering, being at the early stage of system development, is the phase from where an error could be propagated to later stages. This section discusses some of the challenges commonly faced in the requirements engineering phase of software development, with a focus on safety-critical system

development. Two instances of requirement definition errors that have occurred in real world systems are examined in this section of the report. In addition, a number of current safety-critical software system development standards are reviewed

2.1 Requirement Engineering Challenges

In all software development life cycle models, the requirements engineering phase is the first to the conducted. Martins and Gorschek [5] examined that initial phase of the development cycle, i.e. requirements specification, for safety-critical systems. They explore the many challenges that the practitioners of safety-critical systems face. Martins and Gorschek asserted many safety-critical systems failures could be traced back to the software development process working with incomplete specifications. From their interview of software development practitioners, with a range of development experience, ranging between two and twenty-five years they concluded that the development of safety-critical requirement specifications should be done by the experts.

The use of traditional software development process models, such as the waterfall methodology, has its greatest challenges during the requirements engineering phase of the development cycle. Early practice saw the use of the same the same development methodologies on both safety and non-safety critical systems. This occurred because there were no established methodologies for specific use in one domain versus the other. Fault Tree Analysis (FTA) [6] has demonstrated to be the most desired for the analysis of safety-critical system. Other fault analysis strategies, such as FMEA (Failure Mode and Effect Analysis) [7] and HAZOPs (Hazard and Operability Studies) [8] are also used. GSN (Goal Structuring Notation) [8] is one of the few formal notations with applicability to the development of safety critical systems. In some safety critical application domains, software systems have to be certified before they can be placed into productive use. Such application domains usually have development specifications/guidelines that articulate approved development methodologies. Such restrictions limits the use of unapproved development methodologies in developing such safety critical systems and only established (traditional) methodologies may be available.

Another challenge in the use of traditional software development methodologies is that of the assessing the non-functionality of *usability*. In the work of Martins and Gorschek [5], a number of new and useful software development methodologies were reviewed but they lack evidence of appropriate usability determination. The researchers concluded that such methodologies would be unacceptable to software developers where usability is a dominate feature. Thus, usability and usefulness are important to the requirements engineering phase of software development.

The existing gap between new software development technologies that are derived from academic research and adaption of these technologies in the software development industry is the next identified challenge. Martins and Gorschek analyzed the impact of new software development technologies in the industry, through the application of a rigor-relevance model [9]. This type of analysis has be written about in over 150 articles, where in half of those analysis efforts were conducted by a single researcher; with another quarter of the documented analysis work done by the researchers and the developers and the remainder done by the developers. The researchers [5] theorize that the validation effort would be best done by the team of researchers and software development practitioners. It is also their view that this could be accomplished by the software developers being more involved with the research work in academia.

Throughout the software development, cycle there is an issue of communication when work moves from one phase of the development cycle to another; this is evident when communication has to occur between the requirement engineers and the other team members, such as the safety engineers. This is the next challenge discussed. The main form of communication, in these instances, is usually informal and often results in some requirements not being delivered. In developing safety-critical systems it is necessary that not only are all requirements precisely stated but that they also be fully communicated throughout the full development life cycle.

A final challenge reviewed is that of accounting for non-safety-critical systems in safety-critical system development because the non-safety-critical systems interact with the safety-critical systems. An example of this type of safety-critical system interacting with non-safety-critical system is demonstrated when an emergency medical system (the safety critical system) has to interact with a general communication system, such as a telephonic system. This then requires that the engineers working on the safety-critical system engage with the communication system engineers in a formal framework, as a component of the overall safety-critical system development [3].

2.2 Requirement Errors in Real Life

In Robyn R. Lutz [10] work, there was the examination of a series of software errors that occurred during the testing of the software that were installed on the “Voyager” and “Galileo” spacecrafts. In support of his work, Lutz formulated a definition of a system error, as a variance between correct (expected) value of a process and the actual (observed) value of the process. The researcher asserted in the report errors in the development of safety critical systems arise from misunderstandings in the system requirements specification, which has high consequences for safety critical system requirement specifications. The classification of errors that may occur in the development and operation of safety-critical systems include, process flaws, program faults, and human errors.

The program faults, which accounted for 93 percent of the errors that occurred on the Voyager spacecraft and 72 percent of the errors that appeared on the Galileo spacecraft [10], occur whenever communication between the development team members and other team members’ safety-critical system development is faulty. These percentages of errors are significant and the main cause was miscommunication between the members and groups of the development teams and was traced to incorrect software requirement specifications.

An additional type of error that occurred because of working with incomplete requirement specifications is Functional Faults that are placed in the category of Program Faults. Functional faults are evident when the development team is working with incomplete specifications or specifications that are not fully understood by the team of developers. Such scenarios are problematic in general for software development and would be detrimental to safety-critical system development. The cited studies of the identified errors on the aforementioned spacecrafts, demonstrate the impact of such errors in real-world situations. The goal of this work was directed in a different direction from that of prior similar works, as follows: The software system deployed on the spacecrafts had complex requirements, which resulted in software systems, themselves, being complex. Errors in complex systems of these kinds will

lead to failures that may be hazardous. The existence of incomplete or undetermined system requirements in the development of safety critical systems will lead to further errors in the system design and implementation. Lutz work [10] reviewed the development variances in the development of non-safety-critical systems and the development of safety-critical systems.

It has been previously acknowledge that requirements engineering is vital to the success of the development of safety critical systems. From the analysis of [10], 387 errors were identified during the testing phase of the software system. The analysis further identified that the majority of the errors existed in the system requirements [10]. Some of these system requirement errors arose from incomplete requirements specifications, while others arose from misunderstanding of the requirements specifications. A third type of system requirement specification error is that of the absence of formal specification of the system requirements. It was determined that within the requirements engineering activities for safety-critical system development, process flaws and program faults were evident. Lutz [10] recommends that there is a need for better software development methodologies to respond to the changes that occur throughout the software development life cycle, with specificity to the domain of safety critical systems, such as that of the two identified spacecraft systems.

2.3 Safety-Critical Systems and Formal Methods

In the work of Hamilton et al., as reported in [11], the authors document an investigative projects conducted by NASA in the use of formal methods (specification) in the development of safety critical software systems. The work sought to ascertain the benefit of applying formal specification methodologies during requirement engineering of safety critical software systems. The authors asserted that prior studies found that most software development errors occurred at the requirements engineering phase and the cost of these errors are great when attempts are made to correct that during later stages of the software development cycle. An additional issue identified in [11] was one the authors labeled as the "quality ceiling" [11]. Safety-critical systems have a high quality requirement, which necessitates the derivation of new software development requirements engineering strategies when there are no suitable ones available [11]. An absence of the skill to accomplish this task, results in the situation the authors title "quality ceiling" [11]. The impact to the development of safety-critical software systems by the previously identified problems can be mitigated by the application of formal specification methodologies. In [11], a trio of projects was used in evaluating the benefits of applying formal specification methodologies.

Hamilton *et. al.*, examined three projects, where two were avionic software systems for the space shuttle program. The researchers asserted that the issues with the three projects could be classified as being of the type "quality ceiling. The implication of this assertion was that the software system requirements had no path to improvement. The avionic software system, under consideration, was designed to provide flight navigation functionality. When the astronaut activated a specific set of jet thrusters, the system implemented a directional change. The software system implementation of the various jet thruster selection process was used over a number of years. Subsequently, new system specifications were created from the requirement after the application of formal specifications to the software system requirements.

The introduction of formal specification techniques in the requirements engineering phase of the system development process, resulted in greater understanding of the system requirements and the identification of defects in the existing system requirement specifications. An example of this discovery is as follows: the statement that the software process should "never select a failed jet thruster" was included in the initial specification. This requirement condition was never implemented in the initial system design. This missing system requirement condition was identified after the application formal specification methods. Before the use of formal specification in the software development exercise, this system requirement was not identified, even though it was of critical importance to the operation of the system. The authors [11] highlighted the importance and benefit of formal specification methodology in safety-critical system development by citing this example.

In the first trial project, the project software requirements engineering team were concerned that the requirements analysis phase of the software development methodology was not appropriately structured. It was there assessment that the identification of the system requirements for the problem domain was largely left to the team of developers. Three highly critical issues were identified, in a larger set of over 70 issues that were identified because of the formal specification analysis of the system. The requirements engineers also determined that it was essential that formal specification methods be used during requirement analysis [11]. The researchers also discovered errors in the interface of the software system as that were difficult to identify during requirement analysis. The application of formal specification methods proved to be useful in this exercise.

The second trial project was avionic software for On-Orbit-Digital-Auto-Pilot. The use of formal specification techniques on this project also resulted in the identification of a small set of interface problems. While formal specification methods are not usually employed in the system prototyping phase of software development, its application may identify issues/errors that arise from the design of the user interface system; this was the case with the second trial project.

2.4 Safety-Critical Standards

Jonathan Bowen [12] identified a collection of safety-critical standards that are appropriate for the application of formal specification techniques in the development of safety-critical software system. It is to be noted that there are established software development standards for the development of safety-critical systems; in addition to the use of formal specification, methodologies in some specific application domains and this paper address this issue. In [12] it is noted that the software engineers have to be cognizant of a greater quantity and quality of issues when developing safety-critical software systems versus that of developing non-safety-critical software systems. In is within this context that it is deemed necessary to review established standards for the development of safety-critical systems. Of the several standards discussed in Bowen's paper, the RTCA DO-178 [13] is reviewed in this report. Formal specification methods are used as a part of this aviation requirement specification [12]. A small collection of standards is documented in Figure 2, as taken from Bowen's paper [12].

The RTCA DO-178C is the de facto standard guideline for avionic software development in the USA. A corresponding EUROCAE (European Organization for Civil Aviation Equipment) guideline for software development, the ED-12, was development

in conjunction with DO-178C for use in Europe. The DO-178C describes the requirements for certification of software system in airborne operation but does not specify how these requirements are to be met. It is left up to the software developers to incorporate notations, processes workflows, and methodologies that comply with the objectives of DO-178C for certification. This standard supports the use of formal specification methods, objected oriented-methodologies, mode-based development and verification, and CASE tool usage. DO-178C had a predecessor, DO-178B, which has been retired with the release of DO-178C in 2012. DO-178B was updated in 1992 but did not address a number of the existing software development methodologies specifically, objected-oriented, model-based development, and formal specification. The DO-178C was codified to address those and other deficiencies in DO-178B. The DO-178A addresses the certification of electronic hardware for use in airborne operation.

The work presented in [12] highlights the many standards that have been defined and used internationally. The authors noted that the presence of the many standards does not mean that they are all used, in their respective domains. The authors also recommended that the domain-specific software development standards and guidelines should undergo periodical revisions to ensure that they are current with the evolving safety requirements of the respective domains. An important observation from [12] is that some of them may not address the use of formal specification in the development of such systems.

Country	Body	Sector	Name	FMs content	FMs mandated	Status	Year
Europe	IEC	Nuclear	IEC880	No	N/A	Standard	1986
UK	HSE	Generic	PES	No	N/A	Guideline	1987
Europe	IEC	Generic	IEC65A WG9	Yes	No	Draft	1989
			IEC65A 122	Yes	No	Proposed	1991
Europe	ESA	Space	PSS-05-0	Yes	No	Standard	1991
UK	MoD	Defence	00-55	Yes	Yes	Interim	1991
UK	RIA	Railway	–	Yes	No	Draft	1991
US	RTCA	Aviation	DO-178A	No	N/A	Guideline	1985
			DO-178B	Yes	No	Guideline	1992
Canada	AECB	Nuclear	2.234.1	Yes	Yes	Report	1992
	OH		982C-H69002	Yes	Yes	Procedure	1993
US	DoD	Defence	MIL-STD-882B	No	N/A	Standard	1985
			MIL-STD-882C	No	N/A	Due	1993
US	IEEE	Generic	P1228	No	N/A	Draft J	1993

Figure 2. List of Development Standards Source: Bowen, J. (1993) [12]

3. FORMAL SPECIFICATION TECHNIQUE ANALYSIS

In order to provide a comprehensive survey of formal specification methodologies of safety-critical software systems a review of the application of formal specification in safety-critical domains is presented in this section.

Formal specification techniques have been applied in multiple safety critical software system domains. These applications of formal specification techniques in safety critical software system development were instituted for a variety of reasons. In some cases, the application of formal specification techniques was adopted for formally verification of the functional and nonfunctional requirements of the safety critical system. In other applications, it was for lowering the risk of possible errors. The comparative analysis conducted in this report, the benefits of applies formal specification techniques in safety critical software system development will focus on the following: the properties of formal specification techniques and determine whether the benefits are preserved. The benefits of applying formal specification techniques under consideration are “unambiguity”, “consistency”, and “completeness”.

3.1 Safety-Critical Medical Equipment

The first safety critical domain of review wherein formal specification technique was applied is in the area of a medical equipment for cancer treatment [14]. The movement of the neutrons in the beam emitted by the cyclotron was operated by a device that was controlled by the safety-critical software system. The formal specification models that were designed, during the software development cycle, were reviewed by the researchers in [14]. The software engineers on this project developed Z specification models of the cyclotron software control system. There were a number of hardware safety critical features put in place for the operation of the cyclotron; one of these being the door of the device must be closed before the beam is turned on. The design of this safety-critical software system required the integration of multiple sub-systems various components and operations of the device. The Z notation was used to document all the known constraints on the safety critical functional requirements of the system. During the operation of the device, multiple safety-critical situation could arise. An example of one such situation would be one in which, a door was opened during the emission of the radiation beam. A preventative action, for such a scenario, would be the deactivation of the beam emission, if a “door-open” state were detected. The process of incorporating all intervening conditions in the formal specification of the safety

critical system facilitates creating complete specification of the system.

The requirements engineering process for the system involved the capture of all known states, operations, and interlocks. The formal specification technique applied on this project required the aforementioned system features be specified in the Z notation. The software engineers on the project had to develop specifications to cover thousands of state variables in the system. This metric was indicative of the size of the other system features on which formal specification techniques were applied, i.e. for which Z notation specifications were developed. This work documented in [14] did approached the formal specification process in two phases. The initial phase required the individual identification of the system components. The formal specification work then transition to the next phase in which, the identified system components were specified by use of the Z formal specification notation.

The application of the software engineering principles of modularity, separation of concern, and decomposition allowed the developers to address each component individually then address the composition of the individual components. The developers identified a number of state variables that had no associated state components but these state variables had to be specified. This was accomplished by defining single design-level components for them. The formal specification of the system in the Z notation necessitated the definition of a set of Z idioms for the description of various operations. The formal specification developers were able to coordinate their effort more effectively by the use of sub-systems, conditions, and modes.

The developers of the Z system specification could not guarantee the completeness of the specification but provided assurance that the derived formal specifications of the software system were consistency. The formal specification developers also asserted that further and complete Z specification of the system could be achieved if the appropriate commitments were made. It is herein asserted that the use of formal specification in the design of the safety-critical system, as described in [14] facilitated the correct specification of the known system requirements and assurance of the requirements being consistent and unambiguous.

3.2 Safety-Critical Automotive Systems

The automotive industry is one of the major demanding manufacturing industries, in which safety critical operations are conducted at various stages of the production cycle and within the products. The current state of development in this industry is that there is a high demand for the correct development of safety-critical software systems, in which formal specification methodologies are applied. The purpose for applying formal specification technique in this case was for the elimination of ambiguity in the specification of the system [15]. An addition goal of the application of formal specification techniques in the system development of [15] was to achieve an acceptable level of consistency and completeness in the development of the safety critical systems. The standard applied in the automotive software development domain is that of the ISO 26262, which applies to the guarantee of safety in programmable electric and electronic equipment [16]. The ISO 26262 standard is regarded as the definitive standard for E/E systems, though there are a number of processes that were described in an updated approached that is included in the work of Fang, *et. al*. [15]. A part of this work effort resulted in a different standard the GJB2786A-2009, being used to account for the missing processes from ISO 26262. The GJB2786A-2009 standard is in the development of software systems in the military and nuclear power engineering. In this standard, the use of formal specification methodology is emphasized. The specification addresses the use of formal specification at the system requirement analysis and the software testing phases of development.

OSEK OS [17] is the specification of a software operation system that is intended for use in the automotive industry. The properties, as stated in the specification for this operating system, had to be formalized, using the approach established in the paper [15]. An example of one of the properties that was formalized is given by "The interrupt routine cannot get the resources". During the application of formal specification techniques to the specification of the OSEX operating system, a number of errors were detected. A number of these errors were considered hazardous to the safe operating of the vehicles on which the operating system was applied. The application of formalization specification techniques to the properties of the OSEK operating system increased the confidence in the elimination of ambiguity, system requirement completeness, and system requirement consistency. It was the conclusion of the work that all the desired properties that are achievable form the application for formal specification techniques were achieved on this project.

3.3 Safety-Critical Avionics Systems

The next industrial domain in which formal specification has been used that is reviewed in this survey is that of aviation systems. The papers [18,19,20] document the use of formal specification techniques in the aviation domain of safety critical software systems. In the domain of avionic software systems, specifically on-board avionic software systems the scheduling of such systems operations is considered to be of the highest in the area of safety critical systems [18]. The operating systems that manage the scheduling and execution of the executable modules and resource management are considered vital operations. Budiyanto *et. al*. paper [19] examined the work done in applying formal specification techniques to the development of the Integrated Modular Avionics (IMA) system. The IMA was initially applied on the Boeing 777 aircraft. The ARINC-653 standard, is one of the standard specification used in the development of avionic software systems and was used on the IMA project [19]. The approach taken on the IMA project was to partition the system components in order to reduce the coupling between the components. The approach taken herein resulted in the division of the IMA system into three sub-systems, namely: a health monitoring system, the hardware interface system, and the real-time operating systems. Concurrency process could be independently executed with each partition. The function of the health monitor system was to ensure that the entire system was operating as specified by the requirements specification, and to provide a recovery mechanism in the event of any system failure. A conclusion reached in the work of developing IMA is that formal specification techniques can be successfully used in developing real-time systems [19]. The researchers determined that use of formal specification techniques facilitates the elimination of ambiguities and provide for completeness of the system specifications.

The Dong *et. al*. paper [18], similar to that of the Budiyanto *et. al*. paper [19], addresses the use of formal specification techniques in developing avionic software systems. The difference is that the Budiyanto *et. al*. paper [19] paper looked at a civilian avionic software system, the Dong *et. al* paper focuses on an instance of avionic software system for the military. While the two papers

examined works in to avionic application areas, they both focused on a similar problem of scheduling activities in software system operation. In the Dong *et. al.* paper [18], the work of the developers was the definition of formal specification of the system scheduling requirements, while providing for the generalization of the system specification. This generalization of the system specification was to allow for its use in other operational flight programs [18]. On this project, the developers identify the most difficult system specification issue, as that of incorporating timing constraints for synchronization of the processes. The system requirements stipulated that two constraints were to be addressed in synchronizing the processes. The two constraints that had to be addressed in the synchronization process were the precedence constraint and the exclusion constraint. The precedent constraint was targeted at resource allocation management and required that one process had to complete before another process that uses the associated data could be started. The exclusion constraint restricted two or more processes from attempting to execute concurrently. In compliance with object-oriented software system development on this project, the Object-Z notation was employed as the formal specification on this project. The developers had to option to use either a preemptive or a non-preemptive scheduling strategy on this project, but for the OFP option the selected a non-preemptive approach. Another system requirement addressed was the avoidance of deadlock situations, wherein two or more processes would be competing for a single resource, resulting in a collision. A series of Object-Z schemas were defined to represent various aspects of the system. An example of an Object-Z schema that was developed was the representation of a varied time interval process for resource availability. To facilitate the representation of the periodic processes, it was specified as an object in the Object-Z notation. The Object-Z schema contained the constraints for the process synchronization, as a means to represent the synchronization of the processes. Multiple processes were constituted in a schedule, which then had to be synchronized. The object-oriented feature of inheritance was used to represent the schedule and process relationship [18].

On this military avionic software system project, the system requirements were for the operation of two separate computers to schedule, such that one was for normal avionic system operation and the other for weapons transmittal. The scheduling challenges including that of synchronizing the various processes on each computer separately and synchronizing the processes across the two computers. Towards making the specification of this military avionic software system complete, the object-oriented concepts of inheritance and class-union were employed in the formal specification representation. Another feature of the formal specification system design is that the software specification representation was generalized in order to be applicable to other similar aircrafts [18]. At the end of the project, it was asserted that the use of formal specification technique was a success. In light of that assertion, it can be assume that the desired outcomes from the application of formal specification techniques, such as unambiguity, consistency, and completeness were achieved on this project.

Bharadwaj, and Heitmeyer [20] was the final work reviewed in this domain of avionic safety-critical software systems. In this paper, the authors examined the use formal specification methodology in the domain of the naval avionic safety-critical software systems. The software system of this project involved an avionic system that manages the operation of certain military aircrafts when such aircrafts are operating below a specified altitude. On this project, a cost reduction strategy was applied in the definition of the system requirements. On this project, the formal specification strategy involved the use of development tools for consistency checking that was conducted regularly to ensure that this property was always present. Because of the implementation of this strategy, it is assessed that the application of formal methods to the development of this safety-critical system the desired outcome of system requirement completeness was achieved.

4. CONCLUSION

The first conclusion drawn from the survey of the various application of formal specification techniques, notations, and methodologies to the development, verification, and validation of system specifications is that it is a critical and essential practice. In the case studies of this survey, the application of formal specification strategies at various stages of the development life cycle had contributed positively to the development of these safety critical software systems. The application of formal specification techniques to the derivation, validation, and verification of system requirements has resulted in system requirements that are more consistent and complete, and less ambiguous. Formal specification techniques have been successfully applied to multiple software design and development issues, such as concurrency, redundancy, and scheduling. From the survey, it has been determined that formal specification techniques, is also helpful in the development of safety-related operating systems and their implementation. The work in this report, surveyed the use of formal specification notations and methodologies in the development of safety-critical software systems in the industrial areas of avionics, medical equipment, and the railway system. From the analysis of the multiple works surveyed, it is concluded all demonstrated success and multiple benefits from the application of formal specification techniques. From the survey work, it is asserted that safety critical software system requirement definition, design, and implementation can benefit from the inclusion of formal specification techniques in the development life cycle. A further assertion is that the verification and validation of the system requirements also benefits from the application of formal specification techniques. Overall, the application of formal specification techniques increases the confidence in the reliability of the delivered software system, as it adds unrealized safety features and attributes to the system. A more trustworthy system is produced when the system is formally specified and verified. Conversely, the absence of formally specified requirements may lead to a less reliable system and in the domain of safety critical systems; this would be unacceptable. The defining feature of safety-critical software systems is the potential harm to or loss of life if such systems fail. Thus, it is concluded that the adaption of formal specification methodology in the definition of safety-critical software system's requirements should be universally accepted.

5. REFERENCES

[1] Jones, C. 2018. Software Methodologies: A Quantitative Guide. CRC Press, Taylor & Francis Group, Boca Raton, FL.

[2] Bedrij, O. J. 1962. Carry-Select Adder, *IRE Trans. Electron*, 11, 3,(June 1962), 340-346.

[3] Wing, J. M., 1990. A specifier's introduction to formal methods. *Computer*, 23, 9, (Sept. 1990), 8-22.

[4] ISO/IEC 13568, 2002. *Information Technology: Z Formal Specification Notation - Syntax, Type, System, and Semantics* First ed. ISO/IEC.

[5] Martins, L. E. G. and Gorschek, T. 2017. Requirements engineering for safety-critical systems: overview and challenges, *IEEE Software*, 34, 4, (July/Aug. 2017), 49-57.

[6] Xiang, J.; Futatsugi, K.; He, Y. 2004. Fault Tree and Formal Methods in System Safety Analysis, *Proceedings. The Fourth International Conference on Computer and Information Technology (ICCICT)*, Wuhan, China, 1108-1115.

[7] Tay, K. M. 2009. On Fuzzy Inference System Based Failure Mode and Effect Analysis (FMEA) Methodology, *Proceedings of the 2009 International Conference of Soft Computing and Pattern Recognition, (SOCPAR)*, 2009. 329-334.

[8] Daramola, O.; Stalhane, T.; Sindre, G.; Omoronyia, I. 2009. Enabling hazard identification from requirements and reuse-oriented HAZOP analysis, *Proceeding of the Proceedings of the 2009 International Conference of Soft Computing and Pattern Recognition (SOCPAR)*, 3-11.

[9] Martins, L. E. G. and Gorschek, T. 2011. *A method for evaluating rigor and industrial relevance of technology evaluations*. Empirical Software Engineering. 16. (June 2011). 365-395.

[10] Lutz, R. R. 1993. Analyzing software requirements errors in safety-critical, embedded systems, *Proceedings of the IEEE International Symposium on Requirements Engineering*, California, USA 126-133.

[11] Hamilton D., Covington R., and Kelly, J. 1995. Experiences in applying formal methods to the analysis of software and system requirements, *Proceedings of the IEEE Workshop on Industrial-Strength Formal Specification Techniques*, Boca Raton, FL. 30-43.

[12] Bowen, J. 1993 Formal methods in safety-critical standards, *Proceedings of the Software Engineering Standards Symposium*, Brighton, 168-177.

[13] RTCA, *Software Considerations in Airborne Systems and Equipment Certification. DO-178C*, Radio Technical Commission for Aeronautics (RTCA), Washington DC, USA,. 2011.

[14] Jacky, J. 1995. Specifying a safety-critical control system in Z, *IEEE Transactions on Software Engineering*. 21, 2. (Feb. 1995) 99-106.

[15] Fang, Q., Zhang, C., Ye, X., Shi, J. Zhang, X. 2014. A new approach for developing safety-critical software in automotive industry, *IEEE 5th International Conference on Software Engineering and Service Science*, Beijing, 64-69.

[16] Bellairs, R. 2019 *What Is ISO 26262? An Overview*. Preforce.

[17] OSEK 2005. *OSEK/VDK Operating System*, version 2.2.3, OSEK.

[18] Jin Song Dong, N., Fulton, Zucconi L., Colton, J. 1997. Formalising process scheduling requirements for an aircraft operational flight program. *First IEEE International Conference on Formal Engineering Methods*, Hiroshima, Japan. 161-168.

[19] Budiyanto, I. B., Kistijantoro, A. I., Trilaksono, B. R. 2015 Formal verification of integrated modular avionics (IMA) health monitoring using timed automata. *International Seminar on Intelligent Technology and Its Applications (ISITIA)*, Indonesia. 291-296.

[20] Bharadwaj, R., Heitmeyer, C. 2000. Developing high assurance avionics systems with the SCR requirements method. *Proceeding of the 19th Digital Avionics Systems Conference.* 1. 1-8.