

Chapter 2

Formal Methods and Tools for Safety of Critical Systems

K.S. Kushal, Manju Nanda and J. Jayanthi

Abstract Advances in the quality of Safety-Critical Software Systems are very much essential in addressing the correctness, safety and security attributes of the system. The development processes of such critical systems are imperative at corresponding stages in accomplishing its key attributes. The use of formal methods and tools coupled with formal verification techniques presumes explicit definition of system and its properties which meets the specifications. A meticulous mathematical notation used to represent the critical systems at early stages of their development process is the substratum of Formal Methods. Model checking, a formal verification technique, encompasses specification and modelling languages that improve the overall software architecture. This paper describes various tools at different phases of development process of Safety-Critical Systems, aiding formal methods and verification techniques in software practices. Also we present a unified-framework architecture which integrates various such tools used in the development of Safety-Critical Systems and their underlying software.

Keywords Formal methods • Formal verification techniques • Model-checkers
Specification languages

K.S. Kushal (✉) · M. Nanda · J. Jayanthi
Aerospace Electronics & Systems Division, CSIR-National Aerospace Laboratories,
Bengaluru, India
e-mail: ksk261188@gmail.com

M. Nanda
e-mail: manjun@nal.res.in

J. Jayanthi
e-mail: jayanthi@nal.res.in

2.1 Introduction

The use and application of mathematical techniques, Formal Methods, in exploring the systems relative behaviour is benefitted with predictive and accurate computation of properties of the system. Safety-Critical Systems are such systems which exhibit discrete behaviour and state diversifications, with abrupt transitions enabling the state changes successively. Formal Methods defy this distinct behaviour of these systems and predicts the system properties with the help of mathematical models. Such models also ensure in improvising the quality attributes in developing the system. This in turn increases the confidence in achieving highly integrated software. The formal methods mandate the use of formal specifications and models using formal languages during the development life cycle. The formal specifications govern the development stages such as design, architecture, implementation, coding, verification, validation and maintenance. These provide an accurate means of predicting and analysing the behaviour of the system. Application of these in the early stages of design and development life cycle of the software system provides corrective and preventive measures in their specification for the determination of their satisfiability.

With the use of formal methods and its techniques a compelling capability of improving the quality of the software being developed is provided, by the mathematical models/theorems [1]. During recent times, with the advances in the complexity of the hardware and the software counterparts of the Safety-Critical Systems, a rigid and a robust technique is very much required during its development phase in their entire life cycle. Various qualitative metrics are to be considered in analysing the impact of the application of formal methods from that of the conventional methods. Quantifying the software of Safety-Critical Systems in applications such aerospace, security applications (such as Net-Centric Warfare and Cyber-Physical Systems), pose a great challenge in assessing their qualitative metrics for functionality, safety and security. Safety, reliability and security are the major concerns in engineering an integrated module for Safety-Critical application. The consideration of these criticalities early in the development phase provides substantial foundation and enhances the efficacy of the application of formal methods and their supported tools. In addition to these methodologies, formal verification techniques yield affirmative solutions in deciding the critical decisions and theories involved in the design and development of Safety-Critical System. These further induce a sense of confidence in the practical developmental procedures of the software counterparts for Safety-Critical Systems.

In this paper, Sect. 2.1 gives an introduction about the need for Formal Methods. Section 2.2 provides an insight into the pre-work about the formal methods and tools for Safety-Critical Systems, while Sect. 2.3 gives an approach adopted in presenting a unified-framework architecture for the application of formal methods in Safety-Critical Systems. Section 2.4 deals with the conclusion and future scope of the work presented in this paper.

2.2 Literature Survey

2.2.1 *Formal Methods-Based Database—Intelligent Knowledge Database (IKD)*

Over a decade, several processes aiding the enthusiasts, researchers and developers in retrieving the information related to formal methods and formal verification techniques have proven to be a major impediment. These kinds of information retrieval systems and engines needs to be intelligent and flexible in making the search options and the process of recovering the necessary information much simpler and user friendly. Such systems handles humongous amount of data pertaining to formal methods. Dr. (Ms.) Nanda et al. proposed an intelligent, integrated, unified and efficient intelligent knowledge database system [2]. This system served the purpose of retrieving the information from a single source. This is very much essential in enabling the analysis of safety critical applications wherein the formal methods and the formal verification techniques are applicable. The IKD tool acts as an extensible knowledge-based tool which provides with the crucial information pertaining to formal methods, their applicability and the tool support comprehensively. This tool is very robust with the segregation of the information related to these methodologies and its applications at abstract phases of the development life cycle. This tool basically makes use of the keyword search engine [3] as proposed by Monika Henzinger in retrieving the Web Information. Thus, this tool is alike encyclopaedia for various formal methods and formal verification techniques, as well as the supported tools/tool chains along with their application differentiated under a broad classification set.

In the due course of design and development of such a knowledge database [4], it was surveyed that there tends to be a skeptical view about the usefulness of these methodologies and techniques. Also the need for a unified platform wherein such methodologies and techniques along with their respective tool support can provide the necessary artifacts supporting the need to consider such methodologies. This in turn enhances the safety-critical system's software counterpart expectations as desired.

2.2.2 *Development of Tool Related and Tool Applicability Metrics*

The advancements in the software counterparts of Safety-Critical Systems are flourishing. This has resulted in an increase in the functionalities being embed in the software. This is as a result of a substantial increase in the number of electronic components within avionics systems. As an impact of these there is an increase in the automation required to address the criticality that are involved in their development life cycle. All these factors yield a lighter-greener aircraft with state-of-art technologies, control and functionalities. The association of formal methods and

formal verification techniques with the adherence to the standards has found acceptance for evolution, over a wide spectrum of methodologies, tool and techniques which can formally meet the specifications as desired.

A one point solution to this problem is to use normal conventional techniques and methodologies. These techniques which are aided by their respective tool/s for specific phases in their developmental life cycle are utilised. The analysis of the results at each stage in their development life cycle—i.e. requirement, architecture, design, code and testing are performed to ensure the software safety and reliability of certain Safety-Critical Systems. This proves to be more tedious and costly in producing the desired quality of safety-critical software [5]. From the rigorous analysis, it was evident that the conventional techniques were less effective and efficient because of the below factors:

- Highly person dependent—varies from individual to individual (interpretation of the specifications)
- More effort was required in terms of time and money

These factors are derived by the type of tools, their capabilities, operating environments, feature-set and their engines based on the applications for which they were used. Also the metrics generated by these tools or tool-set is also evaluated under various scenarios in order to conclude that conventional methods or techniques are not best suited for such kind of applications. Using formal methods and techniques, associated tools were found to provide with much better quality output. This plays a very crucial role in instigating a high level of confidence in assuring the correctness and completeness of the software. It was also found that the compliance to the specifications was formally proved. The metrics such as completeness, property violations, verification of the critical specifications and its properties, and validating the output with that of the specifications [1] with the generation of test cases were evaluated.

2.2.3 Development of Process Related Metrics

The processes in the development life cycle for safety-critical software are the phases relevant to their development process, as shown in Fig. 2.1. These are broadly classified into the following:

- i. **Requirements Capture and Management:** The specification of the requirements for the design and development of the software desired for the Safety-Critical System. These specify the required functionality that is desired for the Safety-Critical System to be embedded with.
- ii. **Architecture Design:** The capture of the requirements and interpretation of the entire software as blocks, with the behavioural metrics such as data flow representation and control flow representation.

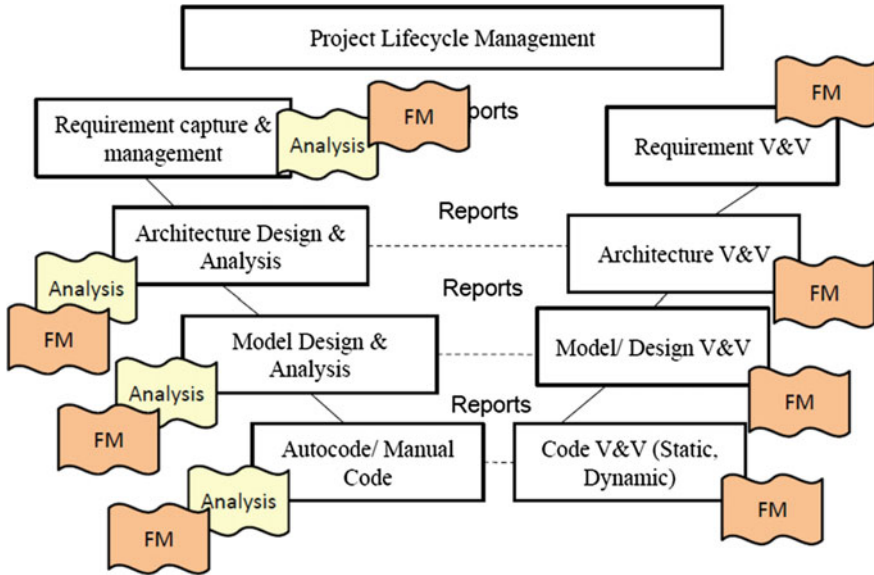


Fig. 2.1 Software development life-cycle—V Model

- iii. **Model Design:** The implementation of the software behaviour with the specification of required attributes and necessary arguments. This also includes the mechanisms and the objectives that conform to the actual specifications.
- iv. **Auto-code/Manual Code:** The annotation of the attributes, arguments with the specifications, written usually using a high-level language (C, C++, JAVA, Ada) in representing the desired system. This is either manually generated or automated from the design or sometimes from the architecture design.
- v. **Testing (Verification and Validation):** The verification of the annotation with the inclusion of pre and post conditions in validating the system performance. This can also be either automated or manually written with suitable constraints as assertions, flagging the time consumption in the process. This is dependent on the type of compilers used and the language in which the code is generated (if automated) or written (if manual).

2.3 Approach

With the application of conventional Software Development Life-Cycle (SDLC), there is zero automation and it requires manual intervention at each individual and relevant phases. Also there is no application of formal methods and techniques in conventional SDLC. This proves less effective with the necessity for lot of manual effort, time and cost. The evolution of this particular methodology resulted with

partial automation of the SDLC as Model-Based SDLC. This process stands ahead with more effectiveness as the manual effort required is comparatively less (only required at suitable phases, with application of formal methods and techniques also at certain phases), and also cost and time effective. The complete automation of the entire SDLC with the integration of various formal methods and techniques, tool/s that support formal methods and techniques are included. This provides a seamless transition from one phase to another. This also ensures the correctness and completeness of each phase. This is basically dependent on the tool or sets of tools used while automating the entire process.

The evaluation of metrics generated from the inherent phases of SDLC and the tool/s support at each phase of development, suggests a need for migration from no automation process (conventional SDLC) to complete automation. This helps in imbuing a higher confidence level at relevant phases of development of software for a particular Safety-Critical System. The exclusive implementation of formal methods and formal verification techniques in either of the processes will adhere to one of the standards in their field of application such as Aerospace (RTCA DO-178B/C [6, 7], ARP 4761 [8]), Industrial (IEC 61508 [9]), Military (MIL-STD-254 [10]), with the assurance of a seamless integration of various tools and techniques that are based on formal methods.

2.3.1 RTCA DO-178B/178C Software Development Life-Cycle

The Aerospace standard RTCA DO-178B/178C SDLC includes the following set of details at each phase, which are necessary for software to be qualified. This is without the supplements and there is no mandate as for the application of formal methods and techniques. There are two phases [6]:

1. System Life-Cycle Phase
 - i. Planning Process
 - a. Plan for Software Aspects of Certification
 - b. Software Development Plan
 - c. Software Configuration Management Plan
 - d. Software Quality Assurance Plan
 - e. Software Verification Plan
 - ii. Development Process
 - a. Software Requirements Data
 - b. Software Design Description
 - iii. Verification Process

- a. Hardware Software Integration Test Plan
- b. Low Level Integration Test Plan
- c. HIS Verification Cases and Procedures
- d. Verification Results

2. Certification Phase

- i. Software Accomplishment Summary
- ii. Software Configuration Index
- iii. Software Life-Cycle Environment Configuration Index
- iv. LLI Test Report
- v. HIS Test Report
- vi. Traceability Matrix—Traceability of System Requirements to Software Requirements and Software Requirements to Test Cases.

The above-mentioned phases are in coherence with the RTCA DO-178B workflow. With the inclusion of formal methods and the process of automation, the standards have been evolved with additional supplements being provided with DO-178C as follows:

1. Tool Qualification (RTCA DO-330)
2. Model-Based (RTCA DO-331)
3. Object-Oriented (RTCA DO-332)
4. Formal Methods (RTCA DO-333)

The standards mentioned above mandates the use of formal methods and techniques accordingly depending upon the type of the software being developed.

The work in this paper concentrates on automating the SDLC and provides a seamless integration between various tools either developed in-house or commercially off-the-shelf tools. This also aims at integrating various formal techniques such as formulae, theorem provers, model checkers. The approach is defined based on various tools or set of tools that are supporting formal methods and techniques, at different phases of the development process of the software for Safety-Critical Systems. This forms a basis for the development of a unified-framework architecture that provides a single platform to integrate various formal methods-based tools and techniques and also handle the process of SDLC with much ease. This is done with the development of expertise over the appropriate tools at the appropriate phases by means of case studies. Also the metrics generated by these tools or techniques at each phase, for the safety application, are analysed by means of case studies. Each case study to substantiate the SDLC process is being followed with RTCA DO-178B Level A Criticality.

At each phase of the SDLC in the unified-framework architecture, the software is subject to extensive formal methods. The application of the formal methods and techniques in the unified-framework architecture monitors the process at each phase substantially supported by the analysis also based on formal methods and techniques at their respective phases. The dependency of each with their subsequent phases is addressed by this framework. This also provides a backward traceability

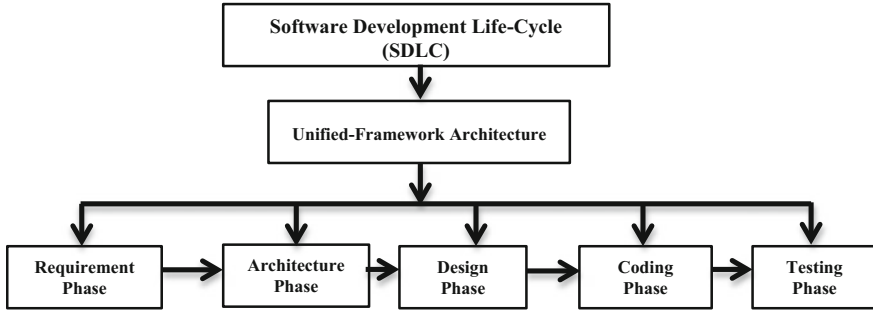


Fig. 2.2 Unified-framework architecture

and compatibility with the phases in the development life cycle, as shown in Fig. 2.2. The safety analysis of the software is addressed in the analysis of the software at its relevant phase. The unified framework also supports the conventional and partially automated workflows apart from the complete integrated automation in order to compare and evaluate the metrics. The framework also provides the standard guidelines and also mandates the application of criticality levels at suitable phases and applications.

For example; the safety analysis at the architecture phase is analysed with the help of FTA (Fault-Tree Analysis) [11], and FMEA (Failure Mode and Effect Analysis) [12], with the help of formal tools such as OpenFTA [13] and OSATE [14] respectively. The metrics generated by these tools are evaluated to establish the robustness and also include high-level confidence in the development of the software.

2.4 Conclusion and Future Scope

The analysis of the metrics generated from various tools or set of tools, their evaluation led to the need for the development of a unified formal guidelines, metrics for safety and security applications. This is needed in order to ensure the necessity for such applications and at CSIR-NAL we are developing a unified-framework architecture to seamlessly integrate tools and techniques, proving their applicability. As a part of the future work, design and implementation of the unified-framework is being carried out. The metrics are generated with extensive case study from the unified-framework for benchmarking the formal methods-based tools and techniques. This activity is also being carried out at CSIR-NAL. The improvisation of these benchmarks will also be handled in future by taking case studies from various

other safety applications such as medical, military. It is also planned to integrate the safety semi-formal method-based techniques as a part of this unified-framework.

Acknowledgements The authors would like to thank the Director, CSIR-NAL, Bengaluru for supporting this work.

References

1. Heitmeyer C (2005) Developing safety-critical systems: the role of formal methods and tools. In: Proceedings of the 10th Australian workshop on safety related programmable systems, Sydney, Australia, 25–26 Aug 2005, pp 13–29
2. Nanda M, Jayanthi J, Madhan V (2012) Intelligent knowledge database (IKD) tool for formal methods. *Int J Soft Eng Appl (IJESA)* 3(6):117–127. doi:[10.5121/ijesa.2012.3609](https://doi.org/10.5121/ijesa.2012.3609)
3. Henzinger M (2000) Tutorial: web information retrieval. *IEEE Proceedings of 16th international conference on data engineering*. 29 Feb–03 Mar 2000, San Diego, CA, pp 693. ISBN: 0-7695-0506-6, ISSN: 1063-6382
4. Dondossola G (1998) Formal methods in the development of safety critical knowledge-based components. *Proceedings of the european workshop on validation and verification of knowledge-based systems, CEUR workshop proceedings*. 06–08 June 1998, Povo, Trento, Italy, pp 01–12. ISSN: 1613-0073
5. Place PRH, Kang KC (1993) Safety-critical software: status report and annotated bibliography. Technical Report, CMU/SEI-92-TR-5, ESC-TR-93-182, June 1993
6. King T (2012) Reusing certified, safety-critical avionics software. In: 2012 IEEE/AIAA 31st digital avionics systems conference (DASC). 14–18 Oct 2012, Williamsburg, VA, pp 6A1-1–6A1-6. ISBN: 978-1-4673-1699-6
7. RTCA DO-178B/C, The Aviation Golden Standard, <http://www.rtca.org>
8. ARP 4761—Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment <http://standards.sae.org/arp4761/>
9. IEC 61508 – functional Safety Standard for Electronic Safety related Systems <http://www.iec.ch/functionalsafety/>
10. MIL-STD-254 <https://www.document-center.com/standards/show/MIL-STD-254>
11. Joshi A, Vestal S, Binns P (2007) Automatic generation of fault trees from AADL models. In: Workshop on architecting dependable systems (DSN '07), Critical Systems Research Group
12. Grunske L, Han J (2008) A comparative study into architecture-based evaluation methodologies using AADL's error annex and failure propagation models. In: *IEEE computer society, 2008 11th IEEE high assurance systems engineering symposium*, pp 283–292. ISSN: 1530/2059/08
13. OpenFTA—Advanced Tool for Fault Tree Analysis <http://www.openfta.com/>
14. OSATE—Open Source AADL Test Environment <http://osate.github.io/>

Formal Methods for Safety and Security
Case Studies for Aerospace Applications

Nanda, M.; Jeppu, Y. (Eds.)

2018, XII, 132 p. 42 illus., 24 illus. in color., Hardcover

ISBN: 978-981-10-4120-4