Chapter 2
Background

Abstract Formal methods based system development is considered as a promising approach to develop the safe critical systems. This chapter discusses the standard safety life-cycle, traditional safety analysis techniques, traditional system engineering approach, standard design methodologies and safety standards that are used for developing the critical systems. Furthermore, we have given a list of successful industrial case studies based on formal techniques. Moreover, we discuss the role of medical device regulations. Finally, this chapter shows the usability of formal techniques for developing the critical systems and to motivate for developing a new methodology, and associated techniques and tool in the context of medical device development, which are covered in the remaining chapters.

2.1 Introduction

Critical systems are tremendously grown in functionality in both software and hardware, and due to increasingly the complexity of critical systems it is very hard to predict the absence of failure. Moreover, some of these failures may cause catastrophic financial loss, time or even human life. One of the main objectives of software engineering is to provide a framework to develop a critical system that operates reliably despite this complexity. It has been shown in [97, 113] that the promising results are achievable only through the use of formal methods in the development process. More than a decade, several formal methods based techniques and tools are used by industries and academic research projects [62, 111]. The backbone of formal methods is considered to be mathematics, which often supports related techniques and tools based on logico-mathematical theory for specifying and verifying the complex systems. The techniques and tools based on formal methods provide a certain level of reliability under some constraints. Formal verification is considered as a benchmark technique, particularly in the area of safety critical systems, where important safety properties are required to prove rigorously before implementing a system. However, the use of formal methods helps to speculate the hidden peculiarity of a system like inconsistencies, ambiguities, and incompleteness.

In the past, formal methods based technique was not into practice in the software development life-cycle due to the use of complex mathematical notations; inadequate tools support and too hard to apply. Special training was required to use

N.K. Singh, *Using Event-B for Critical Device Software Systems*,
DOI 10.1007/978-1-4471-5260-6_2, © Springer-Verlag London 2013

formal methods to apply in the system development process. Increasingly, number of successful development of techniques and tools related to the formal methods, the industries have started to adopt it for verifying the safety properties of complex systems [13, 14, 23, 97]. For verifying a critical system, industries prefer to use formal methods-based techniques such as model checking or theorem proving in place of the traditional simulation techniques. In both areas related to the model checking and theorem proving, the researchers and practitioners are performing more and more industrial-sized case studies [9, 11, 13, 24, 38, 61, 62, 78], and thereby gaining the benefits of using formal methods.

This chapter briefly discusses safety critical systems, examines the use of formal techniques to provide safety and reliability, analysis the use of traditional safety techniques for software, surveys on regulations for medical devices, and gives a list of successful industrial case studies based on formal techniques. Reliability and safety are the most important attributes of critical systems. The main objective of this chapter is to provide information about current safety issues in medical domain particularly for the safety critical software systems. It should be noted that the formal methods are the most important techniques that are applicable for a safety related software development for medical devices using several classical safety analysis techniques.

2.1.1 Structure of This Chapter

This chapter contains a concise survey that reviews the existing literatures relating to the development and analysis of a software for safety critical systems, which identifies current valuable approaches for developing the safety critical software, and reviews the methods and analysis techniques available to the system developers. Section 2.2 gives an overview about reliability and safety. Section 2.3 presents a role of a software in safety-critical systems and Sect. 2.4 describes safety life-cycle for critical systems. Section 2.5 presents traditional safety analysis techniques. Section 2.6 explores the traditional system engineering approach, and Sect. 2.7 gives a list of standard design methodologies for the system development process. Section 2.8 depicts about safety standards, and Sect. 2.9 presents medical device standards and discusses the current issues of regulations. Section 2.10 presents a list of industrial projects related to the formal methods, and finally, Sect. 2.11 discusses the use of formal methods for the safety critical software systems.

2.2 Reliability and Safety

2.2.1 Reliability

Reliability is a fundamental attribute for the safe operation of any critical system. According to the Institute of Electrical and Electronic Engineers (IEEE), "*Reliability is the ability of a system or component to perform its required functions under*

stated conditions for a specified period of time" [54]. Reliability can be used for prediction, analysing, preventing and mitigating failure over time of a complex critical system. In the context of safety, there are several elements of reliability. These elements are operational reliability and performance reliability. Operational reliability can estimate the probability of failure of a system, while performance reliability measures the adequacy of features to successfully perform under the specific conditions. Reliability analysis aims to protect a system from failures of its components, software and hardware [67].

A fundamental challenge in reliability analysis is the uncertainty for failure occurrences and consequences. To protect a system, a quantitative approach has been pushed forward for the design, regulation and management of the safety of hazardous systems. The reliability assurance is a process that is considered by manufacturers during product development according to the regulating standards [18, 22, 33, 54, 58]. The reliability is quantified in terms of probability. Reliability has a time oriented characteristic that can be expressed as the Mean Time Between Failures (MTBF) [95]. When we use probability or characteristics of the underlying life distribution to measure reliability, it must be emphasised that reliability is a relative measure of the performance of a system. It is relative to the user requirements, system failures, expected lifetime of the device, operating environment conditions, system functionality and behaviour of the system changes with time.

Reliability engineering is a function to calculate the expected reliability of a system, process and behaviour in advance. The main objective of reliability engineering is to deliver reliable product in order to satisfy behaviour requirements, safe operation, lower cost, and to maintain company reputation [95]. Nowadays, reliability engineering is a well established discipline that can provide an integration of formal methods to investigate the system requirements, correctness of the system by addressing the following questions: (1) why a system fails? (2) how to develop a reliable system? (3) how to measure the reliability of design, process and operation of a system? and (4) how to maintain system reliability during system operation through fault diagnosis and prediction [17, 116].

2.2.2 Safety

Safety can be defined as "*freedom from those conditions that can cause death, injury, occupational illness, or damage to or loss of equipment or property, or damage to the environment*" [83]. Safety can provide some standards to ensure quality and functionality of a system. The safety standards eliminate all potential risks that can cause loss of life, injuries or property damage. Critical systems that meet certification standards, are safe to use in practice. It provides confidence to the user to use for their purpose in daily life.

Safety is like reliability that concentrate on the designing phase of a system. A system must be designed for safety. System safety is an engineering and management discipline that encapsulates human, machine, environment, designing, testing,

operating and maintaining system to achieve acceptable risk within the timing and cost constraints in the system life-cycle [56]. Hazard analysis can improve the safety that defines real or potential conditions that can cause injuries, illness, loss of system, property or damage environment.

2.2.3 Safety vs. Reliability

As a conventional approach, it is assumed that a reliable system is safer and vice versa. However, it is not always true and it can lead to a lot of confusion to analysis a system failure. Actually, it is often true that the safer system can be less reliable. For example, an inoperative elevator can provide maximum level of safety. The inoperative elevator cannot do any functionality like opening or closing the door, moving up or down, after pressing any button. To use the elevator, in this state is always safe, but the reliability of the elevator is zero. The inoperative elevator has not any functionality, it is absolutely unreliable and ineffective to use for moving up or down to different floors. To improve the safety of a reliable system, system designer introduces some elements to add the functionalities. Such as, designers can introduce elements and controls for moving up or down of the elevator. These new elements can reduce the reliability of the elevator. Such that, a sensor can provide a proper opening or closing door operation. If the sensor is out-of-order, then the elevator will not move. Here, the sensor behaviour reduces the reliability and increases the safety of the system.[1]

Reliability and safety are the main attributes to determine effectiveness of a system, where effectiveness is influenced by the life-cycle activities related to the design, manufacturing, use and disposal of the product [22]. IEC 60513 [50], fundamental aspects of safety standards for medical electrical equipment, provides a safety standard for developing the medical systems that assures the basic safety and essential performance. IEC 60601 [52] address reliability stating that "*reliability of functioning is regarded as a safety issue (for life-supporting equipment) and where interruption of an examination or treatment is considered as a hazard for the patient*."

According to the FDA [33] regulation safety is defined as: "There is a reasonable assurance that a device is safe when it can be determined, based upon valid scientific evidence, that the probable benefits to health from the use of the device for its intended use and conditions of use, when accompanied by adequate directions and warnings against unsafe use, outweigh any probable risks." Effectiveness is defined thus: "There is a reasonable assurance that a device is effective when it can be determined, based upon valid scientific evidence, that is a significant portion of the target population, the use of the device for its intended uses and conditions of use, when accompanied by adequate directions to use and warnings against unsafe use, will provide clinically significant results" [94].

[1] http://www.aldservice.com/en/safety/what-is-safety.html.

2.3 Software in Safety-Critical Systems

Software is a vital part of any system, especially in embedded systems, where it is used to control the whole functionality of the systems. The embedded systems have major role to control the behaviour of the safety critical systems. When we use these systems, we consider that their risk has been minimised and uses of the systems are effectively safe. The system is not only safe, but we also expect other attributes like reliable and cost effective. Main safety-critical systems are commercial aircraft, medical care, train signalling systems, air traffic control, nuclear power, and weapons, where any kind of failure can quickly lead to human life in danger, loss of equipment, and so on. The industries are responsible for designing and delivering the safety-critical systems according to the standards authorities [18, 33, 54, 58], which satisfy the requirements.

To address the problem of system's failure related to the software errors for example, overdoses from Therac-25 for treating cancer through radiation [74], the overshooting of the runway at Warsaw airport by an Airbus A320 [79], Intel Pentium floating point divide [91], 5000 adverse events for Insulin Infusion Pump (IIP) reported by FDA [114, 115] and Ariane 5 flight 501 going off [76]. All these problems and many more are considered as a part of the "software crisis". The term "software crisis" has been introduced in late 1960s to describe the failures of the systems in which software-development problems cause the entire system [36]. In 1968, a meeting is organised by NATO related to the software crisis. This crisis had as its root cause the problem of complexity brought about in many cases by sheer length of programs combined with a poor control over how each line of code affects the overall system. Almost three decades later, this problem still remains as indicated in [36].

Software crisis is a well-known problem for other engineering disciplines, and over the years of experience has been accumulated to provide effective solutions: the technology has been available, and it has been shown to work with a very high degree of confidence. Software are using frequently in the system development, which is also classified as an engineering discipline, so it would seem natural that one can apply the insights and quickly surmount any hurdles. However, it is true that the engineering insights are applicable to modern the critical-system development to come over the traditional approaches of the system development.

2.3.1 Software Safety and Reliability

Increasing size and complexity of software in critical systems, the software has a primary threat for the reliability. Most of the reliability engineering techniques address failures in hardware components. Software architecture analysis methods concentrate to analyse the quality and behaviour of a system at the early stage of the system development. Several useful reliability engineering techniques are available in literature to analyse and design a reliable system. A comprehensive survey of

these techniques is given in [70, 77]. Software quality has been promoted in the software architecture analysis domain. The software architecture is an important process that helps to predict important qualities of a system and to identify the potential risks [29]. To provide an early reliability analysis that covers software components, it is advantageous to utilise both results from software architecture analysis and conventional reliability analysis approaches [101].

According to the IEEE, software safety can be defined as *"freedom from software hazard," where software hazard is defined as "a software condition that is a prerequisite to an accident," and an accident is defined as "an unplanned event or series of events that results in death, injury, illness, environmental damage, or damage to or loss of equipment or property"* [54]. The use of formal methods in software development process provides safety assurance that the software does not show any failure cases. There are several techniques that are used to identify the software bugs at the early stage of the system development. Each phase of the software development is verified and validated using several techniques from requirements analysis to code generation [54, 55].

International regulatory standards provide guidelines for designing, operating and maintaining the critical systems [48]. To analyse the reliability, the hardware and software barriers must take into account. However, hardware barriers are more reliable than the software barriers according to the past history of the system functionality in terms of performance, proof-checking, and regress testing of the hardware components [32]. In a complex system, self-test are not sufficient to identify potential failures. Therefore, proof-checks are used to perform at regular intervals to cope with undetected hardware failures.

The hardware systems are subject to ageing and wear. Ageing and wear characteristics of the hardware systems provide a way to calculate the reliability using MTBF. However, the software systems are not applicable to use statistical technique like MTBF for reliability calculation, because software systems are not subject to ageing and wear. Tools and techniques related to the software failures are not similar to the hardware failures due to different characteristics of both software and hardware systems. The software systems do not follow the physical laws of degradation or failure as per the hardware systems [116].

The software reliability is an important challenge in the area of safety critical systems, where software may be used to control the hardware components. The software failures can be identified using software-centric approach and system-centric viewpoint. The software-centric approach looks for failure modes and to evaluate their probabilities, and the system-centric viewpoint is based on practical observation related to the specifications and requirements, which encapsulate software design failures.

The fault injection method is a technique for quantitative analysis of the software failure that deliberately inject faults in the software and count the number of times that the software maintains its function in spite of the injected fault [1, 46, 105]. However, this approach is not effective to discover all hidden failures. Hence, another feasible approach to building the reliable software is to use the systematic software development process. The main objective is to evaluate different fault tolerant approaches throughout the software development process [116].

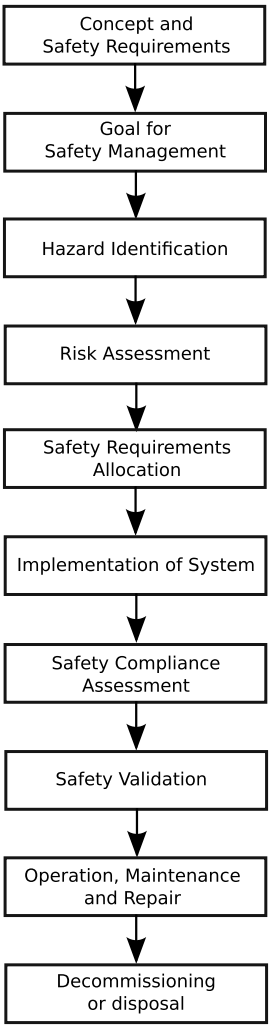


Fig. 2.1 Safety life-cycle (adapted from [53])

2.4 The Safety Life-Cycle for Critical Systems

Safety is a most important system property, that should be methodically analysed along the system life-cycle. A number of standards and recommended practices define the processes and the objectives of the safety life-cycle, such as IEC 61598 [53], MODEF [30]. Figure 2.1 depicts a stepwise implementation of the system development safety life-cycle. The main objective of this development cycle is to guide system designers and developers in what they need to do in order to claim that their systems are acceptably safe for their intended uses. The purpose of the overall safety life-cycle is to force safety to be addressed independently of functional issues, thus overcoming the assumption that functional reliability will automatically produce

safety [53, 92]. This development cycle is accepted by all industry sectors in developing the advanced safe critical systems. The life-cycle phases are briefly described as follows:

- The initial concept phase is used to identify the functional requirements of the system, related environment where the system will be operated, and possible design approaches for developing the system.
- The second phase is used to set the goal for management and technical activities to consider the safety implications of the developing system through assessing the required safety level to ensure that the system achieves and maintains the required level of functional safety. The goal should be produced at the beginning of system life-cycle and it must be reviewed at regular interval.
- In Phase 3, hazard identification process is applied to identify the possible hazards, which might arise during construction, installation, operation, maintenance and disposal of the system. This hazard identification process is applicable throughout the system life-cycle. The main formal techniques for hazard analysis are FHA, FTA, FMEA and HAZOP.
- Risk assessment process is used to identify a set of possible risks through analysing the identified hazards, and check against tolerability criteria. A set of actions must be taken to reduce the overall risks. The action can be decided under consideration of possible consequences of hazards to a tolerable level. The risk assessment process helps to discover possible requirements for the safety integrity level for the system.
- The safety requirements are separately assessed for different parts of the system and the whole system is reviewed to ensure that the risk will be reduced to an acceptable level and system is safe in use. Any critical system is too complex in functionality. To implement the safety functions, a simple technology should be used to avoid the overall complexity of the system.
- This phase of the safety life-cycle is related to system implementation, where safety related parts or components are implemented to satisfy the safety requirements.
- Assessment of the specific components or parts of the system must comply with the safety requirements to ensure that the component of the system meets the given safety requirements. The assessment process is based on analysis and auditing techniques.
- Safety validation phase is used to verify the system against the claimed safety properties. This process assures that the system have been achieved a set of goals and system is safe to use in practice. Moreover, during the verification process arising problems are also resolved.
- This phase of safety life-cycle related to the system operation and maintenance, which ensures that the system will be safe during the maintenance process. Various safety related system problems arise due to a poor maintenance process. Thus the system must be designed for maintainability. The use of the system in different environment should also be analysed to evaluate the system behaviour and must ensure the safety of the system.

- Finally, safety considerations that may apply during decommissioning should also be taken into account. Thus an assessment of the impact of the decommissioning should be made on both the components and the process of the system. This process will use hazard and risk assessment approaches to determine the level of safety-related work. The safety related work must be satisfied during the decommissioning activity of the critical system.

2.5 Traditional Safety Analysis Techniques

Safety provides protection from hazard to human life, the environment or property. There are not such a magical thing that can guarantee for absolute safety. However, a system can be enough safe that can accept any risk related to the life, environment or property. The risk can be measured through probability and the complex calculations of a system, while a system can be failed due to use of any harmful substances in the process. However, software is not a harmful substance. Software can be used to control the system behaviour using a set of processes. Moreover, the software can contribute to safety, e.g. through control over hazardous physical processes [72]. Software hazard and safety analysis refer to the process of assessing and to make contribution to design a safety software. According to [81], four safety-relevant elements of a system development process are defined as follows:

1. Identifying hazards and associated safety requirements.
2. Designing the system to meet its safety requirements.
3. Analysing the system to show that it meets its safety requirements.
4. Demonstrating the safety of the system by producing a safety case.

2.5.1 Hazard Analysis

Software development life-cycle and engineering techniques are used to design and develop a system to meet all the functional requirements. These techniques place a little effort to examine failure cases of a system. However, a highly critical system like aviation, medical or automotive needs to consider all possible failure scenarios to avoid from any hazard. Different kinds of techniques may be employed for safety assessment from hazard analysis. When a system has many components, then take a modular approach for analysing a system using System Hazard Analysis (SHA) and Subsystem Hazard Analysis (SSHA). The SHA discovers all associated hazards of a system, while the SSHA discovers how an operation of a particular component affects on the whole system.

The SHA and SSHA analyses are performed by several techniques, which are provided by the standard authorities. Traditional safety analysis techniques such as Hazard and Operability study (HAZOP) [92], Functional Hazard Assessment (FHA) [109], Fault Tree Analysis (FTA) [73], and Failure Mode Effects Analysis

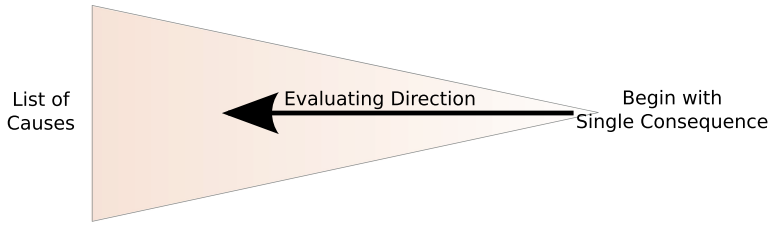


Fig. 2.2 FTA—Evaluating back from consequence to cause

(FMEA) [31] are standards to apply for hardware intensive systems that are also applicable for the software systems. Traditional safety analysis therefore begins by defining the hazards associated with a system, determines their severity, and then attempts to identify the factors that can initiate the hazards. These safety analysis techniques provide a rigorous way to examine the causes and their consequences of the identified hazards.

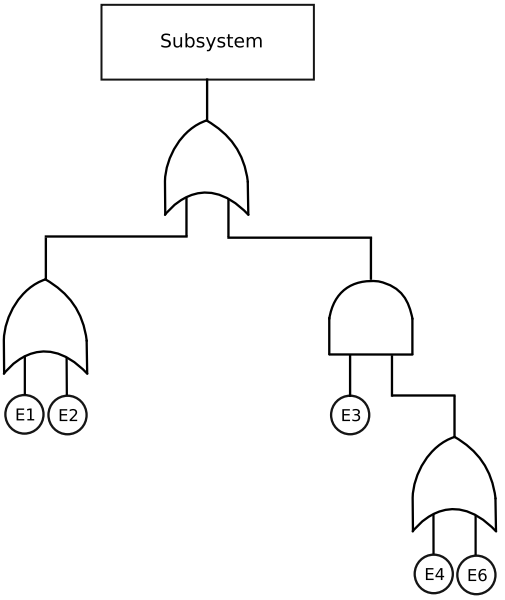
Functional Hazard Assessment (FHA)

Hazard are unfavourable conditions that a system should avoid to occur or must be identified in advance. Once the hazards are known that it becomes possible to trace backwards from the hazards to the particular events that can cause them. Functional Hazard Assessment (FHA) is used to identify such type of hazards that can be occurred because of functional failure. The safety analysis techniques concentrate on defining the required functionality and analysing the consequences of failures. The FHA is an informal process that is used to document hazards and determine their severity. The FHA produces a list hazards in tabular form with different degree of severity [109].

Fault Tree Analysis (FTA)

Where a system is self-contained, having its boundaries well defined, one focuses on the hazards that are internal to the system, which may be termed faults. Thus, a fault is always a hazard, but not conversely. At this level, we have another technique to analyse the systems using Fault Tree Analysis (FTA) [73]. The FTA is a safety analysis technique that is deductive and top-down method of analysing system design and performance to identify all the possible failures or errors. It is based on a feedback process that can start with a system level hazard and try to discover backward for identifying all the possible causes of hazards (see Fig. 2.2). The FTA shows a list of hazards according to the hazard level. Although, the FTA has limited use for identifying the faults of a system using a visual technique that can trace higher level events down to their contributing events in form of failures, errors or faults. The FTA is represented in a tree structure that shows various factors to contribute a high

Fig. 2.3 FTA tree



level event. The fault trees can also be used in a confirmatory role where they are particularly useful in showing that a probability requirement for a hazardous failure mode has been met by the system. Figure 2.3 depicts a basic architecture of the FTA. In this figure, highest level event (hazard) is traced backward to identify the source of errors or faults. Events and gates in fault tree analysis are represented by symbols. The source of errors or faults are known as the base events (errors) [73].

Failure Mode Effects Analysis (FMEA)

Failure modes and effects analysis (FMEA) is a step-by-step approach [31] to identify the possible hazards in a complex system that facilitates the identification of potential problems in the design or process by examining the effects of lower level failures. The FTA safety analysis technique is based on top-down approach, while the FMEA is used a bottom-up approach. In this bottom-up analysis, the technique determines possible failures of a system and produces a list of probable failures according to the degree of severity. The feed-forward technique of the FMEA is used to discover possible failures or errors through forward tracing (see Fig. 2.4). FMEA is useful for evaluating a new process prior to implementation, and for assessing the impact of proposed changes on the existing processes. The output of FMEA presents in a tabular form that describes the failure modes, in which something might fail, and the consequences of those failures.

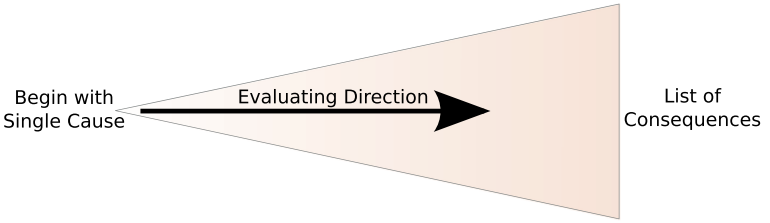


Fig. 2.4 FMEA—Evaluating forward from cause to consequences

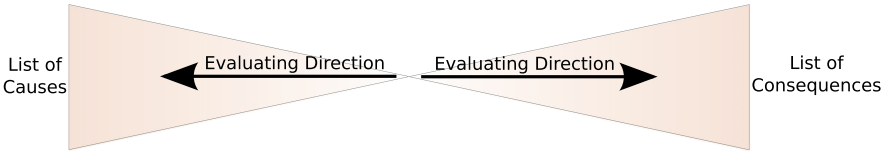


Fig. 2.5 HAZOP—Evaluating from the fault in both directions for causes and consequences

Hazard and Operability Analysis (HAZOP)

Hazard and operability studies (HAZOP) are more commonly used at the broadest level for analysing process plants like chemical and nuclear industries [57]. The HAZOP supports the chemical process industry, takes a representation of a system and analyses how its operation may lead to an unsafe deviation from the intent of the system [57] with special attention to the environment of operation. This technique is very popular in industries because it aims to predict possible failures, and identify their impact.

HAZOP [92] is a most prominent formal technique for identification of the hazards. This technique examines all the essential components and their interconnections of a system to explore the possible causes of errors and their consequences. Particularly, HAZOP is a powerful technique for exploring the interaction between parts of a system. HAZOP is based on a theory that assumes risk events are caused by deviations from design or operating intentions. Identification of such deviations is assessment and generally facilitated by using a set of "guide words" as a systematic list that includes process, and deviation perspectives. HAZOP starts to analyse in both directions, backwards to explore its possible causes, and forwards to examine its consequences (see Fig. 2.5).

A set of safety analysis techniques like FHA, FTA, FMEA, and HAZOP, is used to identify a list of base events that can contribute to hazardous conditions. A list of events gives the general categories of safety properties required to the requirement model of a system. A more detailed discussion of the system hazard analyses (SHA) with the software perspective is provided in [57, 72].

2.5.2 Risk Assessment and Safety Integrity

A risk assessment is simply a careful examination of the past data related to the hazard's analysis for the similar systems; from the reliability assessments of components of the system being developed; and other sources. The outcome of the risk assessment presents some kind of gradation and may be expressed in terms of what constitutes a tolerable and intolerable risk. This outcome results help for regulating industrial risk, and to determine whether a risk is unacceptable, acceptable or somewhere in between. Lots of factors are used for determining the risk based on quantitative and qualitative analyses [8]. Using a risk classification of accidents according to the frequency and severity usefully serves as a relatively simple basis for its determination.

Assessment of a risk can decide a necessary level of safety that can be achieved from various functions of a system. This is an issue of safety integrity, which is defined as, "*Safety integrity is the likelihood of a safety-related system achieving the required safety functions under all the stated conditions within a stated period of time*" [108]. The system activities are contributing to the integrity may be characterised by two kinds of requirements:

1. Generation of the new safety requirements of a system is resulting from the design and development.
2. Ensuring that what is being built meets the requirements that have already been specified.

Here, the first requirement is related to the requirement analysis and hazard analyses of a system. The second requirement is related to the reliability engineering techniques, whose consideration may have to be sustained throughout the development as the design evolves with modification to interfaces, rearrangement of components or other kinds of changes. To apply the several techniques like FHA, HAZOP, FMEA and FTA for the fault prediction, fault removal, fault avoidance and fault tolerance, and to achieve the system integrity require together with methods and design of the system, are the main resources for measuring the system reliability [103].

A safety of a system may be simply characterised by a process of reducing risks to appropriate effect. The main objective of a qualitative or quantitative risk assessment is to establish the level of tolerability for any identified risk. If a risk falls in between the states of 'intolerable' and 'acceptable' then any risk must be reduced to 'as low as reasonably practicable'. This is known as the ALARP principle as illustrated in Fig. 2.6. The width of the triangle is proportionate to the level of risk and thus also to the amount of resources that can be justified to reduce it. A comprehensive survey of risks and safety integrity is provided in [8].

2.5.3 Safety Integrity and Assurance

Finally, there is always a question in the development of critical system, "What is the assurance level according to the certain level of integrity of the system?". In

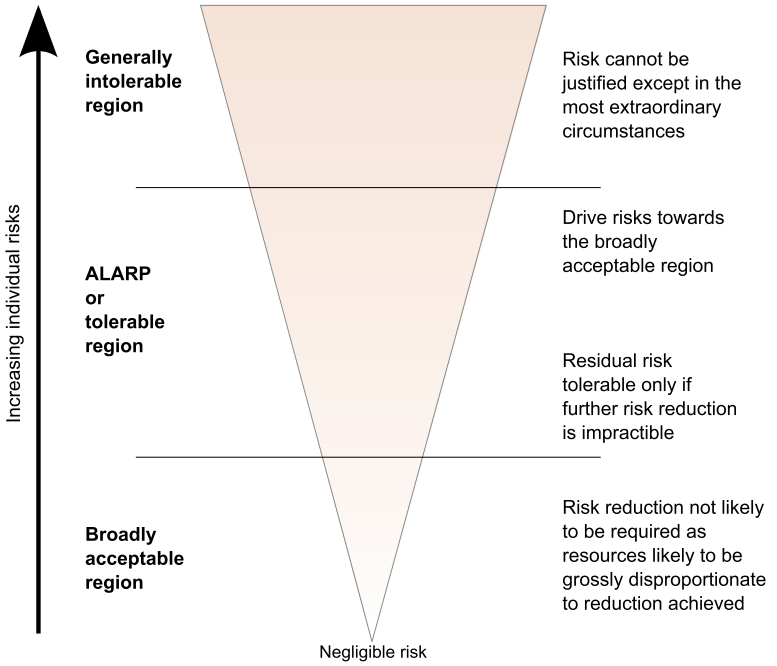


Fig. 2.6 ALARP model of risk level

order to safety assurance of the developed system may be certified as safe, there must be a set of documents, which provides detail justification of the safety. This document contains a list of all hazard's cases with log details and various arguments for indicating that how the system has reached at the required safety levels. The safety case brings in all the aforementioned risk analyses, risk reductions and other integrity and reliability measures, often presenting various statistical evidence. It is a considerable huge amount of a task involves lots of documentation. A software SAM (Safety Arguments Manager) is recognised to support this process and allows to manage all the developing safety cases [82].

2.6 Traditional System Engineering Approach

A critical system uses a standard life-cycle to achieve a certificate from the standard authorities [18, 33, 54, 58]. A system can be considered safe if all the hazards have been eliminated, or the risk associated hazards have been reduced to an acceptable level. Software is a part of a system, which is used within the system to operate the system safely. The integrated software within a system does not show any kind of misbehaviour. However, if the same software is used by multiple systems then the software must have similar behaviour in each system. However, sometimes it is not true. It is believed that each system is different, with different requirements, different

risk level with different hazard's characteristics, it is impossible to know if software is safe without considering the behaviour of the software as a part of the system which it is controlling. Therefore, when considering the process for developing a safe software, it is crucial that the whole system of which the software is a part is considered, as well as the software itself [12].

2.6.1 The Software Safety Life-Cycle

In the past several years, different types of software development life-cycle have been identified. All of them have their own merits and limitations according to the problem complexity, size and type of the system. This book will not enter into a discussion about different life-cycle process models. A detailed description about each life-cycle process model is available in [4, 80, 90, 99]. Here, we only discuss about life-cycle process model related to the safety critical software system.

In recognition of the distinctive nature of safety-related systems, there is a standard development process known as V-model, which is widely accepted by large companies and defence. It is an extension of the standard Waterfall model [4, 8, 98, 108]. The V-model represents a software-development process, where the process steps are bent upwards after the coding phase to form the typical V shape. The V-model presents the relationships between each phase of the development life-cycle and its associated phase of testing. V-model is also called verification and validation model (V & V). This process uses a very intensive testing for removing bugs or errors, which may appear during any stage of the system development.

The typical process of developing a safety-critical software system is generally time-consuming. Most of the development processes are based on the V-model, which is illustrated diagrammatically in Fig. 2.7. This model identifies the major elements of the development process and indicates the structured, and typically sequential, nature of the development process. The sequential nature of development is generally considered essential for reasons of managing communication and scale, for scheduling different phases and disciplines, for managing traceability (which is mandated by relevant safety standards) and for the certification purposes.

In order to produce a safety-related software according to this framework, various techniques are recommended. These include the application of structured analysis techniques to generate a visible modular construction (the principles of modularity are expounded in [89]), and diversity in design, implementation and maintenance to avoid faults due to common mode failures. Many such techniques are very widely applicable, and although they are usefully brought into the safety-critical context, there is not so much literature devoted solely to their use in this specific area. Nevertheless, material is available: for instance, there have been reviews such as [28, 103] to help designers and managers as to the suitability of mainstream programming languages for the safety-critical systems.

Safety requires a lot of integrity, and this is recognised in the safety life-cycle model which separates the specification of safety requirements into purely func-

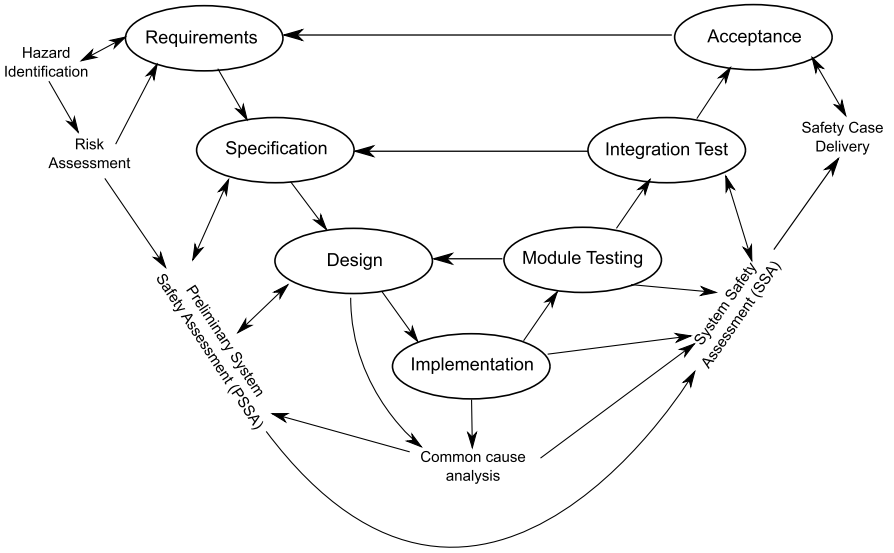


Fig. 2.7 The V model of safety-critical system development

tional requirements and safety-integrity requirements. The safety integrity requirements are calculated individually for each of the functions previously identified. Having done this, one may concentrate on providing the high levels of assurance on the safety-critical aspects. We intend using the safety life-cycle model as a basis, with a view to ascertaining its suitability to support the production of formal models with high integrity. Our contention is that we treat carefully the non-functional requirements and to put forward a selection of viewpoints and methods highlighting further the safety concepts, which are often subtle, then the life-cycle model can be effective [103]. A safe system can be characterised as one in which risks from hazards have been minimised throughout a system life. The process of providing hazard analyses and risk assessments are thus crucial activities to ensure the safety of a system.

In Fig. 2.7, Preliminary System Safety Analysis (PSSA) and System Safety Analysis (SSA) are the collection of various techniques like FTA, HAZOP, FMEA, etc. The aim of all these techniques is to identify failures and derive the safety requirements, which prevent from the occurrence of the hazard. FTA focuses on the different components of a system, while HAZOP focus on the flow between components. There are also a number of other techniques, which are used in the PSSA for analysing failures, an overview can be found in [87].

2.7 Standard Design Methodologies

A design is a meaningful engineering representation of a higher-level interpretation of a system, which is actually a part of an implementation in a source code. Design

process is traceable using reverse engineering technique to the actual stakeholders requirements. The quality of a system can be assessed through predefined criteria for a good design. Analysis and design methods for software have been evolving over the years, each with its approach for modelling needs a world-view into software [86]. The following methodologies are common, which are used in current practices.

- Structured Analysis and Structured Design (SA/SD)
- Object Oriented Analysis and Object Oriented Design (OOA/OOD)
- Formal Methods (FM) and Model-based Development

SA/SD techniques provide means to create and evaluate a good design of the systems. This technique covers functional decomposition, data flow and information modelling. OOA/OOD considers the whole system into abstract entities called objects, which can contain information (data) and have associated behaviour. It is in practice from last 30 years, which is used in several big projects. It contains Object-Oriented Analysis and Design (OOA/OOD) method, Object modelling Technique (OMT) Object-Oriented Analysis and Design with Applications (OOADA), Object-Oriented Software Engineering (OOSE) and UML. Formal Methods (FM) and Model-based development are a set of techniques and tools based on mathematical modelling and formal logic that are used to specify and verify requirements and designs for the systems and software [86]. Formal method is also a process that allows the logical properties of a computer system to be predicted from a mathematical model of a system by means of a logical calculation. Formal methods can be used for formal specification, formal verification and software models (with automatic code generation) [86].

2.7.1 Design for Reliability

Reliability is an attribute of a system that is derived from research, concept and design through analysing the capacity and performance under the working environment. The reliability level can be established during design phase of the system development. However, a subsequent testing and production cannot improve the reliability without any modification in the basic design. Design reliability techniques integrating with the development process for assuring the safety of a system. Reliability becomes a difficult design parameter due to the increasing complexity and limited knowledge of the system requirements. If reliability is an important attribute of a system then it is quantified during specification of the design requirements.

Reliability is essential for a healthcare and medical devices, which need to be safe and effective. Medical device manufacturers and regulating bodies like the Food and Drug Administration (FDA) [33] and Center for Devices and Radiological Health (CDRH) [22] have a responsibility for assuring the safety and effectiveness of medical devices. The CDRH has standards to analyse system specification, design requirements, and usability of a system. The CDRH [22] requires a complete

and accurate requirements of any medical system for designing and manufacturing a safe system. The CDRH allows premarket review to identify relevant information for processing, manufacturing, assembly handling, maintenance and disposal of the system. Moreover, the CDRH also seeks to determine if the manufacturer has captured the important aspects of the development life-cycle for producing a product [59, 66]. However, the CDRH is also concerned with potential users like patient or clinician, who will use the device. FDA requires product performance to be verified [59, 60, 66] and validated [59, 66]. The FDA supports the performance and safety assessment of a system through providing the evidence that the system is adequate to use in practice. The FDA regulatory oversight of the manufacturing process through the Quality System Regulation [59, 66].

Increasing complexity and safety recalls in the medical systems advocate a new approach for a good design for reliability (DFR) in the medical industries [42]. DFR describes the tools and techniques that can support product and process design to ensure the system reliability. The DFR is a process that spans the entire product development cycle from concept to release of a product. The DFR [42] indicates the following paradigms that are essential to design a complex medical system:

1. Spend significant effort on requirement analysis
2. Critical failure is not an option for medical devices
3. Measure reliability in terms of total Life-cycle cost
4. Don't just design for reliability, design for durability
5. Design for prognostics to minimise surprise failures

2.8 Safety Standards

It is perhaps best to start by considering the various standards that exist for industries, which develop the safety critical systems. Standards are documented agreements containing technical specifications, which produce precise criteria, consistent rules, procedures to ensure reliability, software processes, methods, products, services and use of products, are fit for their purpose in this world. Standards include a set of issues corresponding to the product functionality and compatibility, facilitate interoperability including designing, developing, enhancing, and maintaining. A set of protocols and guidelines, which are produced by the standards, are consistent and universally acceptable for the product development. The standards allow to understand the quality of different products for competing with them, and provide a way to verify the credibility of a new product [22, 54, 58].

Verification and validation (V & V) are part of the certification process for any critical system. There are several reasons, why certification is required for any critical system. For example, medical device like a cardiac pacemaker must obtain a certificate before to use in practices. Certification of the product not only assures about the safety, but also helps to a customer to gain confidence to buy and to use the product, which is also important for commercial reasons like having a sales advantage to industry. Certifications are usually carried out by some national and

international authorities. Certification can be applicable to an organisation, tools or methods, or systems or products. The main objective of the certification bodies is to provide assurance that an organisation can achieve a certain level of proficiency, and that they agree to the certain standards or criteria. In the case of product certification, there are always issues for the certification, whether a methodology or development process is certified or not.

There are many international standards bodies. More than 300 software standards and 50 organisations are developing software standards [34]. Standards come in many different flavours, for example, de-facto standards, local, national and international standards. Some of the standards are more specific related to the defence, financial, medical, nuclear, transportation, etc. (see the Appendix).

There are number of standards addressing safety and security of a system related to the software development. For example, avionics RTCA-Do-178B [96] or the IEC 61508 [35, 53] as the fundamental standard for the functional safety of E/E/EP systems [35, 53]. The IEC 62304 [51] standard is for the software life-cycles of medical device development that addresses to achieve more specific goals through standard process activity, and helps to design the safe systems. All the necessary requirements for each life-cycle process are provided by the IEC 62304. The process standard IEC 62304 [51] is a collection of two other standards ISO 14791 and ISO 13485, where the ISO 14791 standard is for quality, and the ISO 13485 is for risk management.

Institute of Electrical and Electronics Engineers (IEEE) standards [54] provides a safety assurance level for industries, including: power and energy, biomedical and health care, information technology, transportation, nanotechnology, telecommunication, information assurance, and many more. The IEEE standard is approved by authority and considers the users recommendations before apply into the development process. All these standards are reviewed at least every five years to qualify the new amendments in the systems.

Food and Drug Administration (FDA) [68] is established by US Department of Health and Human Services (HHS) in 1930 for regulating the various kinds of product like food, cosmetics, medical devices, etc. The FDA is now using standards in the regulatory review process to provide a safety to the public before using any product. The FDA provides some guidelines on the recognition to use of and consensus standards. The FDA is interested in the standards because they can help to serve as a common yardstick to assist with mutual recognition, based on the signed Mutual Recognition Agreement between the European Union and United States. The FDA standard classifies the medical devices based on risk and the use of medical devices. The FDA provides some standard guidelines for the medical devices, and the medical devices require to meet these standards. Time to time lots of amendments have been done in the FDA standards [33, 68] according to the use of medical devices to provide a safety.

Common Criteria (CC) [18] is an international standard that allows an evaluation of security for the IT products and technology. The CC is an international standard (ISO/IEC 15408) [58] for computer security certification. CC is a collection of existing criteria: European (Information Technology Security Evaluation Criteria

(ITSEC)), US (Trusted Computer Security Evaluation Criteria (TCSEC)) and Canadian (Canadian Trusted Computer Product Evaluation Criteria (CTCPEC)) [19–21]. The CC enables an objective evaluation to validate that a particular product or system satisfies a defined set of security requirements. The CC provides a framework for the computer users, vendors and testing organisations for fulfil their requirements and assures that the process of specification, implementation and testing of a product has been conducted in a rigorous and standard manner.

There are several ways to tackle the complexity issues of software, which major the software at industrial scales and usability of the software. The Software Engineering Institute, funded by the military, has produced a Capability Maturity Model (CMM) [90] by which may be assessed the quality of management in a software engineering team. The CMM broadly refers to a process improvement approach that is based on a process model. A process model is a structured collection of practices that describe the characteristics of effective processes; the practices included are those proven by experience to be effective. The CMM can be used to assess an organisation against a scale of five process maturity levels. Each level ranks the organisation according to its standardisation of processes in the subject area being assessed.

2.9 Regulations for Medical Devices

All kinds of medical products have to comply with national or international regulatory bodies that can provide safety assurance to use the medical products. The pathway from product design to the final product is often unclear and number of challenges and questions increase as medical device become more complex. The regulating bodies cover the essential requirements to regulate the standards of safety and performance of the medical devices. Medical device manufacturers agree to follow medical device development standards to provide the life-saving technologies to patient without compromising in safety with low cost.

The past decades shows several recalls related to the safety issues in the medical devices [63]. Everyday lots of defects are reported by consumers that are a serious consequence due to medical device failures. Faults in medical devices, such as pacemakers, defibrillators, artificial hip, and stents, have caused severe patient injuries and deaths. In 2006, FDA reported 116,086 device related injuries, 96,485 malfunctions, and 2,830 deaths; a more recent independent analysis claims there were 4,556 device-related deaths in 2009 [45, 63]. These recalls have raised many questions related to the device development process, designing and testing tools, and resources are adequate to ensure that the developed device are safe and secure to use in practice. However, the adoption of medical regulations has increased the rates of infant mortality, life expectancy, and premature and preventable deaths all over the world.

2.9.1 Device Classification

The Food and Drug Administration (FDA) has classified all the medical devices into three classes based-on the safety and effectiveness level [93]. The safety and effectiveness levels are categorised in the low, medium and high risks, respectively. Device classification determines different types of regulatory requirements that must be followed by the medical device manufacturers.

Class I

Class I devices are sufficient to provide reasonable assurance of the safety and effectiveness of the device with minimal potential for harm. The devices of this class are simpler than the Class II and Class III. These devices are subject to only general controls. Manufacturer registration with the FDA, good manufacturing techniques, branding and marking of the products are the main issues that are covered under the general controls [93]. These general controls are sufficient to provide safety and effectiveness of the devices. Class I devices are exempt from the premarket notification and the FDA determines low risk of illness or injuries to patient [93]. Class I devices include tongue depressors, bedpans, elastic bandages, examination gloves, and hand held surgical instruments and other similar types of common equipment.

Class II

Class II devices more complex than Class I devices, and the general controls of the Class II are insufficient to assure safety and effectiveness. To provide such assurances, additional methods are required [93]. Class II devices are also subject to special control in addition to the general controls of Class I. Special controls may include standard performance, labelling requirements and premarket review to reduce or mitigate risk. Class II assures that the used devices will not because of injuries or harm to patients. X-ray machines, powered wheelchairs, infusion pump, surgical drapes, surgical needles, suture material and acupuncture needles are the main devices of this class.

Class III

Class III devices have insufficient information to assure safety and effectiveness solely through the general and special controls that are sufficient for Class I and Class II devices [93]. In addition of the general controls of Class I, premarket approval and a scientific review are needed to ensure the safety and effectiveness of the Class III devices. Class III devices are described as those for which “insufficient information exists to determine that general controls are sufficient to provide reasonable assurance of its safety and effectiveness or that application of special controls

that can provide such assurance and if, in addition, the device is life-supporting or life-sustaining, or for the use of substantial importance for preventing impairment of human health, or if the device presents a potential unreasonable risk of illness or injury" [93]. Class III includes devices which are life-supporting or life-sustaining, and devices which present a high or potentially unreasonable risk of illness or injury to a patient. Class III includes complex devices like heart valve, breast implants, implanted cerebral stimulator and cardiac pacemaker.

2.9.2 Regulation Issues

Development in the area of medical devices is rapidly changing. Over the last 25 years, medical devices have evolved from analog to digital systems. In the current development, microprocessor, software, smart sensor and actuator are the main components of medical systems. Most of the medical devices are based-on embedded real-time system. The functionality of these complex systems is mainly based on software to provide robustness, safety and effectiveness. An embedded system may be used for special-purpose computer system to perform any particular task due to resource limitation. The life of medical devices has decreased due to more rapid innovation in enabling technology and demand for the more robust systems. Increasing complexity of the medical systems has raised many recalls. Regulating bodies are used to control the quality of medical devices and to provide safety in use. The current development techniques and existing tools are not sufficient to provide assurance to use any medical device. Due to failure cases and constraints in exiting approach, the regulating bodies have offered several research challenges in the area of medical device development. The following challenges provide a framework for thinking about the main issues of current medical regulations [22, 33, 110]:

- A new platform and implementation technologies is required to support science- and engineering-based design, development, and certification to analysis the quality of advanced medical devices and new emerging technologies.
- Software based on medical devices must be validated according to the state of the art taking into account the principles of development life-cycle, risk management, validation and verification.
- Simulation based closed-loop modelling is required to evaluate the medical devices.
- Use quantitative analysis to evaluate a risk and to identify the safety issues of medical devices.
- Preventing from a malicious malfunction of software of the medical devices, and handling the emerging issues for information security and privacy.
- To provide a protection against emerging infectious diseases and terrorism.
- To use a formal methods-based design techniques to develop the medical devices.
- Developing a new approach to use clinical data in evaluating medical devices.
- Development of the robust, safe and sustainable medical devices with low manufacturing cost with increasing quality and performances.

2.10 Industrial Application of Formal Methods

This section surveys previous works related to the critical system development. A common theme in much of this work is to use formal methods. Formal methods provide numerous tools and techniques for solving the different kinds of problems. Mainly formal methods are applicable for verification and validation of a system. Formal methods are used to verifying the specification of a system. Although the safety-critical systems have got the confidence in the development due to use of formal methods, such techniques are applicable in a wide variety of application areas in industry. Formal methods have been used to improve the quality of the system as well as verifying the correctness of a system at an early stage of the system development. A set of examples that pioneered the application of formal methods, to more recent examples that illustrate the current state of the art. Here, we have given a list of industrial applications, where formal methods have been used in the projects. A detail survey of all these projects is presented in [13, 14, 23, 97].

2.10.1 IBM's Customer Information Control System

A successful application of formal methods was the verification of the Customer Information Control System (CICS) in 1980, which was collaborated between Oxford University and IBM Hursley Laboratories [49]. The overall system contains more than 750,000 lines of code. Some part of the code was produced from Z specifications, or partially specified in Z, and the resulting specifications were verified using a rigorous approach. Some tools, related to the type checking and parsing were developed during the project, which were used to assist the specifier and code inspector. More than 2000 pages of formal specifications were developed for verifying the system. Measurements taken by IBM throughout the development process indicated an overall improvement in the quality of the product, a reduction in the number of errors discovered, and earlier detection of errors found in the process [23]. Furthermore, it was estimated that the use of formal methods reduced 9 % of the development cost for the new release of the software.

2.10.2 The Central Control Function Display Information System (CDIS)

The Center Control Function Display (CDIS) System was delivered from Praxis to the UK Civil Aviation Authority in 1992 for London's airspace as a new air traffic management system [39]. The CDIS system consists of fault tolerant architecture of a distributed network, where more than 100 computers are linked together. Formal methods were used at various levels of the system development. The requirements analysis phase was represented by formal descriptions using structured notations.

The VDM [10] tool was used for specifying the whole system, which specified concurrent system behaviours. At the product design level, the VDM code was refined into more concrete specifications, and a lower level code was formally specified and developed using CCS [85]. The productivity of the system was better than the traditional system development and the quality of the system was improved through finding some faults.

2.10.3 The Paris Métro Signalling System (SACEM)

The SACEM system [44] was developed by several industrial partners GEC Alsthom, MATRA Transport and CSEE (Compagnie des Signaux et d'Entreprises Électriques) in 1989. The system was responsible for controlling the RER commuter train system Paris. The existing system was made of embedded software and hardware, where software had 21000 lines of code. Some parts of the SACEM software were formally specified in the B modelling language [2] for the proving purpose. The SACEM project is an example of "reverse engineering" process, where formal specification and verification were conducted after developing the code. Finally, the system was certified by the French railway authority.

ClearSy has developed the screen door controllers for Paris metro line using B formal methods [71]. The models are developed using correct by construction approach and to prove the absence of failure in the system behaviour. A constructive process was used during system specification and design leads to a high-quality system.

2.10.4 The Traffic Collision Avoidance System (TCAS)

Formal specification of the Traffic Collision Avoidance System (TCAS) [15] is another interesting example of the application of formal methods in the air-traffic transport domain. The TCAS system is used by all commercial aircraft for reducing the chance of a mid-air collision. In early 1990s, a safety critical system research group at the University of California, produced a formal requirements specification for the TCAS due to occurring some flaws in the original TCAS specification. The formal specification was developed into Requirements State Machine Language (RSML) [75], which is based on a variant of Statecharts [40]. The original specification was not supported by existing formal methods tools, but nevertheless, it was very useful for the project reviewers, in the sense of improving the original specification. Heimdahl et al. [43] successfully checked the consistency and completeness of the TCAS specification and provably-correct code generated from the RSML specification.

2.10.5 The Rockwell AAMP5 Microprocessor

The microcode of AAMP5 microprocessor was formally specified and verified, which was produced by Rockwell [84]. This project was undertaken by Collins Commercial Avionics (CCA) and SRI. The AAMP5 microprocessor has a complex architecture, designed for Ada language and implements floating-point arithmetic in the microcode. PVS theorem prover [26] was used for specifying and verifying the microcode of the AAMP5 instructions.

2.10.6 The VIPER Microprocessor

VIPER microprocessor was developed with a simple architecture, specifically for the safety critical applications [27]. Formal methods were used throughout the development cycle of VIPER, at the different level using different techniques. This work was conducted by the Royal Signals and Radar Establishment (RSRE). Some parts of the system were specified by the HOL theorem prover and LCF-LSM language [37]. Mainly top level specification and abstract level view for register transfer level were carried out in the HOL. There was not any significant result through this formal verification except finding some minor flaws in the system, which had no concerns for the fabricators of the chip.

2.10.7 INMOS Transputer

In 1985, a microprocessor manufacturing company INMOS starts to use the formal program specification, transformation and proof techniques for designing a microprocessor. Formal methods based techniques were used for designing or developing the components of the INMOS Transputer. Different types of formal techniques like, Z, Occam and CSP were the main tools for specifying system requirements. For example, the Z specification language was used to specify the IEEE Floating Point Standard, and the combined approach of Z and Occam was used to design the scheduler, for the microprocessor. Later, the CSP with other formal techniques were used in design and verification of new features on the third generation Transputer (T9000), Virtual Channel Processor (VCP). The VCP is a device that allows several logical connections between two processors that was implemented by a single physical connection. This successful application of formal methods offers to apply into a hardware engineering environment [25].

2.10.8 The Mondex Electronic Purse

In this section, we have mentioned the Mondex Electronic Purse as a significant example of the use of formal methods in an industrial-scale application. The Mondex

Electronic Purse [16, 100, 112] is an electronic system for e-commerce, based on smart card, produced by NatWest Development Team. Electronic purse must ensure the security of each transaction. Formal methods were used by a several group of researchers for verifying the protocol of money transfer over an insecure and lossy medium. The whole formal specification of the Mondex system was developed and proved from an abstract model to concrete model using refinement approach. The abstract model was focused specially on the safety properties of the system.

2.10.9 Darlington: Trip Computer Software

This case study describes the computerised shutdown system of Darlington Nuclear Generating Station (DNGS). The shutdown application contains two independent systems, Shutdown System One (SDS1) and Shutdown System Two (SDS2). The SDS1 is operated by dropping neutron-absorbing rods into the core; the SDS2 is operated by liquid poison injection into the moderator [25, 107]. The Trip computers are connected with plant sensor to shutdown the system, whenever shutdown is required. This Trip computers are used alone to concern the safety issues. The shutdown systems were required a high level of confidence to obtain the certification standard. The regulatory bodies were not sure to check the validity of the software. Thus, the formal techniques were used to identify the discrepancies in the shutdown systems. The verification process was conducted on the complete system. The entire process is reported in [5]. The final system was redesigned or modified according to the regulators and concludes that the new develop system is safe for use.

2.10.10 The BOS Control System

The BOS Software is an automatic system, which is used to protect the harbour of Rotterdam from flooding, while concurrently also controls the ship traffic [104]. BOS controls a movable barrier, taking decisions of when and how the barrier has to move, based on chaotic behaviour of water level, tidal info, and weather preconditions. BOS is a highly critical system, which is characterised by IEC 61508 [53]. The design and implementation of the BOS were undertaken by CMG Den Haag B.V., in collaboration with a formal methods team at University of Twente. Different kinds of methodologies were applied during development of the system. Mainly formal methods were used to specify the crucial part of the system for validating the system specification. The control part of the system was formally specified in PROMELA and the data part into Z specification language. The formal validation of the design focused on the communication protocol between BOS and an environment. The final implementation of the system was done in C++ which was generated from Z specification. At the initial level of the system development, formal methods helped to uncover several issues in the existing system. Overall use of the formal methods improves the quality of the system.

2.10.11 NIST Token-Based Access Control System (TBACS)

Token-Based Access Control System (TBACS) is a smart card access control system that is based on cryptographic technique. This system was developed by US National Institute for Standard and Technology (NIST) [25], where they used formal techniques in order to verify all the essential safety properties. A set of permitted and prohibited actions were the main safety properties that were mainly focused on information access and transmission. These safety properties were formally expressed in mathematical logic using a set of invariants. In this development process, a theorem prover tool FDM was used for verification purpose. The FDM tool was very useful to identify a significant flaw related to the smart token that was easily removed without any excessive cost of the system development. The TBACS experiment provides a proper guidelines to satisfy related standards.

2.10.12 The Intel® Core™ i7 Processor Execution Cluster

Intel Core i7 processor [65] is used to verify using formal methods. The Intel Core i7 processor is a multi-core processor, where formal methods were used for pre-silicon validation of the execution cluster EXE, a component that is responsible for carrying out data computations for all microinstructions. The EXE cluster implements more than 2,700 microinstructions and supports multi-threading. The formal methods were used here to verify the data-path, control logic, and the state of the components. Formal methods based on symbolic simulation, and inductive invariants were used in the validation process of the processor. The significant contribution was of this project that the formal verification completely replaced traditional coverage driven testing and proved that the formal verification was a viable alternative approach for traditional testing techniques in terms of time and costs with respect to quality of the system.

Here, we have presented a list of projects related the critical system development using formal methods. All these projects have used different kinds of formal techniques for discovering the bugs at the early stage of the system development and have shown that formal methods could be a significant approach for verifying the systems. Formal method techniques are very expensive and hard to apply in the system development process due to complexity of mathematics and the limitations of existing tools [26, 64, 88]. Main limitations are, each tool based on formal method can be used for only specific purpose, and a formal model developer requires good experience to use formal methods and knowledge of related mathematics. To know the significant use of formal methods [13, 14, 23, 97] as well as handling its complexity, in this book, we propose a new development life-cycle methodology, where each step is based on formal techniques. In this context, we develop a chain of techniques and tools for supporting the system development life-cycle using formal techniques from requirement analysis to code generation.

2.11 Formal Methods for Safety-Critical Systems

This section presents the use of formal methods in the critical device system software development through providing some informal definitions of the main concepts.

2.11.1 Why Formal Methods?

Providing a high integrity system with the embedded software requires a careful argument for its justification. Demonstrating the requirements through sufficient statistical evidence based on testing, and other general reliability measures has been shown to be doubtful. Thus, some other kinds of arguments have to be written, which must be precise—in language that is well-defined, whose meaning is clear, and with the ability to prove statements without doubt. Since natural language is unable to fulfil such demands, the only possible solution is to use a mathematical approach—formal methods [103].

A formal approach is an ideal for verification, the activity guaranteeing correctness, that we are building the system right and particularly, that successive refinements of a specification are consistent with each other. More than that, the discipline which they encourage often leads to a more careful analysis of the most basic assumptions and definitions in the design, a benefit which is often understated [103]. In particular, they may point to ambiguities in the requirements' definition. Formal methods are thus effective for validation—making sure that we are building the right system [13, 47, 102].

The main objective of formal methods is to help developers to build the reliable systems. Formal methods is a cutting-edge technology for developing the critical systems, where high safety and security are required. Mathematics is a basic foundation for formal logic that provides some ways to discover potential errors at the early stage of the development. Figure 2.8 presents modified V-model after introducing formal methods in a development process. This figure shows that module testing and integration testing are not required due to formally verified system at the specification and design level. Formal methods help to reduce the burden of exhaustive testing, which are used by the traditional development. Formal techniques verify the whole system at the early stage of the system development during specification and design, and to prove the correctness of the system. We cannot say that formal method is a silver bullet, but it is more reliable than the other traditional development approaches. Now formal methods' techniques are feasible to apply for any larger and complex problems.

2.11.2 Motivation for Their Use

The use of formal methods is very limited in current industrial practice. It is mainly used for verification and validation of any specific part of the system. Specifically,

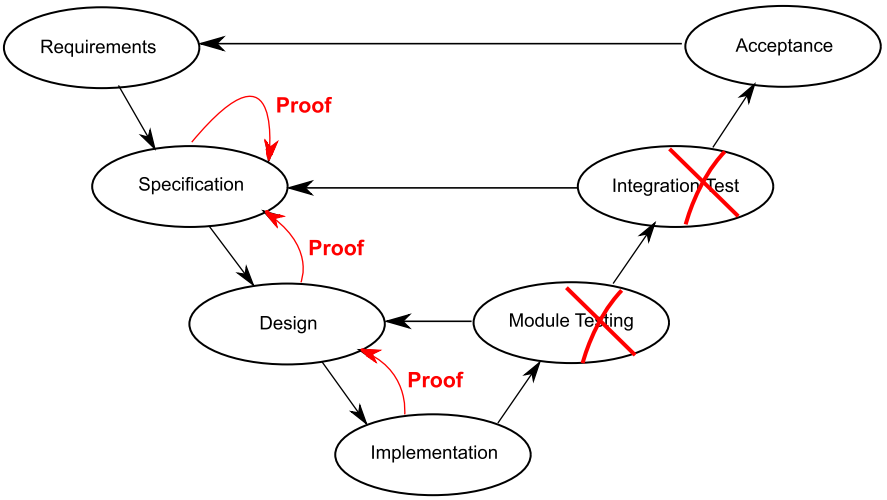


Fig. 2.8 The V-model of safety-critical system development using formal methods

it addresses that the formal methods are not well integrated into established critical system development processes. There are a number of reasons for this. First, the application of formal methods requires high abstraction and mathematical skills to write specifications and conduct proofs, and to read and understand formal specifications and proofs, especially when they are very complex. Second, existing formal methods do not offer usable and effective methods to use in the well-established industrial software process. There are lots of effective tools are available, which are crucial for formal methods application, but existing tools are not able to support a complete formal software-development process, although tools supporting the use of formal methods in limited areas are available in [26, 64, 88]. To make formal methods more practical and acceptable in industry, some substantial changes must be made.

This book proposes a development life-cycle and a set of associated techniques and tools to develop the highly critical systems using formal techniques from requirements analysis to automatic source code generation. In this context, we have developed a set of techniques and tools related to the Event-B modelling language [3]. Event-B modelling language is only used for verifying the part of a system. There is not a set of supporting tools, which can be used for the formal software development. The proposed techniques and tools have filled all missing tools and provide a rigorous framework for the system development process. The proposed approach is evaluated through a "Grand Challenge" case study, relative to the development of the cardiac pacemaker. This case study is related to the medical domain. Our main objective is to use this case study to show the effectiveness of our proposed approach and give the evidence that developed techniques and tools are applicable for any critical systems.

In this book, we have provided some possible solutions for the emerging problems in the area of software engineering related to the development of the critical

systems. We have captured some missing things in the existing tools related to the formal methods that are essentially required for developing any highly critical system. We have proposed a set of new techniques and tools to model the critical systems, which cover some set of weakness in the existing approach. No one method or tool can serve all purposes. From the experience, we have learnt what kinds of techniques can have the most impact. To be attractive to the practitioners, methods and tools should satisfy the following criteria, where we realise that some of these criteria are ideals, but it is still good to strive for them and some of the basic criteria [23] are required in the development of methods and tools:

1. Methods and tools should provide significant benefits for developing a system, when starting to use them.
2. Helps for writing clear, consistent and unambiguous specifications.
3. It should be possible to amortise the cost of a method or tool over many uses. For example, it should be possible to derive benefits from a single specification at several points in a programme life-cycle: in design analysis, code optimisation, test case generation, and regression testing. Moreover existing developed specification can be reused for other development processes.
4. Methods and tools should work in conjunction with each other and with common programming languages and techniques. Developers should not have to "buy into" a new methodology completely to begin receiving benefits. The use of tools for formal methods should be integrated with that of tools for traditional software development, for example, compilers and simulators.
5. Notations and tools should provide a starting point for writing formal specifications for developers who would not otherwise write them. The knowledge of formal specifications needed to start realising benefits should be minimal.
6. Methods and tools should support evolutionary system development by allowing partial specification and analysis of selected aspects of a system.

A new method or tool should have precise strengths and weakness, limitations, modelling assumptions and to support for ease integration with other technique's, etc. Clear selection criteria helps the potential users to decide what method or tool is most appropriate for the particular problem. Given that no formal methods technique is likely to be suitable for describing and analysing every aspect of a complex system, a practical approach is to use different methods in combination. Based on the results of the survey performed in this chapter it is possible to identify the contribution that this book makes. We have given our motivation for developing new techniques and tools as follows:

- *Development life-cycle methodology*: This is the heart of the book, which presents a methodology for the critical system development from requirement analysis to automatic code generation with standard safety assessment approach. It is an extension of the waterfall model [8, 108] with some rigorous approaches to produce a reliable critical system. This methodology combines the refinement approach with a verification tool, model checker tool, real-time animator and finally generates the source code using automatic code generation tools. This kind of approach

is very useful to develop the whole system using formal techniques and to verify the complex properties of a system and to discover the potential problems.

- *Environment modelling*: The most challenging problem is an environment modelling, for instance, to validate and verify the correct behaviour of a system model, requires an interactive formal model (an environment formal model). For example, a cardiac pacemaker or cardioverter-defibrillators (ICDs) formal models require a heart model to verify the correctness of the developed system. No any tools and techniques are available to provide an environment modelling to verify the developed system model. The main objective is to use formal approach for modelling the medical device and biological environment to verify the correctness of the medical systems.

  To model a biological environment (the heart) for a cardiac pacemaker or cardioverter-defibrillators (ICDs), we propose a method for modelling a mathematical heart model based on logico-mathematical theory. The heart model is based on electrocardiography analysis [7, 41, 69], which models the heart system at cellular level [106]. The main key feature of this heart model is the representation of all the possible morphological states of the electrocardiogram (ECG) [6, 7]. The morphological states represent the normal and abnormal states of the electrocardiogram (ECG). The morphological representation generates any kind of heart model (patients model or normal heart model using ECG). This model can observe a failure of impulse generation and failure of impulse propagation.
- *Refinement chart*: There are several ways to handle the design complexity of a system. Refinement technique is the most common approach, which facilitates to build a system gradually. We have discovered a very simple way to present the whole system based on operational behavioural using a refinement chart. The refinement chart is a graphical representation of a complex system using layering approach, where functional blocks are divided into multiple simpler blocks in a new refinement level, without changing the original behaviour of the system. The final goal to use this refinement chart is to obtain a specification that is detailed enough to be effectively implemented, but also to correctly describe the requirements of a system. The purpose of the refinement chart is to provide an easily manageable representation for different refinements of a system. The refinement chart offers a clear view of assistance in "system" integration. This approach also gives a clear view about the system assembling based on the operating modes and different kinds of features. For example, if a developer does not want to provide any particular feature in any system, then using the refinement chart, it is possible to find that removable feature easily and not to include in the final development system. However, it can also provide the information that, which other parts will be affect-able, when we remove the particular operating modes.
- *Real-time animator*: Lots of formal methods based animator are available for different formal languages. But all kinds of animator use only *toy-data* sets. No any tool is available for real-time data testing without generating the source code of the system. We have provided an architecture to use a set of real-time data for animation using formal specification. Here, we have discovered that the medical

experts are unable to understand the complex formal specifications, so that we have proposed a new technique to apply a real-time data set to animate the formal specification and a domain expert can anticipate in the system development. Another objective is to develop this technique also for requirement traceability according to the domain experts. For example, through the animation of the formal model using real-time data set, domain experts can help to find the missing behaviour of a system.

- *Automatic source code generation form formal models*: Different kinds of code generation tools are available, and can generate source code into any programming languages but the main constraint is that all those tools are not applicable to generating the codes from Event-B modelling language. The main objective is to develop a set of code generation tools, which can support automatic code generation into several programming languages from Event-B modelling language and supports in the development life-cycle of a critical system from requirement analysis to code generation.
- *Integration of different approaches*: We proposed a new framework to compose different kinds of formal techniques to model a critical system to overcome the existing problems. An integration of formal techniques in the development process of a critical system provides the modelling concepts with formal semantics that captures at a high-level of abstraction. Modelling concepts should not be restricted due to apply the verification techniques for checking the correctness of a system. However, the system specifier should have the freedom to use the intuitive modelling concepts neglecting the complexity they impose on verification. The integration framework should bridge the gap between modelling concepts and input that is required for verification tools. For instance, integration of theorem prover and model checker can be used for verifying the essential properties. The compositional reasoning strategies using theorem prover and model checking can reduce the verification effort and to verify the required safety properties. The model checker helps to discover lots of errors and strengthening the safety properties through careful cross analysis of the model animation. System specifications are verified by both the model checker and theorem prover tools to prove the absence of any error.

References

1. Abdelmoez, W., Nassar, D. M., Shereshevsky, M., Gradetsky, N., Gunnalan, R., Ammar, H. H., et al. (2004). Error propagation in software architectures. In *Proceedings, 10th international symposium on software metrics* (pp. 384–393).
2. Abrial, J.-R. (1996). *The B-book: Assigning programs to meanings*. New York: Cambridge University Press.
3. Abrial, J.-R. (2010). *Modeling in Event-B: System and software engineering* (1st ed.). New York: Cambridge University Press.
4. Acuña, S. T., & Juristo, N. (2005). *International series in software engineering. Software process modeling*. Berlin: Springer.

5. Archinoff, G. H., Hohendorf, R. J., Wassyng, A., Quigley, B., & Borsch, M. R. (1990). *Verification of the shutdown system software at the Darlington nuclear generating station*. Presented at the international conference on control instrumentation and nuclear installations, Glasgow.
6. Artigou, J. Y., & Monsuez, J. J. (2007). *Cardiologie et maladies vasculaires*. Paris: Elsevier Masson.
7. Bayes, B. V. N., de Luna, A., & Malik, M. (2006). The morphology of the electrocardiogram. In *The ESC textbook of cardiovascular medicine* (pp. 1–36). Oxford: Blackwell.
8. Bell, R., & Reinert, D. (1993). Risk and system integrity concepts for safety-related control systems. *Microprocessors and Microsystems*, *17*, 3–15.
9. Berry, G., Bouali, A., Fornari, X., Ledinot, E., Nassor, E., & de Simone, R. (2000). ESTEREL: A formal method applied to avionic software development. *Science of Computer Programming*, *36*(1), 5–25.
10. Bjørner, D., & Jones, C. B. (Eds.) (1978). *The Vienna development method: The meta-language*. London: Springer.
11. Blanc, L., & Dissoubray, S. (2000). Esterel methodology for complex system design. *Microelectronic Engineering*, *54*(1–2), 163–170.
12. Blanchard, B. S., & Fabrycky, W. J. (2006). *Prentice Hall international series in industrial and systems engineering*. *Systems engineering and analysis*. Upper Saddle River: Pearson Prentice Hall.
13. Bowen, J., & Stavridou, V. (1993). Safety-critical systems, formal methods and standards. *Software Engineering Journal*, *8*(4), 189–209.
14. Bozzano, M., & Villafiorita, A. (2010). *Design and safety assessment of critical systems* (1st ed.). Boston: Auerbach.
15. Britt, J. J. (1994). Case study: Applying formal methods to the traffic alert and collision avoidance system (TCAS) II. In *Ninth annual conference on computer assurance*, COMPASS'94 (pp. 39–51).
16. Butler, M., & Yadav, D. (2007). An incremental development of the Mondex system in Event-B. *Formal Aspects of Computing*, *20*(1), 61–77.
17. Cai, K.-Y. (1996). *Introduction to fuzzy reliability*. Norwell: Kluwer Academic.
18. CC. Common criteria. http://www.commoncriteriaportal.org/.
19. CC (2009). Common criteria for information technology security evaluation, part 1: Introduction and general model. http://www.iec.ch/.
20. CC (2009). Common criteria for information technology security evaluation, part 2: Security functional requirements. http://www.iec.ch/.
21. CC (2009). Common criteria for information technology security evaluation, part 3: Security assurance components. http://www.iec.ch/.
22. CDRH (2006). Safety of marketed medical devices. Center for Devices and Radiological Health, US FDA.
23. Clarke, E. M., & Wing, J. M. (1996). Formal methods: State of the art and future directions. *ACM Computing Surveys*, *28*, 626–643.
24. Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., et al. (2007). Combination of abstractions in the Astrée static analyzer. In *Lecture notes in computer science*. *Proceedings of the 11th Asian computing science conference on advances in computer science: Secure software and related issues* (pp. 272–300). Berlin: Springer.
25. Craigen, D., Gerhart, S., & Ralston, T. (1993). An international survey of industrial applications of formal methods. In J. P. Bowen & J. E. Nicholls (Eds.), *Workshops in computing*. *Z user workshop*, London, 1992 (pp. 1–5). London: Springer. ISBN 978-3-540-19818-5.
26. Crow, J., Owre, S., Rushby, J., Shankar, N., & Srivas, A. (1995). *A tutorial introduction to PVS*. Computer Science Laboratory, SRI International.
27. Cullyer, W. J. (1989). Implementing high integrity systems: The viper microprocessor. *IEEE Aerospace and Electronic Systems Magazine*, *4*(6), 5–13.
28. Cullyer, W. J., Goodenough, S. J., & Wichmann, B. A. (1991). The choice of computer languages for use in safety critical systems. *Software Engineering Journal*, *6*, 51–58.

29. Dobrica, L., & Niemelä, E. (2002). A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering*, *28*(7), 638–653.
30. Draft defence standard 00-56 (1996). Safety management requirements for defence systems containing programmable electronics. Ministry of Defence, UK.
31. Eubanks, C. F., Kmenta, S., & Ishii, K. (1996). System behavior modeling as a basis for advanced failure modes and effects analysis. In *Proceedings of the ASME design engineering technical conference*. London: UCL Press.
32. Fankhauser, H. (2001). Safety functions versus control functions. In U. Voges (Ed.), *Lecture notes in computer science: Vol. 2187. Computer safety, reliability and security* (pp. 66–74). Berlin: Springer.
33. FDA. Food and Drug Administration. http://www.fda.gov/.
34. Fries, R. C. (2011). *Handbook of medical device design*. New York: Dekker.
35. Gall, H. (2008). Functional safety IEC 61508/IEC 61511 the impact to certification and the user. In *Proceedings of the 2008 IEEE/ACS international conference on computer systems and applications*, AICCSA'08 (pp. 1027–1031). Washington: IEEE Comput. Soc.
36. Gibbs, W. W. (1994). Software's chronic crisis. *Scientific American*, September.
37. Gordon, M. J. C. (1983). *LCF-LSM: A system for specifying and verifying hardware*. University of Cambridge Computer Laboratory.
38. Halbwachs, N., Caspi, P., Raymond, P., & Pilaud, D. (1991). The synchronous dataflow programming language Lustre. In *Proceedings of the IEEE* (pp. 1305–1320).
39. Hall, A. (1996). Using formal methods to develop an ATC information system. *IEEE Software*, *13*(2), 66–76.
40. Harel, D. (1987). *Algorithmics: The spirit of computing*. Boston: Addison-Wesley Longman.
41. Harrild, D. M., & Henriquez, C. S. (2000). A computer model of normal conduction in the human atria. *Circulation Research*, *87*, 25–36.
42. Hegde, V., & Raheja, D. (2010). Design for reliability in medical devices. In *Reliability and maintainability symposium (RAMS), 2010 proceedings—annual* (pp. 1–6).
43. Heimdahl, M. P. E., & Leveson, N. G. (1996). Completeness and consistency in hierarchical state-based requirements. *IEEE Transactions on Software Engineering*, *22*, 363–377.
44. Hennebert, C., & Guiho, G. (1993). SACEM: A fault tolerant system for train speed control. In *The twenty-third international symposium on fault-tolerant computing*, FTCS-23 (pp. 624–628). Digest of papers.
45. High Confidence Software and Systems Coordinating Group (2009). *High-confidence medical devices: Cyber-physical systems for 21st century health care* (Technical report). NITRD. http://www.nitrd.gov/About/MedDevice-FINAL1-web.pdf.
46. Hiller, M., Jhumka, A., & Suri, N. (2001). An approach for analysing the propagation of data errors in software. In *Proceedings of the 2001 international conference on dependable systems and networks (formerly: FTCS)*, DSN'01 (pp. 161–172). Washington: IEEE Comput. Soc.
47. Hinchey, M. G., & Bowen, J. P. (1995). *Prentice Hall international series in computer science. Applications of formal methods*. London: Prentice Hall.
48. Hokstad, P., & Corneliussen, K. (2004). Loss of safety assessment and the IEC 61508 standard. *Reliability Engineering & Systems Safety*, *83*(1), 111–120.
49. Houston, I., & King, S. (1991). CICS project report experiences and results from the use of Z in IBM. In S. Prehn & W. Toetenel (Eds.), *Lecture notes in computer science: Vol. 551. VDM'91 formal software development methods* (pp. 588–596). Berlin: Springer.
50. IEC60513 (1994). International Electrotechnical Commission: Fundamental aspects of safety standards for medical electrical equipment. http://www.iec.ch/.
51. IEC62304 (2006). International Electrotechnical Commission: Medical device software—software life-cycle processes. http://www.iec.ch/.
52. IEC60513 (2007). International Electrotechnical Commission: Medical electrical equipment. http://www.iec.ch/.
53. IEC61508 (2008). IEC functional safety and IEC 61508: Working draft on functional safety of electrical/electronic/programmable electronic safety-related systems. http://www.iec.ch/.

54. IEEE-SA. IEEE Standards Association. http://standards.ieee.org/.
55. IEEE Std. 1012-1998. IEEE standard for software verification and validation. http://standards.ieee.org/.
56. IEEE Std. 1074-1997. IEEE standard for developing software life cycle processes. http://standards.ieee.org/.
57. Imperial Chemical Industries Ltd., Chemical Industries Association, & Chemical Industry Safety and Health Council (1977). *A guide to hazard and operability studies*. London: Chemical Industry Safety and Health Council of the Chemical Industries Association.
58. ISO. International Organization for Standardization. http://www.iso.org/.
59. ISO 13485. International Organization for Standardization: Medical devices—quality management systems—requirements for regulatory purposes. http://www.iso.org/.
60. ISO 14971. International Organization for Standardization: Medical devices—application of risk management to medical devices. http://www.iso.org/.
61. Jetley, R. P., Carlos, C., & Purushothaman Iyer, S. (2004). A case study on applying formal methods to medical devices: Computer-aided resuscitation algorithm. *International Journal on Software Tools for Technology Transfer*, *5*(4), 320–330.
62. Jetley, R., Purushothaman Iyer, S., & Jones, P. (2006). A formal methods approach to medical device review. *Computer*, *39*(4), 61–67.
63. Johnson, J. A. (2012). FDA regulation of medical devices. http://www.fas.org/sgp/crs/misc/R42130.pdf.
64. Jones, C. B. (1990). *Systematic software development using VDM* (2nd ed.). Upper Saddle River: Prentice Hall.
65. Kaivola, R., Ghughal, R., Narasimhan, N., Telfer, A., Whittemore, J., Pandav, S., et al. (2009). Replacing testing with formal verification in Intel Core^{tm} i7 processor execution engine validation. In *Proceedings of the 21st international conference on computer aided verification*, CAV'09 (pp. 414–429). Berlin: Springer.
66. Kanholm, J. (2003). *ISO 13485:2003 & FDA QSR, 21 CFR 820, quality manual: 34 procedures and forms*. Los Angeles: AQA Press.
67. Kapur, K. C. (2007). *Reliability and maintainability* (pp. 1921–1955). New York: Wiley.
68. Keatley, K. L. (1999). A review of the FDA draft guidance document for software validation: Guidance for industry. *Quality Assurance*, *7*(1), 49–55.
69. Khan, M. G. (2008). *Rapid ECG interpretation*. Clifton: Humana Press.
70. Laprie, J. C. C., Avizienis, A., & Kopetz, H. (Eds.) (1992). *Dependability: Basic concepts and terminology*. Secaucus: Springer.
71. Lecomte, T., Servat, T., & Pouzancre, G. (2007). Formal methods in safety-critical railway systems. In *10th Brazilian symposium on formal methods*, Ouro Preto (pp. 29–31).
72. Leveson, N. G. (1991). Software safety in embedded computer systems. *Communications of the ACM*, *34*, 34–46.
73. Leveson, N. G., & Harvey, P. R. (1983). Software fault tree analysis. *The Journal of Systems and Software*, *3*(2), 173–181.
74. Leveson, N. G., & Turner, C. S. (1993). An investigation of the Therac-25 accidents. *Computer*, *26*, 18–41.
75. Leveson, N. G., Heimdahl, M. P. E., Hildreth, H., & Reese, J. D. (1994). Requirements specification for process-control systems. *IEEE Transactions on Software Engineering*, *20*(9), 684–707.
76. Lions, J. L. (Chairman) (1996). *Ariane 5 flight 501 failure: Report by the inquiry board* (Technical report). Paris: European Space Agency.
77. Lyu, M. R. (Ed.) (1996). *Handbook of software reliability engineering*. Hightstown: McGraw-Hill.
78. Macedo, H. D., Larsen, P. G., & Fitzgerald, J. (2008). Incremental development of a distributed real-time model of a cardiac pacing system using VDM. In *Lecture notes in computer science*. *Proceedings of the 15th international symposium on formal methods*, FM'08 (pp. 181–197). Berlin: Springer.

79. Main Commission (1994). *Report on the accident to Airbus A320-211 aircraft in Warsaw on 14 September 1993* (Technical report). Warsaw: Aircraft Accident Investigation.
80. Marciniak, J. J. (2002). *Encyclopedia of software engineering* (2nd ed.). New York: Wiley.
81. McDermid, J. A. (2002). Software hazard and safety analysis. In *Proceedings of the 7th international symposium on formal techniques in real-time and fault-tolerant systems*, FTRTFT'02 (pp. 23–36). London: Springer. Co-sponsored by IFIP WG 2.2.
82. McDermid, H. C., Forder, J., & Storrs, G. (1993). Sam—a tool to support the construction, review and evolution of safety arguments. In *Directions in safety-critical systems* (pp. 195–216). London: Springer.
83. MIL-STD-882C (1993). System safety program requirements. US DoD. http://www.system-safety.org/.
84. Miller, S. P., & Srivas, M. (1995). Formal verification of the AAMP5 microprocessor: A case study in the industrial use of formal methods. In *Proceedings, Workshop on industrial-strength formal specification techniques* (pp. 2–16).
85. Milner, R. (1982). *A calculus of communicating systems*. Secaucus: Springer.
86. NASA Technical Team (2004). *NASA software safety guidebook* (Technical report). NASA Technical Standard.
87. Neumann, P. (1995). Safeware: System safety and computers. *Software Engineering Notes*, *20*, 90–91.
88. Overture. Overture: Formal modelling in VDM. http://www.overturetool.org/.
89. Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, *15*, 1053–1058.
90. Paulk, M. C. (1995). *The SEI series in software engineering*. *The capability maturity model: Guidelines for improving the software process*. Reading: Addison-Wesley.
91. Price, D. (1995). Pentium FDIV flaw-lessons learned. *IEEE MICRO*, *15*(2), 86–88.
92. Redmill, M. C. F., & Catmur, J. (1999). *System safety: HAZOP and software HAZOP* (1st ed.). Chichester: Wiley.
93. Register, O. F. (1999). *Code of federal regulations*. *Guidance for industry and FDA: Regulation of medical devices: Background information for international officials*.
94. Register, O. F. (2011). *Code of federal regulations*. *Title 21, Food and drugs, Pt. 1-99* (p. 511). Revised as of April 1, 2011. US Independent Agencies and Commissions. ISBN 9780160883941.
95. Rouse, W. B., & Compton, W. D. (2009). Systems engineering and management. *Information, Knowledge, Systems Management*, *8*(1–4), 231–240.
96. RTCA (1992). Do-178B, software considerations in airborne systems and equipment certification. Committee: SC-167. http://www.rtca.org/.
97. Rushby, J. (1995). *Formal methods and their role in the certification of critical systems* (Technical report). Safety and reliability of software based systems (twelfth annual CSR workshop).
98. Schumann, J. M. (2001). *Automated theorem proving in software engineering*. New York: Springer.
99. Sommerville, I. (1995). *Software engineering* (5th ed.). Redwood City: Addison-Wesley Longman.
100. Stepney, S., Cooper, D., & Woodcock, J. (2000). *An electronic purse: Specification, refinement, and proof* (Technical monograph PRG-126). Oxford University Computing Laboratory Programming Research Group.
101. Tekinerdogan, B., Sozer, H., & Aksit, M. (2008). Software architecture reliability analysis using failure scenarios. *The Journal of Systems and Software*, *81*(4), 558–575.
102. Thomas, M. (1993). The industrial use of formal methods. *Microprocessors and Microsystems*, *17*, 31–36.
103. Trafford, P. J. (1997). *The use of formal methods for safety-critical system*. PhD thesis, Kingston University.

104. Tretmans, J., Wijbrans, K., & Chaudron, M. R. V. (2001). Software engineering with formal methods: The development of a storm surge barrier control system revisiting seven myths of formal methods. *Formal Methods in System Design*, *19*(2), 195–215.
105. Voas, J. (1997). Error propagation analysis for cots systems. *Computing and Control Engineering Journal*, *8*(6), 269–272.
106. von Neumann, J. (1966). *Theory of self-reproducing automata*. Chicago: University of Illinois Press. A. W. Burks (Ed.).
107. Wassyng, A., & Lawford, M. (2003). Lessons learned from a successful implementation of formal methods in an industrial project. In K. Araki, S. Gnesi, & D. Mandrioli (Eds.), *Lecture notes in computer science: Vol. 2805*. *FME 2003: Formal methods* (pp. 133–153). Berlin: Springer.
108. Wichmann, B. A., & British Computer Society (1992). *Software in safety-related systems* (Special report). BCS.
109. Wilkinson, P. J., & Kelly, T. P. (1998). Functional hazard analysis for highly integrated aerospace systems. In *Certification of ground/air systems seminar* (pp. 4–146). New York: IEEE. Ref. No. 1998/255.
110. Wizemann, T. (Ed.) (2010). *Public health effectiveness of the FDA 510(k) clearance process: Balancing patient safety and innovation: Workshop report*. Washington: National Academies Press.
111. Woodcock, J., & Banach, R. (2007). The verification grand challenge. *Journal of Universal Computer Science*, *13*(5), 661–668.
112. Woodcock, J., Stepney, S., Cooper, D., Clark, J. A., & Jacob, J. (2008). The certification of the Mondex electronic purse to ITSEC level E6. *Formal Aspects of Computing*, *20*(1), 5–19.
113. Woodcock, J., Larsen, P. G., Bicarregui, J., & Fitzgerald, J. (2009). Formal methods: Practice and experience. *ACM Computing Surveys*, *41*, 19:1–19:36.
114. Xu, H., & Maibaum, T. (2012). An Event-B approach to timing issues applied to the generic insulin infusion pump. In Z. Liu & A. Wassyng (Eds.), *Lecture notes in computer science: Vol. 7151*. *Foundations of health informatics engineering and systems* (pp. 160–176). Berlin: Springer.
115. Zhang, Y., Jones, P. L., & Jetley, R. (2010). A hazard analysis for a generic insulin infusion pump. *Journal of Diabetes Science and Technology*, *4*(2), 263–283.
116. Zio, E. (2009). Reliability engineering: Old problems and new challenges. *Reliability Engineering & Systems Safety*, *94*(2), 125–141.

http://www.springer.com/978-1-4471-5259-0

Using Event-B for Critical Device Software Systems

Singh, N.K.

2013, XVIII, 326 p., Hardcover
ISBN: 978-1-4471-5259-0