Design and Safety Assessment of Critical Systems

CRC Press
Taylor & Francis Group
AN AUERBACH BOOK

Marco Bozzano
Adolfo Villafiorita

Design and Safety Assessment of Critical Systems

Design and Safety Assessment of Critical Systems

Marco Bozzano
Adolfo Villafiorita

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
AN AUERBACH BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Version Date: 20140529

International Standard Book Number-13: 978-1-4398-0332-5 (eBook - PDF)

To Antonio

To Barbara

Contents

Preface

Safety-critical systems—namely, systems whose failure may cause death or injury to people, harm to the environment, or economical loss—are becoming more complex, both in the type of functionality they provide and in the way they are demanded to interact with the environment. Traditionally, safety analysis techniques and procedures are used to identify risks and hazards, with the goal of eliminating, avoiding, or reducing the probability of failure. However, these techniques are often performed manually and hence are a time-consuming activity, itself vulnerable to human error, because they rely on the ability of the safety engineer to understand and to foresee system behavior. The growing complexity of safety-critical systems requires an adequate increase in the capability of safety engineers to assess system safety, encouraging the adoption of formal techniques.

This book is an introduction to the area of design and verification of safety-critical systems, with a focus on safety assessment using formal methods. After an introduction covering the fundamental concepts in the areas of safety and reliability, the book illustrates the issues related to the design, development, and safety assessment of critical systems. The core of the book covers some of the most well-known notations, techniques, and procedures, and explains in detail how formal methods can be used to realize such procedures. Traditional verification and validation techniques and new trends in formal methods for safety assessment are described. The book ends with a discussion on the role of formal methods in the certification process. The book provides an in-depth and hands-on view of the application of formal techniques that are applicable to a variety of industrial sectors, such as transportation, avionics and aerospace, and nuclear power.

Who should read this book. The book is addressed to both researchers and practitioners in the areas of safety engineering and safety assessment who are interested in the application of formal verification in such areas. It can also be of interest to computer scientists and individuals skilled in formal verification who wish to see how these methodologies can be applied for safety assessment of critical systems. The book can also be used as a reference book for (bachelor and master) students in engineering and computer science.

Prerequisites. The book is mostly self-contained and should be generally accessible to people who have a basic background in mathematics or computer science at the level corresponding to the early years of university courses in engineering or computer science. A prior exposure to topics such as propositional logic, automata theory, model checking, and probability theory could be desirable, although not indispensable.

Structure of the book. The book is structured as follows:

- **Chapter 1, Introduction,** introduces and motivates the main topics of the book.
- **Chapter 2, Dependability, Reliability, and Safety Analysis,** looks in detail at some of the most typical safety criteria that pertain to the design and assessment of safety-critical systems. We start by introducing some common terminology and continue by presenting some fault models and the approaches to dealing with faults, namely fault, detection, fault prediction, fault tolerance, and fault coverage.
- **Chapter 3, Techniques for Safety Assessment,** introduces the traditional notation and techniques for safety assessment. Starting with the definition of hazard and accident, we continue by presenting fault trees, FMECA, HAZOP, Event Tree Analysis, Risk Analysis, and Risk Measures.
- **Chapter 4, Development of Safety-Critical Applications,** looks at the development process of safety-critical systems, by highlighting those management and organizational aspects that most influence the development of safety-critical systems. In this chapter we present a generic development approach that is inspired by various development standards in both the civil and military sectors.
- **Chapter 5, Formal Methods for Safety Assessment,** is an in-depth presentation of formal methods and the role they play in the verification and validation of safety-critical systems. We divert from the "traditional" approach related to the usage of formal methods and propose how formal methods can be effectively used to automate safety analysis techniques.
- **Chapter 6, Formal Methods for Certification,** describes some widely adopted standards for the certification of safety-critical systems. We start with the certification process of aircraft systems and continue by describing how formal methods can be applied to support certification activities.
- Finally, the appendices describe the NuSMV model checker and the FSAP platform, and provide some more references and starting points for further development.

Additional information, including the code for the examples presented in this book, can be retrieved from the web site `http://safety-critical.org`.

Acknowledgments

The authors would like to thank and acknowledge all the people who provided feedback and support for (earlier versions of) this book, namely, Andreas Lüdtke, Andrea Mattioli, Matteo Negri, Chris Papadopoulos, and Komminist Weldemariam. Special thanks go to Viktor Schuppan for giving specific advice on Chapter 5 and to our editor, John Wyzalek, and all the staff at Taylor & Francis for their help and support.

Finally, Marco Bozzano would like to dedicate the book to his brother Antonio, who prematurely died in November 2004.

Adolfo Villafiorita wishes to thank his family for all the help and support they provided. A big thank-you to his wife Barbara, his dad Enzo, and Andrea, Ombretta, and Rienzo. A special dedication goes to his mom, Geni, who could not see this book completed.

Marco Bozzano

Adolfo Villafiorita

About the Authors

Marco Bozzano is a senior researcher in the Embedded Systems Unit of Fondazione Bruno Kessler, Italy. He has strong expertise in the application of formal methods, and he has published a number of papers in the area of formal verification of safety-critical systems.

Adolfo Villafiorita is a senior researcher at Fondazione Bruno Kessler. He has many years of experience in the application of formal methods in technology transfer projects and in the development of security and safety-critical applications. He is a contract professor at the University of Trento.

Chapter 1

Introduction

1.1 Complex Safety-Critical Systems

Every journey has a start. Ours is the definition of complex safety-critical systems, given in SAE (1996), a set of guidelines for the development of avionic systems:

> "**A complex safety-critical system** is a system whose safety cannot be shown solely by test, whose logic is difficult to comprehend without the aid of analytical tools, and that might directly or indirectly contribute to put human lives at risk, damage the environment, or cause big economical losses."

The definition is peculiar, as it puts together two concepts—namely, *complexity* and *criticality*—that can be defined independently. The motivation for presenting them together in SAE (1996) is obvious: airplanes are both complex and critical. We use this definition for the following reasons:

1. There is a steady trend toward the use of digital systems of increasing complexity in safety-critical applications. Systems need not be digital or complex to be safety critical: The Wright brothers invented and flew the airplane 12 years before Alan Turing was born. However, the flexibility and performance of digital technologies have greatly contributed to increasing their adoption in the safety-critical sector.
2. Systems that are both complex and critical represent an engineering challenge with which traditional techniques have difficulties dealing. Citing Lyu (1996): "The demand for complex hardware/software systems has increased more rapidly than the ability to design, implement, test, and maintain them."

A more detailed discussion and some data will help clarify and put them in perspective.

1.1.1 A Steady Trend toward Complexity

One of the most effective descriptions of the impact that digital technologies have had and are still having on system engineering is given in Brooks (1995), a seminal book about software project management:

> No other technology since civilization began has seen six orders of magnitude price-performance gain in 30 years. In no other technology can one choose to take the gain in **either** improved performance **or** in reduced costs.

Such a trend, first noted by Moore (1965) and since then reported numerous times[1], not only has promoted a capillary diffusion of digital control systems, but has also been a key enabler for the delivery of systems with more functions and increased complexity. Let us provide some more details about both impacts.

The reduction in costs is increasing diffusion. According to Ebert and Jones (2009), in 2008 there were some 30 embedded microprocessors per person in developed countries, with at least 2.5 million function points of embedded software. (A function point is a measure of the size of a software system. Tables to convert function points to lines of source code are available. For instance, Quantitative Software Management, Inc. (2009) estimates one function point as 148 lines of C code.) Millions of these embedded microprocessors are used for safety-critical applications and many of them have faults. For instance, between 1990 and 2000, firmware errors accounted for about 40% of the half-million recalled pacemakers (Maisel et al., 2001; Ebert and Jones, 2009).

The gain in performance is increasing complexity. A recent report by the Jet Propulsion Laboratory (Dvorak, 2009) analyzes the size of NASA flight software for both human and robotic missions. Data for robotic missions are shown in Figure 1.1, where the x-axis shows the year and the name of the mission and the y-axis shows the flight software size, using a logarithmic scale.

As can be seen from the diagram, software size is growing exponentially. The same trend is shown by NASA manned missions (Apollo, Shuttle, and International Space Station), although the number of data points is too small to demonstrate any trend, as pointed out by the authors of the report. Similar growth can also be observed in other domains, such as civil and military avionics, automotive, and switching systems, to name a few (Ferguson, 2001; Dvorak, 2009; Ebert and Jones, 2009).

[1] As an example, we quote from Air Force Inspection and Safety Center (1985), written 25 years ago: "The development and usage of microprocessors and computers has grown at a phenomenal rate in recent years, from 1955, when only 10 per cent of our weapon systems required computer software, to today, when the figure is over 80 per cent."

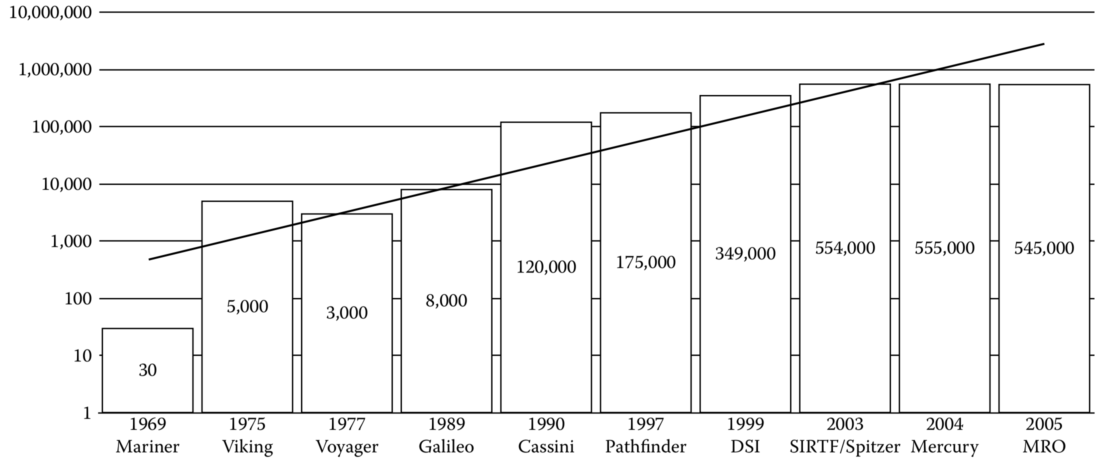

Figure 1.1 Growth in flight software, NASA missions. (Source: From Dvorak, D.L., Editor (2009). NASA Study on Flight Software Complexity. Available at http://oceexternal.nasa.gov/OCE_LIE/pdf/1021608main_FSWC_Final_Report.pdf.)

Together with size, complexity is also increasing. This is due to various factors, among which we mention:

- **Number of functions.** In the words of Dvorak, (2009), software is working as a sponge "because it readily accommodates evolving understanding, making it an enabler of progress." A very good example of this phenomenon are jet-fighters. Modern jet-fighters, in fact, are designed to be slightly aerodynamically unstable. That is, a small variation in the current flight conditions causes the plane to abruptly change trajectory. This feature allows *fast transients*, that is, quick changes in speed, altitude, and direction—maneuvers on which the life of a pilot in combat might depend. However, to maintain the flight level, the airplane must be kept under active and constant control. This is performed by a digital system (called fly-by-wire, FBW), because the precision and the frequency of corrective actions that must be taken make the task impossible for a pilot. The FBW, thus, continuously reads plane's data and pilot's commands and constantly positions the actuators according to the data received, in practice decoupling the pilot from the controls of the plane. See, for example, Langer et al. (1992), NASA (2009), and Various Authors (2009) for more details.
- **Number of states.** Software and, more generally, digital systems, have a large number of states that can make their comprehension difficult and exhaustive testing impossible.
- **Discrete behavior.** Software models discrete systems with discontinuous behaviors. That is, a little variation in one program input could cause a great variation in one output. Such discontinuities are, in principle, uniformly distributed over the whole input space. The consequence is that performing a test on an input value tells little about the behavior of the system on the neighboring region. This is rather different from analog systems, in which small variations in one input usually cause small variations in outputs.
- **Invisibility.** As pointed out in Brooks (1995), software is invisible, unvisualizable, and its reality is not embedded in space; software can be visualized only by overlapping several different views (e.g., data-flow, control-flow) that define its behavior. Brook's observation holds up today. The UML, a standard notation in object-oriented programming, defines nine different diagrams to model a software system. At least five of them must be used to properly model a system's architecture (Kruchten, 1995).

1.1.2 An Engineering Challenge

Safety-critical systems have stringent requirements. Not all systems are equally critical when they fail. Looking at the consequences associated with system or function failure is thus a good way to discriminate among the different levels of criticality.

Such a classification can be found, for instance, in Sommerville (2007), where the author distinguishes among:

- Business-critical systems
- Mission-critical systems
- Safety-critical systems

In the first case (business-critical systems), a failure of the system might cause a high economic loss. This is often due to the interruption of service caused by the system being unusable. Examples of business-critical systems are a stock-trading system, the ERP system[2] of a company, or an Internet search engine, such as Google[3].

In the second case (mission-critical systems), failures might cause the loss of a function necessary to achieve one of the goals for which the system was designed. Examples of mission-critical systems are an unmanned spacecraft and the software controlling a baggage-handling system of an airport.

In the third case (safety-critical systems), system failures might cause risk to human life or damages to the environment. Examples of safety-critical systems are aircraft, the controller of an unmanned train metro system, and the controller of a nuclear plant.

Critical systems are further distinguished between fail-operational and fail-safe systems, according to the tolerance they must exhibit to failures. In particular:

- **Fail-operational** systems are typically required to operate not only in nominal conditions—that is when all the (sub)components of the system work as expected—but also in degraded situations, that is, when some parts of the system are not working properly. Airplanes are fail-operational because they must be able to fly even if some components fail.
- **Fail-safe** systems are demanded to safely shut down in case of single or multiple failures. Trains are fail-safe systems because stopping a train is typically sufficient to put it into a safe state.

(Safety-critical) systems fail for the most diverse reasons. Following O'Connor (2003), we mention the following causes for systems to fail:

- **The design might be inherently incapable.** This is typically due to errors during development that result in building a system that it is inadequate for the purpose for which it was devised. Causes are the most diverse. Some essential requirement might have been missed during the specification, such as some environmental condition the system should have been resilient to. Some error

[2] Enterprise Resource Planning system.

[3] Notice that such an interruption would be both a loss for the search engine provider (e.g., missed revenues in advertisement) and its users (e.g., decreased productivity in finding information on the Internet).

might have been introduced during design, as it is so common in software, for instance. We also mention sneaks, integration, and interaction errors that occur when the system does not work properly even though all its components do.

- **The system might be overstressed.** This is often due to the system being operated in environmental conditions for which it was not designed. Notice that in complex systems, stress on a component might be caused by other components failing in unexpected ways. For instance, a short circuit in an electronic component might cause an excessive voltage in another one.
- **Variation in the production and design.** This is due to natural variations in the materials, in the production processes, and in quality assurance procedures. Problems arise when a component that is, let us say, less resistant than average is subject to a load that is above the average.
- **Wear-out and other time-related phenomena.** All components become weaker with use and age. As a consequence, their probability of failure increases over time. The problem can be mitigated by proper system maintenance and replacement of components before they wear out. However, environmental conditions (e.g., mechanical vibrations in rockets, pressure in submarines), design issues (e.g., friction on the insulation of an electric cable installed in the wrong position) can accelerate wear-out in unpredicted ways. Moreover, for certain systems, such as spacecraft, replacement may be impossible.
- **Errors.** Errors can occur in any phase of a system's life cycle. We mentioned above errors occurring during system specification and development. Errors can occur also during maintenance (e.g., a component replaced in the wrong way), during operations (e.g., due to problems in training, documentation, or just distraction), or during system disposal.

An Engineering Challenge. The development of critical systems thus adds a further level of complexity to standard engineering activities because it requires to consider, and properly deal with, all the diverse causes of failure, so that the system can maintain a function even if some components fail or operators make errors.

This requires the adoption of development processes in which safety is considered from the early stages. In aeronautics, for instance, safety requirements (i.e., requirements stating the (degraded) conditions under which systems must remain operational) are defined along with the other system requirements. During system engineering, development activities are conducted in parallel with a set of safety analysis activities that have the specific goal of identifying all possible hazards, together with their relevant causes, in order to assess if the system behaves as required under all the operational conditions. These activities are crucial (e.g., for system certification) to ensure that the development process is able to guarantee the specific safety level assigned to the system.

1.2 Dealing with Failures: A Short History of Safety Engineering

System safety has been a concern in engineering for a long time and it has always been an important factor in determining systems' adoption. Elevators were in use since the third century, but they became popular only after 1853 when Elisha Otis demonstrated a freight elevator equipped with a safety device to prevent falling, in case a supporting cable should break (History of the Elevator, 2009).

Safety engineering, however, matured as a discipline only in the past 50 years. In the following few paragraphs we provide an overview of the main steps that led to this revolution. See Air Force Safety Agency (2000), Ericson (1999), Leveson (2003), and Ericson (2006) for more details.

System safety, as we know it today, is strictly related to the problems the U.S. Air Force experienced with accidents after World War II and its efforts to prevent them. According to Hammer (2003), from 1952 to 1966 the U.S. Air Force lost 7,715 aircraft. In the accidents, 8,547 persons were killed. Although many of those accidents were blamed on pilots, there were many who did not believe the cause was so simple (Leveson, 2003).

During the 1950s, a series of accidents involving the Atlas ICBM contributed to a growing dissatisfaction with the "fly-fix-fly" approach. At the time, safety was not a specific system engineering activity, but rather a concern distributed among the project team. After system deployment, if an accident occurred, investigations reconstructed the causes to allow engineers to "fix" the design and prevent future similar events.

The approach, however, soon became ineffective because it did not help prevent accidents with causes different from those investigated, and deemed too costly and too dangerous, considering, for example, the risks of an accident involving a nuclear weapon. These considerations eventually led to abandoning the existing development practices and adopting, instead, an approach in which system safety activities are integrated into the development process (Leveson, 2003). Such an integrated approach had its roots in a seminal paper by Amos L. Wood, "The Organization of an Aircraft Manufacturer, Air Safety Program," presented in 1946, and in a paper by William I. Stieglitz, "Engineering for Safety," published in 1948 (Air Force Safety Agency, 2000). From Stieglitz, H.A. Watson of Bell Laboratories first conceived Fault Tree Analysis, in connection with the development of the Launch Control System of the Minuteman missile. The technique proved so successful that it was later extensively applied to the entire Minuteman program.

In 1965, Boeing and the University of Washington sponsored the first System Safety Conference and later developed a software system for the evaluation of multiphase fault trees. The technique soon caught on in other areas, most notably the civil nuclear sector, which has been, since then, a great contributor to the technique and to safety in general. After the Apollo 1 launchpad fire in 1967, NASA hired

Boeing to implement an entirely new and comprehensive safety program for the Apollo project. As part of this safety effort, Fault Tree Analysis was performed on the entire Apollo system (Ericson, 1999). The technique was finally consolidated with the release by NUREG of the *Fault Tree Handbook* (Vesely et al., 1981).

Software safety analysis also had its roots in the 1960s. The first references are dated 1969 and, since then, the subject has gained momentum and interest. We cite the Air Force Inspection and Safety Center (1985):

> "Software safety, which is a subset of system safety, is a relatively new field and is going to require a conscientious effort by all those involved in any system acquisition process or development effort to insure it is adequately addressed during system development."

Finally, in recent years, military standards such as MIL-STD-1574A (eventually replaced by the MIL-STD-882 series) and the growing demand for safety in civil applications—especially in the nuclear and transportation sector—have greatly contributed to the standardization of techniques, on the one hand, and to the standardization of development processes of safety-critical applications, on the other.

1.3 The Role of Formal Methods

As highlighted by Bowen and Stavidrou, lives have depended on mathematical calculations for centuries. In the nineteenth century, errors in logarithmic tables caused ships to miscalculate their position and possibly wreck as a result of such errors (Bowen and Stavridou, 1992). Mathematical representations of hardware and software systems (formal methods) have emerged in the past 30 years as a promising approach to allow a more thorough verification of the system's correctness with respect to the requirements, using automated and hopefully exhaustive verification procedures.

As described earlier, safety-critical systems are becoming more complex, both in the type of functionality they provide and in the way they are required to interact with their environment. Such growing complexity requires an adequate increase in the capability of safety engineers to assess system safety, a capability that is only partially matched by the progress made in the use of traditional methodologies, such as Fault Tree Analysis and failure mode and effect analysis, often carried out manually on informal representations of systems. The use of formal techniques for the safety assessment of critical systems, however, is still at a relatively early stage. This is due to the following reasons:

- **The role of formal methods for system design.** Nearly all standards used as references for the development and certification of safety-critical systems make little mention, if at all, of formal methods. Main causes include the maturity of techniques and tools when the standards were issued, skills needed for properly mastering the techniques, and difficulties related to an effective integration of formal methods in the development process.

- **The role of formal methods for system safety assessment.** Formal methods have traditionally been used to support system verification activities. There is, however, a fundamental difference between system verification and safety activities. The first is meant to demonstrate that the *nominal* system works as expected. A *single* counterexample is sufficient to show that a requirement is violated. The second is meant to support design by demonstrating that the *degraded* system works as expected. To do so, it is necessary to highlight *all possible combinations of failures* that lead to losing a function. This requires significant changes, both in the way in which systems are used and in the way verification engines are implemented.
- **Integration between design and safety assessment.** The information linking the design and the safety assessment phases is often carried out informally, and the connection between design and safety analysis may be seen as an *over-the-wall process.* Quoting Fenelon et al. (1994), "A design is produced with some cognisance of safety issues, it is 'tossed over the wall' to safety assessors who analyse the design and later 'toss it back' together with comments on the safety of the design and requests for change. Whilst something of a caricature, the shove is not entirely unrepresentative of current industrial processes." Thus, even when formal methods are used to support design activities, the extra effort spent there cannot be reused for safety assessment, because the formal designs are "lost" by this lack of integration between activities.

Recent developments are making the use of formal methods for system verification more appealing. We mention improvements on the representational power of formal notations, increased efficiency of the verification engines, better tool support, and significant improvements in the ease of use. Steady progress has also been measured with respect to the usability of formal methods for safety assessment. Novel algorithms, based on model checkers, have been defined, for instance, to support the automatic computation of fault trees and to automate common cause analysis; see, for example, Bozzano et al. (2003), Joshi et al. (2005), Åkerlund et al. (2006), Bozzano and Villafiorita (2007), and Bozzano et al. (2007).

Despite the progress mentioned above, we are still far from a full, top-down, and completely automated verification of (complex) safety-critical systems. Formal methodologies, however, represent one of the most promising areas to improve the quality and safety of complex systems.

1.4 A Case Study: Three Mile Island

The complexity of the environment, functions performed, and difficult-to-understand interactions among system parts when components fail are main causes of engineering failures and accidents. In this section we briefly describe the Three

Mile Island accident, one of the worst in the civil nuclear sector. We do so by also presenting a formal model of the plant, written in the input language of NuSMV, a symbolic model checker. The formal model will be used in this book to reproduce the accident and demonstrate some of the techniques used for safety assessment. (See Appendix A for a description of the NuSMV model checker.)

The formal model has been built using the know-how obtained from an analysis of the accident. We cannot therefore pretend, nor do we claim, that the model and formal methods could have been used to build a better plant and prevent the accident.

Nevertheless, the example is a very good exercise for the following reasons:

1. It shows how we can use a formal language, whose expressiveness is limited to finite state machines over discrete data types, to model a physical system in which the behavior is determined by the laws of thermodynamics and nuclear physics. The challenge is to provide a suitable level of abstraction that allows us to accurately model the qualitative behaviors necessary for the analyses we want to perform. This is very common when using formal methods and model checking, as abstraction is often the only way to keep analyses manageable.
2. It shows how we can model a complex physical system using functional blocks. The trick here is to use the blocks to encode the "flow of information" (e.g., a pump is pumping) rather than the "flow of physical elements" (e.g., the coolant flows from the core to the steam generator). This results in a significant difference between the physical structure of the plant (at the level of abstraction considered in this book) and the actual functional block model. The approach is quite common when using model checkers for physical systems.
3. It presents an example in which safety analyses (and, in particular, fault trees) are conducted by observing the dynamics of the system rather than by statically analyzing the system architecture. This is a noteworthy improvement over standard assessment techniques and shows one of the advantages that can be obtained with the adoption of formal methods for safety assessment.

Most of the work presented here can be readily applied to other modeling languages and verification systems. For instance, to see the same example specified using the formalism of safecharts, have a look at Chen (2006).

1.4.1 Pressurized Water Reactors (PWRs)

Pressurized water reactors (PWRs) are second-generation nuclear power plants conceived in the 1950s and used for a variety of applications, ranging from the propulsion of nuclear submarines to the production of electricity for civil use. PWRs are the most common type of nuclear power plant, with hundreds of systems used for