Computational Sciences and Engineering 3(1) (2023) 153-161

University of Guilan

Computational Sciences and Engineering

journal homepage: https://cse.guilan.ac.ir/

Software Validation Using Markov Chain Method and State Transition Diagram

Shahrzad Oveisi[a,*], Mohammad Nadjafi[b], Mohammad Ali Farsi[b], Ali Moeini[a]

[a] Department of Algorithms and Computation, College of Engineering Sciences, University of Tehran, Tehran, Iran

[b] Aerospace Research Institute (Ministry of Science, Research and Technology); Department of Aerospace Management, Law and Standards;Tehran, P.O.B 14665-834, Iran

ARTICLE INFO

Article history:
Received 15 November 2023
Received in revised form 20 December 2023
Accepted 23 December 2023
Available online 23 December 2023

Keywords:
Software reliability
Finite state machine
Markov chain simulation
Transition diagram

ABSTRACT

Software plays a critical role in controlling the behavior of mechanical and electrical systems, as well as facilitating interactions among their components in cyber-physical systems (CPS). The risks associated with CPS systems can lead to the loss of functionality, performance, and even jeopardize human lives. Therefore, ensuring the reliability of software through comprehensive testing is of paramount importance in detecting and preventing potential errors. To achieve this objective, various methods and tools are employed to assess both the static and dynamic behavior of object-oriented software systems. One commonly used tool in the Unified Modeling Language (UML) is the state chart diagram, which visually represents the dynamic behavior of an object-oriented system. These diagrams depict the transitions and actions that occur as an object changes its state, driven by various inputs. To validate the accuracy of UML state chart diagrams, this paper proposes a method utilizing Finite State Machines (FSM) and Transition Tables. By creating a Transition Table, the UML state chart diagram can be effectively validated. To evaluate the proposed method, a set of Test Cases has been generated and applied to a real case study, ensuring the accuracy and reliability of the UML state chart diagram.

1. Introduction

One of the most important qualitative attributes of software systems is the reliability [1,2]. The Institute of Electrical and Electronics Engineers (IEEE) defines software reliability as the probability of a software system or component to perform its intended function under specified operating conditions over a specific period of time. It is an important factor in software quality as it measures the failures that can lead to system crashes or risks. Software failure is described as an external deviation of program operation from requirements, while a fault is a defect in the program during execution that can lead to failure under certain conditions. A fault can be a defective,

* Corresponding author.
E-mail addresses: shahrzad.oveisi@gmail.com (Sh. Oveisi)

https://doi.org/10.22124/cse.2023.26035.1067

nonexistent, or additional instruction, or a set of related instructions resulting from actual or potential failures. Software reliability is influenced by the coding, design, and requirements processes, and improving these steps through inspection or review can enhance software reliability. Evaluation of software reliability can only be done after testing or the completion of the product [1].

Different existing software reliability models can be used to estimate or predict software reliability. These models utilize fault data collected during testing, and reliability can be estimated or predicted by fitting the data to the model. Predicting reliability in the early stages of development can help project management reduce costs by minimizing repetition. Inadequate system reliability leads to undesirable and even unreliable services. Therefore, it is essential to provide suitable models for the characterization, estimation and prediction of software reliability [3]. Software validation is an important activity in order test whether the correct software has been developed. One main purpose validation to evaluate the quality of software. An important quality attribute is reliability. Today, behavioral modeling is a fundamental problem in software systems. Research has shown that most software systems either have problems or have been completely failed, and its main reasons are insufficient understanding of the processes and system specifications. Therefore, proper and accurate modeling of processes is a necessary issue in designing software systems. However, the problem that makes modeling (in particular, behavioral modeling) into an important issue in software engineering has the direct effect of modeling techniques for system evaluation. The more motivated the modeling technique will be, the easier it will be to evaluate [4].

A cyber-physical system (CPS) is a typical product of Industry 4.0, which plays a crucial role in integrating the physical and virtual worlds through real-time data processing services. CPSs enable the monitoring of physical systems using virtual systems, allowing for the analysis of data collected from the physical world in the virtual world. This analysis facilitates informed decision-making that can impact the physical world. Consequently, CPSs enable information integration, sharing, collaboration, real-time monitoring, and global optimization of systems. Various industries, including smart grids, healthcare, aircraft, digital manufacturing, and robotics, have leveraged CPSs for a wide range of applications. CPSs encompass networked control systems (NCSs), wireless sensor networks, smart grids, and more [5,6].

As CPS systems become increasingly complex, detecting faults and flaws has become more challenging. Software forms a crucial component of CPS, and the complexity of these software systems, often consisting of millions of lines of code, can pose risks to their functionality. Therefore, ensuring the stability and reliability of software in CPS requires thorough analysis and fault verification [7].

During software development, requirements analysis plays a vital role in determining the overall safety of the software. By identifying problems and errors that may lead to software failure and mitigating their risks during the requirements phase, the level of risk and system failure can be significantly reduced throughout the subsequent development stages. Unified Modeling Language (UML) is a widely-used modeling language that plays a crucial role in various phases of software development, including the requirements phase. It provides a standardized approach for specifying, documenting, and visualizing the artifacts of software-intensive systems being developed [4].

One of the versatile tools provided by UML is the state chart diagram, which describes the dynamic behavior of software systems. State chart diagrams illustrate the paths through which an object transitions between different states throughout its lifecycle. These paths are graphically represented

using the concept of Finite State Machines (FSM). FSM serves as a computational model for both the dynamic and static behavior of software systems. It operates by producing a finite number of states, one at a time, based on input symbols. The FSM starts from an initial state and ends at a final state, accepting input strings that lead it to the final state and rejecting strings that do not. FSMs are similar to Markov chains and can be converted into finite-state, discrete-parameter Markov chains. They are often used as a graphical tool for designing Markovian usage models and presenting them visually [8-10].

In this paper, the second section presents a proposed method along with a real case study (software of the Data and Command Unit). In subsection 2.1, the DCU Software Test Model is presented, followed by the transformation of this test model into a State Transition Diagram in subsection 2.2. Subsequently, in subsections 2.3 and 3, we will derive the transformation into FSM (Finite State Machine) and FSM Markovian Chains Model, respectively. Finally, in section 4, using the Markov Chain model, we calculate the system reliability.

2. Software Test Model

This section explains the proposed model for software testing, which has been presented for the command system. We first introduce this system and its general architecture. The primary objective of the Data and Command Unit is to ensure the prompt issuance of commands for the separation of the nose, engine, and parachutes based on simulated time and altitude. In order to initiate its functionality, this section is responsible for detecting the start of movement and receiving the Start signal. The Start signal is generated through the simultaneous cutting of the cord and the activation of the mass and spring switch. This signal serves as a command to initiate the operations of the two system processors, which utilize data from pressure sensors and the timeline of their internal timers to carry out their respective tasks. ***Figure 1*** illustrates a visual representation of the general architecture of the Data and Command Unit.

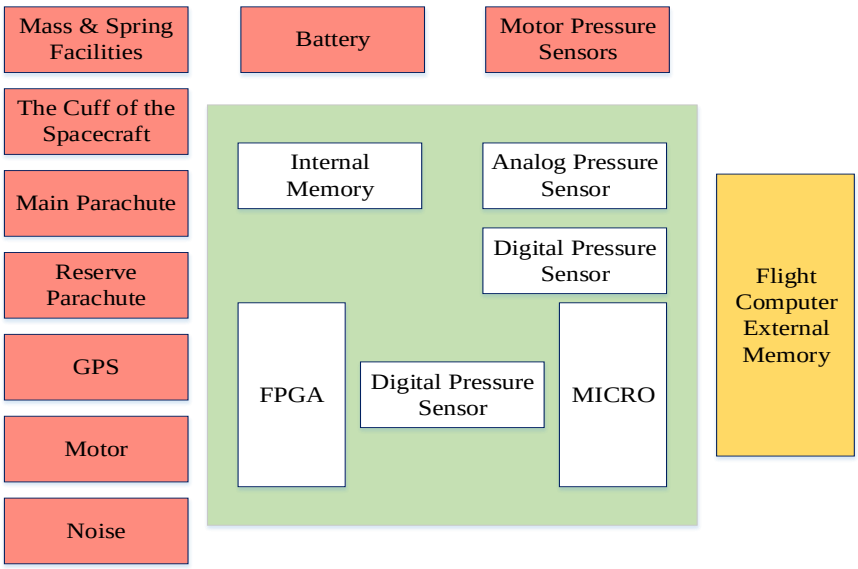

Figure 1. General architecture of Data and Command Unit

2.1. DUC Software Test Model

After detecting the engine shutdown, the software waits for 65 seconds, and then the nose separation flag is sent. Also, 80 seconds after the engine shutdown is detected, the engine separation flag is sent, and 210 seconds later, it begins to check the analog and digital pressure data. An average is taken each time the pressure data is checked, which is used for comparison in the algorithm.

If the mean value of voltage taken for analog pressure is >0.467 V, the 17-km altitude detection flag (analog) is sent. If the mean pressure taken for digital pressure is >8448 m, the 17-km detection flag (digital) is sent. When an altitude of 17-km is detected by both sensors, it will be stored. If the mean voltage taken for analog pressure is >1.5 V, the 10-km altitude detection flag is sent (analog). If the mean pressure taken for digital pressure is >15,104 mb, the 10-km altitude detection flag will be sent. If the analog pressure difference at 17-km altitude is higher than the analog pressure at 10-km altitude by 0.6, then the sensor is reliable, and the 7-km altitude can be detected by this sensor to issue the parachute brake command. 36 seconds after detecting the 17-km altitude, the parachute brake command is again issued. 375 seconds after take-off, the parachute brake command is again issued. 10 seconds after sending each of parachute brake commands, the command to turn on GPS is sent. Now in accordance with the above descriptions, we present the following state transition diagram of the DCU in Next section.

2.2. State Transition Diagram of DCU Software Test Model

The state transition diagram for the DCU with 19 states and 2 final states is shown in ***Figure 2*** Important DCU settings and operations are displayed with eighteen states. Based on the system setting, the final validation status of the Software will be evaluated as safe or unsafe modes. Transitions represent choices which determine the final validation status of the software.

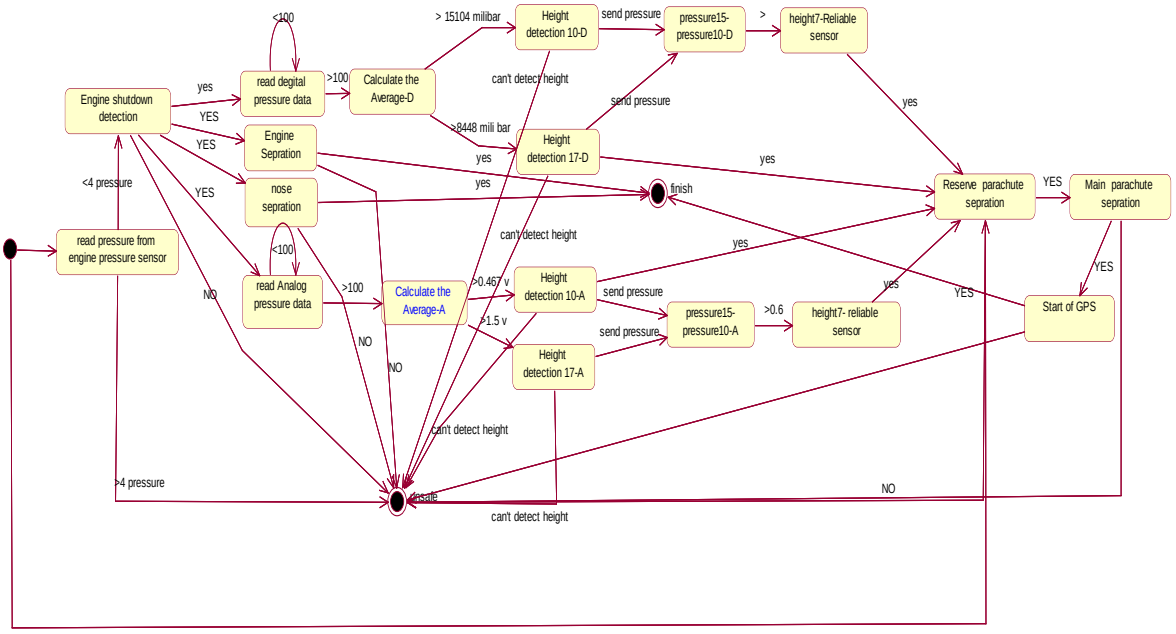

Figure 2. State Transition Diagram for the Data and Command Unit

2.3. State Transition Diagram to FSM Model

DCU state transition diagram shown in ***Figure 2*** can be converted to Finite State Machine (FSM) mode [14,15]. The diagram in ***Figure 3*** depicts Finite State Machines, which are models comprised of a set of states and transitions between them. These transitions can be triggered either by external inputs or internal changes within the system. The execution of the machine begins from a start state and continues until it reaches an accept state. In this section, a Finite State Machine is developed based on the state transition diagram.

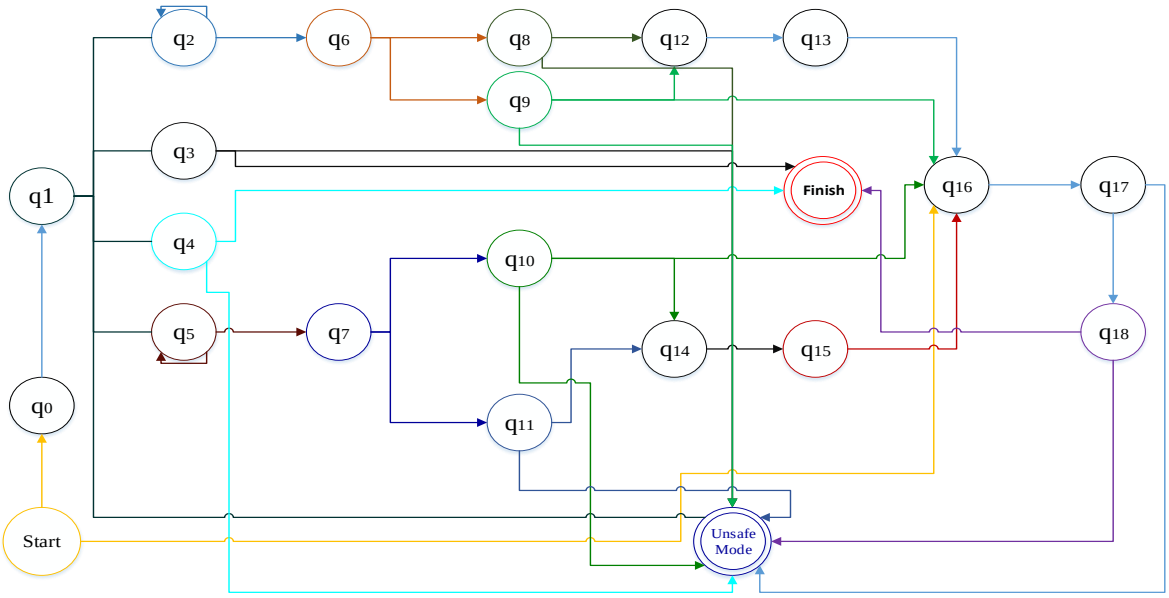

Figure 3. Finite State Machine of Data & Command Unit

The transition table for the above finite state machine is created which shown in ***Table 1***. From the above FSM, the transformation of states from one state to another state on the basis of {q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}, q_{11}, q_{12}, q_{13}, q_{14}, q_{15}, q_{16}, q_{17}, q_{18}} events. These events are considered as terminals for the above finite state machine of Data and Command Unit. The various productions can be induced for the above finite state machine and the corresponding transition table is as shown below in ***Table 1.*** Finite state machines (FSMs) are similar to Markov chains in the respect that both have states and transitions between states. FSMs can be seen as graphical presentations of a Markov chain; Probabilities are assigned to state transitions. Thus an FSM can efficiently be converted to a finite-state, discrete-parameter Markov chain. Furthermore, they allow the use of the same reliability modelling principles that were suggested with Markov chains. Usually, FSMs are applied as a tool for designing Markovian usage models and to present them graphically.

Table 1. The Transition for Finite State Machine

State	State Description
q_0	Read Pressure from Engine Pressure Sensor
q_1	Sensor
q_2	Engine Shutdown Detection
q_3	Engine Separation
q_4	Nose Separation
q_5	Read Analog Pressure Data
q_6	Read Digital Pressure Data
q_7	Calculate Average-D
q_8	Calculate Average –A
q_9	Height Detection10-D
q_{10}	Height Detection17-D
Q_{11}	Height Detection17-A
q_{12}	Height Detection10-A
q_{13}	Pressure15-Pressure 10 D
q_{14}	Pressure15-Pressure 10 A
q_{15}	Height7-Reliable Sensor D
q_{16}	Height7-Reliable Sensor A
q_{17}	Reserve Parachute Separation
q_{18}	Main Parachute Separation

3. FSM Markovian Chains Model

The Markov model offers superior capabilities compared to other combinational methods and is particularly effective in solving systems with dynamic and interdependent behavior. However, it also possesses notable drawbacks, which become exponentially more pronounced as the system size increases [11-13]. The exponential growth in the number of states can lead to impractical and arbitrary models. Consequently, researchers have focused on approximate combinatorial methods that generate only a subset of the Markov chain mode space. Furthermore, the Markov model assumes an exponential distribution for the time-to-failure parameter [14], whereas combined methods can accommodate arbitrary failure distributions. The fundamental concepts in Markov chain modeling are system states and state transitions. A system state represents a specific combination of system parameters that describes the system at any given moment. In the context of system reliability, each state in the Markov model typically represents a unique combination of functioning components. State transitions, on the other hand, track the changes in the system state over time, particularly in the event of failures. The system transitions from one state to another (often to a failure state) based on parameters such as failure rates, error coverage factors, and repair rates [15]. Solving a Markovian model involves solving a set of differential equations, which can be expressed in the following form:

$$A \cdot P(t) = P'(\mathrm{t}) \xrightarrow{\text{yields}} \begin{bmatrix} -a_{11} & a_{12} & a_{13} & \cdots & a_{1j} \\ a_{21} & -a_{22} & a_{23} & \cdots & a_{2j} \\ a_{31} & a_{32} & -a_{33} & \cdots & a_{3j} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{j1} & a_{j2} & a_{j3} & \cdots & -a_{jj} \end{bmatrix} \cdot \begin{bmatrix} P_1(t) \\ P_2(t) \\ P_3(t) \\ \cdots \\ P_n(t) \end{bmatrix} = \begin{bmatrix} P'_1(t) \\ P'_2(t) \\ P'_3(t) \\ \cdots \\ P'_n(t) \end{bmatrix} \quad \textbf{(1)}$$

In the given equation (***Eq. (1)***), the transfer rate from state j to state k is denoted by a_{jk} ($j \neq k$). The diagonal elements a_{jj} in matrix A represent the sum of state transition rates from state j. That is, $a_{jj} =$

$\sum_{k=1,k\neq j}^{n} a_{jk}$. Consequently, the sum of each column in matrix A is equal to zero, indicating that the total transition rates out of each state balance with the incoming transition rates.

In the equation, $P_i(t)$ represents the probability of the system being in state i at time t. The parameter n denotes the number of states in the Markov model. To solve the differential equation, the Laplace transform function can be utilized, which provides the solution including the probabilities of the system being in any state.

The unreliability or unsafe mode of the system can be determined by summing the probabilities of failure for each state, i.e., $\sum_{\forall F} P_{F_i}(t)$, where $P_{F_i}(t)$ represents the probability of the system being in the unsafe or failure mode (F_i) at time t.

The reliability assessment method involves constructing the Markov model of the system under investigation. If the model comprises n distinct states, the probability vector of state occurrence can be defined using ***Eq. (2)***.

$$[P(t)]^T = [P_1(t)P_2(t) \dots P_n(t)] \quad \textbf{(2)}$$

In the context of the Markov model, the reliability of a system can be determined by evaluating the probabilities of all possible states in which the system is operational. This can be calculated using ***Eq. (3)***.

$$R_s(t) = \sum_{j \in Reliable\ States} P_j(t) \quad \textbf{(3)}$$

By analyzing the Markov model and computing the probabilities of each state, we can assess the reliability of the system. This assessment provides valuable insights into the system's performance and helps in identifying potential areas of improvement to enhance its reliability.

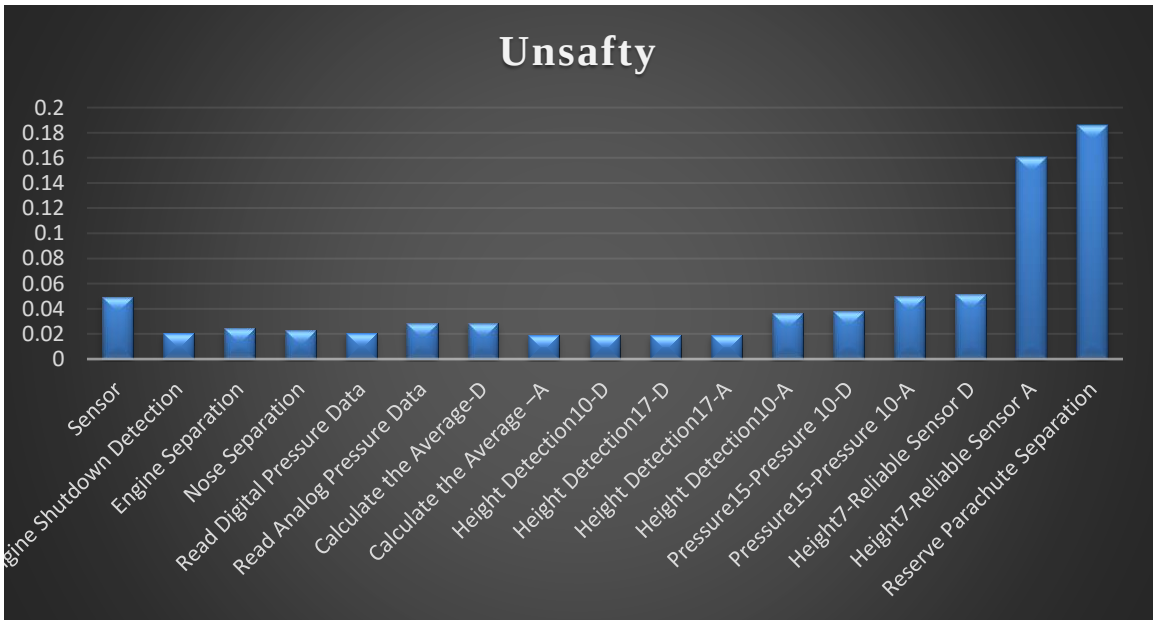

Figure 4. Unsafety each state

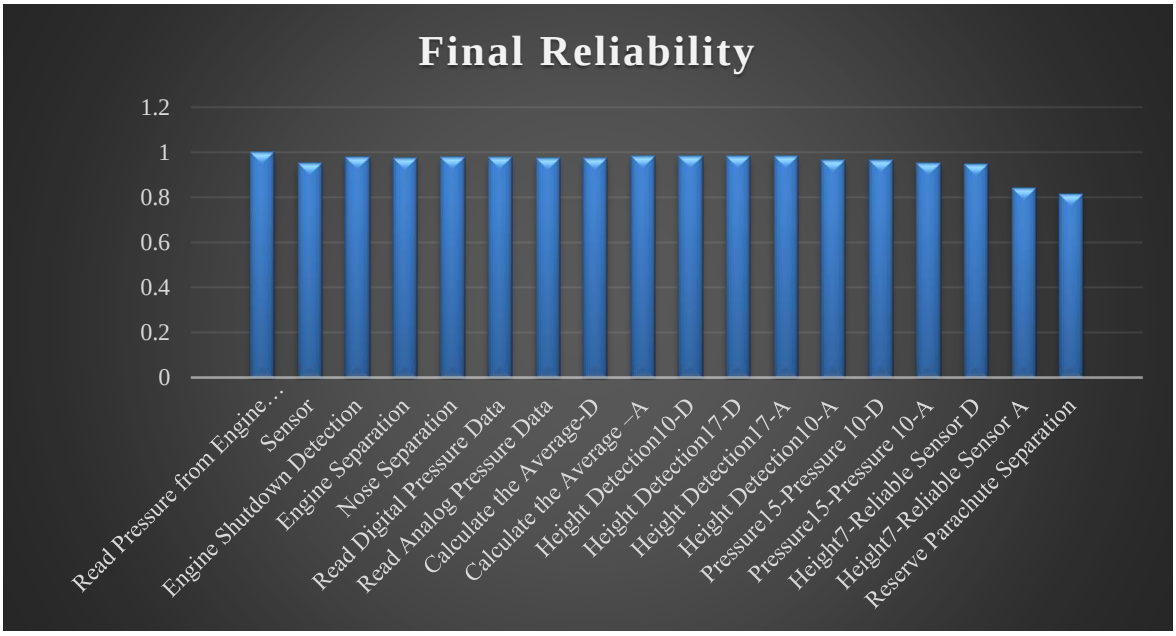

Figure 5. Final reliability each state

4. Conclusion

Software plays a critical role in industries where system failures can have severe consequences. A single software failure has the potential to cause irreparable damage. It is imperative to recognize that research conducted in the United States has shown that more than seventy percent of adverse events attributed to software could have been prevented through proactive measures such as forecasting, error detection, and elimination methods. Therefore, it is crucial to prioritize software reliability prediction methods before implementing software-based systems.

Building upon the significance of examining the reliability of critical systems, this paper proposes a novel approach Given the importance of examining the reliability of critical systems, this paper presents a new method for evaluating and investigating the reliability and unsafety of systems. This method utilizes test model analysis and test cases to prepare the State Transition Diagram model, followed by the transformation of the FSM model to assess reliability. By transforming the obtained FSM model into a Markov Chain model, we were able to evaluate and investigate the reliability of each system state. For future research, we recommend conducting software reliability analysis using other statistical models, such as regression models.

References

[1] Oveisi, S., Moeini, A., Mirzaei, S., & Farsi, M. A. (2023). Software reliability prediction: A survey. *Quality and Reliability Engineering International*, *39(1)*, 412-453.

[2] Oveisi, S., Nadjafi, M., Farsi, M. A., Moeini, A., & Shabankhah, M. (2020). Design software failure mode and effect analysis using Fuzzy TOPSIS based on Fuzzy entropy. *Journal of Advances in Computer Engineering and Technology*, *6*(3), 187-200.

[3] Modarres, M., Kaminskiy, M. P., & Krivtsov, V. (2016). Reliability engineering and risk analysis: a practical guide. *CRC press.*

[4] Kamandi, A., Azgomi, M. A., & A. Movaghar, A. (2006). Transformation of UML models into analyzable OSAN models. *Electronic Notes in Theoretical Computer Science*, *159*, 3-22.

[5] Duo, W., Zhou, M., & Abusorrah, A. (2022). A survey of cyber-attacks on cyber physical systems: Recent advances and challenges. *IEEE/CAA Journal of Automatica Sinica*, *9(5)*, 784-800.

[6] Kamandi, A., Abdollahi Azgomi, M., & Movaghar, A. (2004). A Modelling Tool for Object Stochastic Activity Networks. *In Proc. 9th Annual CSI Computer Conf. (CSICC'04), Tehran, Iran,* 408-419.
[7] Rautakorpi, M., (1995). Application of Markov Chain Techniques in Certification of Software. *Helsinki University.*
[8] Rausand, M., & Høyland, A. (2004). System reliability theory: models, statistical methods, and applications. *John Wiley & Sons, 396.*
[9] Salman, Y. D. A. (2014). Test Case Generation Framework Based On Uml Statechart Diagram. *Lecture Notes in Electrical Engineering.*
[10] Zakaria, N. N., Othman, M., Sokkalingam, R., Daud, H., Abdullah, L., & Abdul Kadir, E. (2019). Markov Chain Model Development For Forecasting Air Pollution Index Of Miri, Sarawak. *Sustainability, 11(19).*
[11] Barbosa, G., de Souza, É. F., dos Santos, L. B. R., da Silva, M., Balera, J. M., & Vijaykumar, N. L. (2022). A Systematic Literature Review on prioritizing software test cases using Markov chains. *Information and Software Technology, 147.*
[12] Rebelo, L., Souza, É., Berkenbrock, G., Barbosa, G., Silva, M., Endo, A., ... & Trubiani, C. (2023). Prioritizing Test Cases with Markov Chains: A Preliminary Investigation. *In IFIP International Conference on Testing Software and Systems,* 219-236. Cham: Springer Nature Switzerland.
[13] Kashyap, A., Holzer, T., Sarkani, S., & Eveleigh, T. (2012). Model based testing for software systems: an application of markov modulated markov process. *International Journal of Computer Applications, 46(14),* 13-20.
[14] Ding, Y., Li, W., Zhong, D., Huang, H., Zhao, Y., & Xu, Z. (2018). System states transition safety analysis method based on FSM and NuSMV. *In Proceedings of the 2018 2nd International Conference on Management Engineering, Software Engineering and Service Sciences*, 107-112.
[15] Saxena, V., & Kumar, S. (2012). Validation of UML Class Model through Finite-State Machine. *International Journal of Computer Applications, 41(19).*