RESEARCH PAPER

Design and Assurance of Safety Critical Software using STPA

P. Rahul [1*], S.Rakesh [2*], V.Naveen [3*] , R.Ravinandhan [4*]

[1*] Assistant Professor, Department of Computer Science and Engineering, V.S.B College of Engineering Technical campus, kinathukadavu, Coimbatore, Tamilnadu, India.

[2*] UG Scholar, Department of Computer Science and Engineering, V.S.B College of Engineering Technical campus, kinathukadavu, Coimbatore, Tamilnadu, India

[3*] UG Scholar, Department of Computer Science and Engineering, V.S.B College of Engineering Technical campus, kinathukadavu, Coimbatore, Tamilnadu, India.

[4*] UG Scholar, Department of Computer Science and Engineering, V.S.B College of Engineering Technical campus, kinathukadavu, Coimbatore, Tamilnadu, India.

***Abstract*—**Modern software-driven systems, like self-driving cars, digital healthcare platforms, and smart infrastructure networks, are are becoming increasingly complex and interconnected. In these kinds of settings, conventional software engineering methods that rely on component-level verification and exhaustive testing are no longer sufficient to ensure system safety. These methods often fail to address flawed requirements, unsafe control actions, or unanticipated interactions among system components. Recent failures in safety-critical systems underscore the necessity of adopting a broader, system-level perspective when designing software. This work presents a control-oriented, STPA-based, model-driven software design method that incorporates hazard analysis directly into the architectural development process. Based on Nancy G. Leveson's system-theoretic safety ideas, the approach treats software not merely as a passive component but as an active controller that enforces safety constraints derived from System-Theoretic Process Analysis (STPA). The proposed framework strengthens traceability by linking identified system hazards to software design decisions and verification activities. Instead of relying mostly on testing after development, safety requirements are built directly into the software logic through clearly defined control constraints and structured process models. It also has a hazard-focused change impact analysis tool to help make system updates safer and keep the system from changing too quickly. This approach is particularly well-suited for complex cyber-physical systems and human-in-the-loop environments because it reduces the need for repetitive safety assessments and enhances assurance confidence.

***Keywords*—*System*-**Theoretic Process Analysis (STPA), Control-Theoretic Software Design, Hazard Analysis, Safety-Critical Systems, Cyber-Physical Systems, Software Safety Constraints, Model-Based System Engineering (MBSE), Software Architecture Assurance.

How to cite this article: Rahul P, Rakesh S, Naveen V, Ravinandhan R. Design and assurance of safety critical software using STPA. Int J Drug Deliv Technol. 2026;16(74s): 2049-2056. DOI: 10.25258/ijddt.16.74s.165

I. INTRODUCTION

Safety-critical software systems are fundamental to modern cyber-physical infrastructures like aerospace control systems, medical devices, railway signalling networks, automotive platforms, and industrial automation environments. In these areas, software decisions directly influence physical processes and human safety. Any malfunction or unsafe behaviour may result in serious consequences, including personal injury, environmental harm, financial loss, and erosion of public trust. As systems get more complicated, independent, and connected, ensuring that software operates safely and reliably has become a critical challenge in modern engineering [4], [5].

Most traditional approaches to safety assurance are grounded in reliability theory and probabilistic risk analysis. The Failure Mode and Effects Analysis (FMEA) and Fault Tree Analysis (FTA) are used to find possible component failures and estimate how likely they are to occur. These methods work well for systems that are mostly hardware based, but they are less effective for predominantly software-based systems. Software does not degrade in the manner that hardware does. Instead, unsafe outcomes frequently result from erroneous design assumptions, defective control logic, unforeseen system interactions, inadequate feedback mechanisms, or incomplete requirements [6].

Recent incidents in aviation, autonomous vehicles, and industrial control systems have demonstrated that accidents frequently arise not from isolated component failures, but from complex interactions among software, hardware, and human operators. In highly integrated systems, these interactions can lead to dangerous situations that traditional failure-based models struggle to predict. As systems become more complex, the number of possible ways for them to interact grows quickly. This renders exhaustive testing both economically infeasible and technically insufficient for safety assurance [7].

An alternative perspective considers safety as a control problem rather than purely a reliability issue. Nancy G. Leveson developed System-Theoretic Process Analysis (STPA) based on this systems-theoretic view of safety.

STPA describes accidents as the consequence of insufficient control or the failure to implement safety constraints within hierarchical control frameworks. The method examines not only component failures; it also looks at unsafe control actions, insufficient feedback, and incorrect process models that could lead the system to dangerous states [8].

This approach is particularly suited to complex systems involving software-human interaction. STPA is often applied only for hazard identification during system design, even though it has its benefits. However, the outputs of STPA are not consistently integrated into the software architecture in many development processes. As a result, safety restrictions found during analysis may not be consistently applied in the software design. This disconnects between safety analysis and implementation can create traceability gaps, complicate certification, and increase reassessment costs when system changes occur [9].

This paper proposes a control-oriented, STPA-driven software design framework that integrates hazard analysis directly into architectural development to address these limitations. The proposed approach treats software as an active controller operating within a feedback loop, responsible for enforcing STPA-derived safety constraints. The identified constraints are systematically translated into explicit control rules embedded within the system's decision-making logic. This approach enforces safety proactively before control actions are executed, rather than reactively verifying them afterwards **[**10**]**. In addition, the proposed framework creates a structured way to trace system hazards, unsafe control actions, safety constraints, software modules, and verification artifacts. This traceability enhances assurance transparency and facilitates certification processes mandated by safety standards [11], [12**]**.

A hazard-driven change impact analysis mechanism is also introduced to make system maintenance safer and evolution more controlled. This lowers the cost of recertification and long-term lifecycle costs.

II. Related Works

The Systems-Theoretic Accident Model and Processes (STAMP), created by Nancy G. Leveson, represented a significant advancement in this domain by framing safety as a control problem. This model posits that accidents transpire when safety regulations are insufficiently enforced within hierarchical control structures, as explained by Leveson [1] and further extended [2]. System-Theoretic Process Analysis (STPA) was created based on STAMP as a systematic way to look for hazards that find Unsafe Control Actions (UCAs) that can put a system in dangerous states. Unlike traditional methods such as FMEA and FTA, STPA does not rely on probabilistic failure estimation. Thomas et al. Furthermore, reliability-based safety analysis methods frequently struggle to verify negative safety requirements, like making sure that a certain control action is never taken in certain situations. These limitations have led researchers to explore alternative safety paradigms that emphasise system behaviour, control structures, and interactions over failure probabilities [3].

Zhang et al. Historically, safety assurance in critical domains has relied extensively on probabilistic risk assessment and failure-oriented hazard analysis. Failure Mode and Effects Analysis (FMEA), Fault Tree Analysis (FTA), and Event Tree Analysis (ETA) are some of the most common methods used in aerospace, nuclear energy, automotive systems, and industrial automation. These methods primarily focus on identifying component failures and evaluating their probability of occurrence. The underlying assumption is that accidents primarily result from hardware faults or random component failures. These methods are effective for predominantly hardware-based systems but are less applicable to software-intensive systems [4].

In systems that are very integrated and tightly coupled, accidents more commonly arise from unsafe interactions among software, hardware, and human operators than from individual component failures. Conventional failure-based methodologies often presuppose the independence of system components, an assumption that is rendered invalid in intricate interconnected architectures [5]. Studies indicate that STPA is particularly effective at identifying hazards arising from software logic errors, timing discrepancies, and human–machine interface deficiencies that are difficult to detect through component-failure analysis alone [6], [7].

Chen and Wang et al. Unlike hardware, software does not degrade over time. Instead, software-related risks typically arise from design errors, incorrect assumptions, incomplete requirements, unintended interactions, or inadequate coordination among system components. Consequently, probabilistic modelling alone is insufficient to fully characterise design-induced hazards in modern cyber-physical systems [9] and further emphasized.

ecent research has demonstrated the effectiveness of STPA across numerous safety-critical domains, such as advanced driver-assistance systems (ADAS), aviation flight control systems, railway signalling networks, medical devices, and industrial automation systems [11]. Instead, it looks at how wrong control actions, delayed or missing feedback, timing problems, or wrong process models can lead to unsafe outcomes. By evaluating the complete system control structure, STPA provides a comprehensive perspective that identifies interaction-related hazards and systemic coordination failures that traditional methodologies may overlook [12].

Kim and Lee et al. Despite these advantages, much of the existing literature focuses primarily on using STPA to identify and document hazards. In many cases, STPA outputs are translated into high-level safety requirements but are not consistently integrated into the software architecture. This separation between safety analysis and implementation can create gaps in traceability, make the enforcement of safety constraints more difficult, and increase the complexity of the certification process [17].

Brown and Miller et al. Furthermore, in conventional development processes, safety analysis, system design, and verification are typically treated as distinct sequential phases [19]. Such fragmentation increases the risk that safety requirements may not be consistently enforced or adequately verified. As systems evolve, maintaining alignment between hazard documentation and software implementation becomes increasingly difficult. This frequently results in substantial expenditure on re-evaluation and re-certification activities [20].

To address these limitations, this study proposes a control-theoretic software design framework directly informed by STPA outcomes. The proposed framework

integrates hazard identification, constraint derivation, controller architecture design, and certification-oriented verification into a unified development process. By treating software as an active controller that enforces safety constraints, safety becomes an inherent property of the system architecture rather than a post-development consideration. Through this integrated approach, the framework improves hazard traceability, enhances assurance transparency, and simplifies certification processes throughout the lifecycle of safety-critical systems.

III. PROPOSED METHODOLOGY

A. Safety Analysis Based on STPA

The proposed framework initiates the design process by applying System-Theoretic Process Analysis (STPA) to identify potential system-level hazards and unsafe control actions.

Unlike traditional fault-based methods that focus on component failures, STPA treats safety as a control problem within a hierarchical control structure. The analysis begins by defining unacceptable system losses and identifying hazards that could lead to those losses. Subsequently, the system's control actions are systematically examined to determine how they may contribute to hazardous conditions. STPA identifies four categories of unsafe control actions (UCAs):

A required control action is not provided when needed.

A control action is executed at an incorrect time or in an incorrect sequence.

A control action is applied for too long or terminated prematurely.

For each identified unsafe control action, detailed causal scenarios are developed to examine how flawed control logic, incorrect process models, or inadequate feedback mechanisms may cause the system to behave unsafely. The outcome of this step is a set of formally defined safety constraints that must be enforced within the software system.

B. Control-Oriented Software Design

Following hazard identification, the software architecture is developed using a control-theoretic approach. In this model, the software functions as a controller operating within a closed-loop feedback system.

The controller:

- Receives sensor inputs,
- Maintains an internal process model representing the current system state,
- Issues commands to actuators.

The internal process model is grounded in assumptions about the system state and operating environment. Prior to executing any control action, the software validates the action against STPA-derived safety constraints.

Let:

S(t) represents the system state at time t

U(t) represents the intended control action

C(t) represents the applicable safety constraints

A control action is permitted only if it satisfies the established safety constraints: Execute U(t) only if U(t) □ C(S(t))

This condition ensures that hazardous control actions are prevented proactively. The feedback loop continuously updates the process model with real-time sensor data, mitigating risks arising from incorrect or outdated state assumptions.

C. Software Requirements Based on Hazards

In this methodology, software requirements are systematically derived from identified hazards rather than solely from functional specifications.

Each hazard is translated into one or more enforceable safety constraints. These constraints are subsequently formalised as structured software requirements and embedded directly into the system logic.

The derivation follows a formally structured chain:

Hazard → Unsafe Control Action → Safety Constraint → Software Requirement → Logic for Implementation

This structured transformation ensures consistent alignment between hazard analysis artefacts and executable software

phase, the framework reduces ambiguity, closes requirement gaps, and strengthens design integrity.

D. Safety Through Hazard Traceability

A significant benefit of this framework is that it enables comprehensive end-to-end traceability.
Each identified hazard is tracked throughout the development and verification lifecycle using the following traceability chain:
Hazard → Safety Constraint → Software Module → Implementation → Test Case → Verification Evidence

This structured linkage enables:
Transparent and justifiable safety rationale
Reduced effort during recertification
Enhanced documentation for regulatory compliance

E. Verification for Certification

The following structured metrics are used to generate certification evidence:

- Percentage of hazard coverage
- Verification of safety constraint implementation
- Full traceability
- Process model accuracy

This structured verification strategy supports compliance with safety standards such as DO-178C, ISO 26262, and IEC 61508. By closely linking verification with hazard analysis, the framework offers more rigorous and justifiable safety assurance than conventional testing methods.

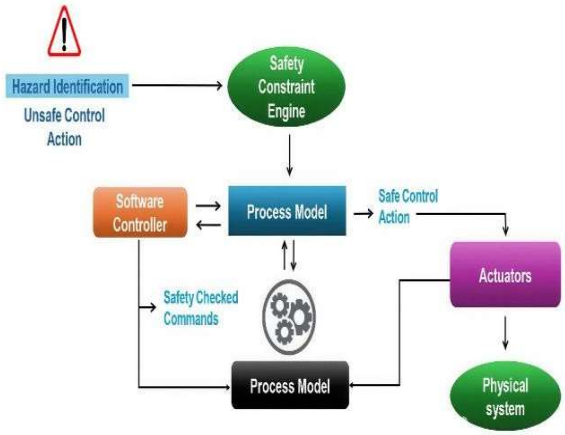

Fig. 1. STPA Architecture

The proposed framework aligns validation activities directly with hazard analysis results, which differs from traditional verification methods that focus primarily on functional correctness and reliability metrics. Fig. 1 illustrates the STPA Architecture. Hazard-focused testing is employed to verify safety behaviour. Each unsafe control action is linked to validation scenarios designed to replicate hazardous conditions. These tests confirm that the

implemented safety constraints are effective in preventing unsafe outcomes.

Algorithm STPA-Based Safety Assurance

Step 1: Identify System Losses and Hazards

The safety analysis begins by identifying potential system losses and hazards that may affect system operation. System losses represent unacceptable outcomes that must be avoided during system execution. The set of system losses is defined as L = {l1, l2, ..., ln}, where L denotes the set of unacceptable system losses. Hazards that may lead to these losses are defined as H = {h1, h2, ..., hm}, where H represents hazardous system states that may cause system failures.

Step 2: Perform STPA Hazard Analysis

System-Theoretic Process Analysis (STPA) is performed to examine how unsafe system behaviour may occur through improper control actions. The system control actions are represented as CA = {ca1, ca2, ..., cak}, where CA represents the set of control actions executed by the system controller. Each control action is evaluated under unsafe conditions such as missing actions, incorrect actions, wrong timing, or actions applied too long or stopped too soon.

Step 3: Generate Unsafe Control Actions

Unsafe control actions that may lead to hazards are identified as UCA = {uca_1, uca_2, …, uca_p}. These unsafe control actions describe situations where control actions may trigger hazardous system states. Each unsafe control action is carefully analysed to understand how incorrect, missing, or improperly timed control actions can affect system safety. Additionally, UCAs are categorized based on conditions such as providing a control action when it is not required, not providing a control action when it is needed, applying a control action too early or too late, or stopping it too soon or applying it for too long. This analysis helps identify potential scenarios that may lead to unsafe system behaviour. By systematically identifying UCAs, designers can better understand system vulnerabilities and improve the overall safety of the system.

Step 4: Derive Safety Constraints

Safety constraints are derived from unsafe control actions to prevent hazardous system behaviour. The constraints are represented as SC = {sc1, sc2, ..., scq}. These constraints ensure that control actions are executed only when safety conditions are satisfied. Each safety constraint defines a rule that restricts the system from performing actions that could lead to unsafe states.

Step 5: Embed Safety Constraints

The derived safety constraints are integrated into the software controller logic to ensure safe system operation. These constraints act as safety rules that guide the controller

in selecting and executing appropriate control actions while preventing hazardous behaviour.

Step 6: Runtime Safety Monitoring

During system execution, safety monitoring is performed using sensor data represented as SD = {sd1, sd2, ..., sdr}. The sensor data updates the process model and ensures that safety constraints are validated before executing control actions.

Step 7: Hazard-Based Verification

Verification is performed by generating test scenarios based on hazards H. Validation tests ensure that safety constraints are correctly enforced and that the system avoids unsafe states.

Step 8: Safety Assurance Decision

If all safety constraints SC are satisfied, the system is approved for deployment. Otherwise, the controller logic is refined, and the verification process is repeated until safe system operation is ensured.

IV. PERFORMANCE ANALYSIS

Structured safety assurance metrics are employed to evaluate the effectiveness of the proposed STPA-driven, control-oriented software design framework. Unlike traditional evaluation methods that focus primarily on reliability metrics or defect density, the proposed evaluation examines hazard mitigation robustness, traceability integrity, verification alignment, and lifecycle sustainability.

The framework is evaluated according to the following assurance criteria:

- Hazard Coverage
- Traceability Completeness
- Verification Transparency

Each metric represents a distinct dimension of safety effectiveness.

A. Ratio of Hazard Coverage (HCR)

Hazard coverage measures the extent to which identified hazards are addressed through implemented safety constraints and associated validation mechanisms. In safety-critical environments, unmitigated hazards directly contribute to residual risk.

Let:
H *total* = Total number of hazards identified by STPA
H *covered* = Number of hazards addressed by implemented safety constraints
The Hazard Coverage Ratio (HCR) is

$$HCR = H_{covered} / H_{total} \quad (1)$$

An HCR value approaching 1.0 indicates that all identified hazards have been systematically mitigated. In the proposed architecture, each Unsafe Control Action (UCA) identified during STPA is associated with one or more enforceable safety constraints integrated within the Safety Constraint Engine. This structured derivation improves coverage compared to traditional failure-based validation methods, which may not find all hazards or only find them by chance.

B. Traceability Completeness Metric (TCM)

Traceability ensures that each identified hazard is explicitly linked to its corresponding safety constraint, software component, and verification artefact. Weak traceability often leads to safety gaps that aren't documented and makes it harder to review certifications.

Let:
T *valid* = Number of hazards with a complete traceability chain
T *total* = Total number of identified hazards

The Traceability Completeness Metric (TCM) is defined as

$$TCM = T\ valid / T\ total \quad (2)$$

The suggested method makes sure that this structured mapping is used during development. This makes it easier to audit, follow the rules, and create structured safety documentation. Fig. 2 illustrates the Traceability Completeness Metric.

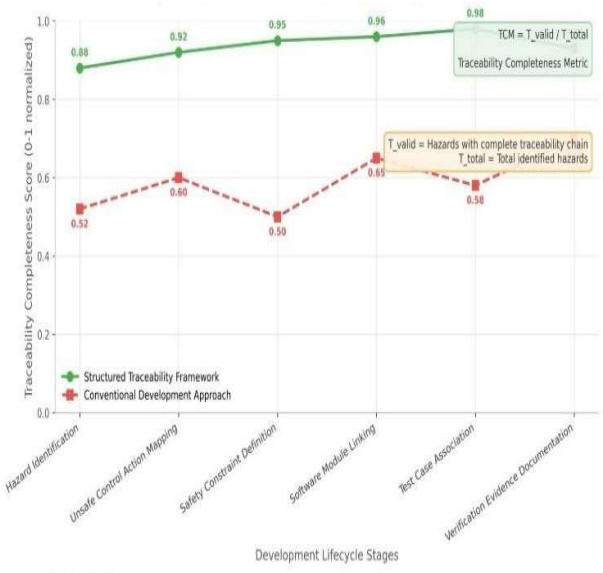

Fig. 2. Traceability Completeness Metric

C. Verification Transparency Index (VTI)

Verification transparency gauges the extent to which validation processes are clearly associated with hazard mitigation goals. Conventional testing methodologies frequently emphasize functional validation without direct correlation to hazard scenarios.

Let:
TC *hazard* = Number of hazard-based test cases
TC *total* = Total number of test cases
The Verification Transparency Index (VTI) is defined as

$$VTI = TC\ hazard / TC\ total \quad (3)$$

A higher VTI indicates greater alignment between testing activities and identified hazards. In the proposed framework, each Unsafe Control Action generates its own dedicated validation scenarios. This ensures that verification activities are grounded in hazards rather than defects alone, thereby strengthening certification evidence.*Maintenance Effort Reduction (MER)*In safety-critical systems, maintenance and system evolution represent two of the most significant cost drivers. In conventional architectures, system changes often necessitate extensive reassessment due to the absence of a clear mapping between hazards and implementation artefacts.
Maintenance Effort Reduction (MER) is quantified as

$$MER = E\ traditional - E\ proposed / E\ traditional \quad (4)$$

E *traditional* = Effort required for change impact analysis in conventional systems
E *proposed* = Effort required under structured hazard traceability

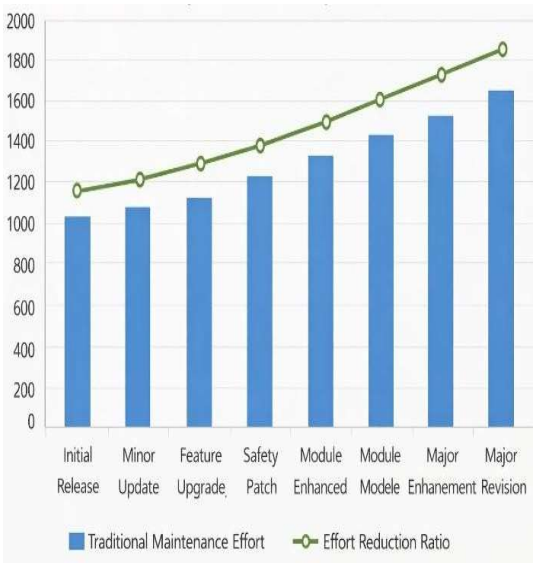

Fig. 3. Maintenance Effort Reduction

Because the proposed framework maintains explicit hazard-to-module mapping, impact analysis becomes localised and systematic. A higher MER value indicates a greater reduction in maintenance effort achieved through the proposed framework. Fig. 3 illustrates the comparative reduction in maintenance effort.

D. Comparative Analysis

A qualitative comparison between traditional failure-based methodologies and the proposed STPA-driven approach is outlined below: Fig. 4 illustrates the Comparative Analysis.

TABLE.I Comparative Analysis

Evaluation Parameter	*Traditional Approach*	*Proposed Framework*
Hazard Coverage	Partial	Structured & Comprehensive
Traceability	Limited	End-to-End Structured Mapping
Verification Focus	Functional Testing	Hazard-Centric Validation

Maintenance Impact	Extensive Reassessment	Localized Impact Analysis

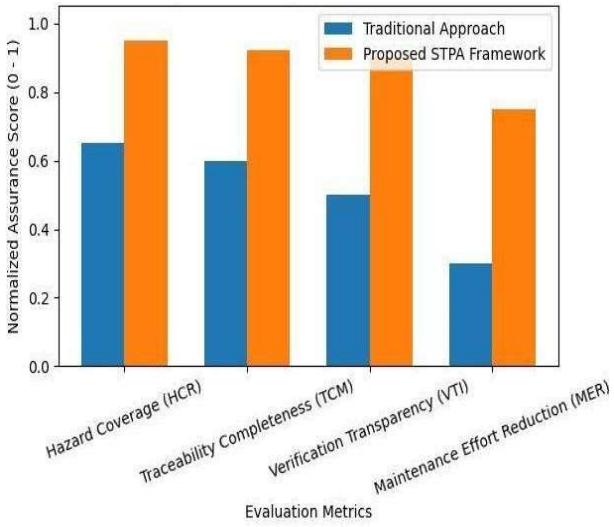

Fig. 4. Comparative Safety Assurance Evaluation

I. CONCLUSION

This paper introduced a control-theoretic software design framework grounded in System-Theoretic Process Analysis (STPA) for the development of safety-critical systems. The proposed approach treats safety as a control problem and systematically embeds safety constraints into the software architecture. This distinguishes it from traditional failure-oriented methods. By tightly coupling hazard identification, constraint implementation, and end-to-end traceability, the framework improves assurance transparency, enhances certification readiness, and strengthens long-term system maintainability. Future work will extend this framework to support integration with Agile and DevOps methodologies, facilitating continuous safety consideration throughout the development lifecycle. Hazard analysis and safety constraints will be incorporated into sprint planning, automated validation workflows, and continuous integration pipelines. The objective of this integration is to accelerate development cycles while maintaining rigorous safety standards for complex cyber-physical systems.

REFERENCES

[1] Leveson, N.G. 'Design and assurance of control software', *IEEE Transactions on Software Engineering*, 2025.

[2] Thomas, J.P., Suo, M. and Leveson, N.G. 'System-Theoretic Process Analysis (STPA) for safety-critical systems', *Safety Science*, 124, p.104567, 2020.

[3] Abdul Khaleq, A. and Wagner, S. 'Safety analysis in complex software systems using system-theoretic process analysis', *Journal of Systems and Software*, 168, p.110643, 2020.

[4] Zhang, Y., Chen, L. and Patel, M. 'Safety analysis of cyber-physical systems using system-theoretic approaches', *IEEE Access*, 10, pp.54623–54637, 2022.

[5] Rahman, M., Lee, T. and Park, K. 'Control-theoretic safety analysis for cyber-physical systems', *IEEE Transactions on Reliability*, 72(2), pp.614–625, 2023.

[6] Gupta, S. and Sharma, R. 'STPA-based safety assurance framework for autonomous systems', *IEEE Software*, 40(4), pp.45–53, 2023.

[7] Sun, Y. and Li, H. 'Safety assurance for autonomous systems using system-theoretic hazard analysis', *IEEE Access*, 12, pp.11345–11360, 2024.

[8] Gupta, A. and Mehta, K. 'Continuous safety assurance in DevOps pipelines for critical systems', *IEEE Software*, 41(2), pp.60–68, 2024.

[9] Chen, L. and Wang, H. 'Integrating hazard analysis into software architecture for safety-critical applications', *IEEE Access*, 11, pp.101234–101247, 2023.

[10] Sathyanarayanan, M., Manivannan, K., Mithin, K., Manoj, G., Mugesh, M. and Kishore, N., "Design and Implementation of a Cybersecurity Framework for Encrypted File Management System," in 2025 6th International Conference for Emerging Technology (INCET) (pp. 1-6). IEEE, 2025.

[11] Mary, P.A., Manivannan, K., Sathish, S., Saran, V.V., Saran, R. and SharukHussain, M., "Automated Parking Management in Urban Environment Using Deep Learning and OCR Technique," in 2025 International Conference on Computational Robotics, Testing and Engineering Evaluation (ICCRTEE) (pp. 1-6). IEEE, 2025.

[12] Pathak, D., Gowda, D., Manivannan, K., Aghav, S., Srinivas, V. and Gireesh, N., "Advanced Machine Learning Approaches to Evaluate User Feedback on Virtual Assistants for System Optimization," in 2024 2nd International Conference on Sustainable Computing and Smart Systems (ICSCSS) (pp. 1140-1147). IEEE, 2024

[13] Dwivedi, A., Manivannan, K., Kumar, S.K., Anand, N., Perwej, Y. and Kamra, R., "A Real-Time Environmental Pollution Monitoring Framework Using IoT and Remote Sensing Technologies," International Journal of Environmental Sciences, 11(7s), pp.1064-1075, 2025.

[14] Manivannan, K., Anuprathibha, T., Kamaleshwaran, P., Madhan Kumar, S., Roshanmahanama, V. and Sabareesh, B., "Smart Ed: Utilizing AI to Deliver Personalized Learning Experiences on Mobile Platforms," in 2025 6th International Conference for Emerging Technology (INCET) (pp. 1-5). IEEE, 2025

[15] Kumar, A. and Singh, P. 'Hazard identification in safety-critical software using STPA', *Journal of Systems and Software*, 188, p.111234, 2022.

[16] Lee, K. and Park, J. 'System-level hazard analysis for autonomous vehicles using STPA', *IEEE Transactions on Intelligent Transportation Systems*, 23(9), pp.15789–15800, 2022.

[17] Kim, S. and Lee, J. 'Safety assurance framework for complex autonomous systems', *IEEE Systems Journal*, 15(4), pp.5432–5443, 2021.

[18] Gupta, R. and Nair, P. 'Applying system-theoretic safety analysis to software-intensive systems', *IEEE Access*, 9, pp.87654–87667, 2021.

[19] Rodriguez, P. and Martinez, L. 'Control-based hazard analysis for industrial automation systems', *Journal of Systems and Software*, 181, p.111056, 2021.

[20] Brown, D. and Miller, T. 'Hazard-driven software verification for safety-critical systems', *Journal of Software: Evolution and Process*, 33(8), e2345, 2021.