The American Journal of Engineering and Technology
ISSN 2689-0984 | Open Access

TYPE Original Research
PAGE NO. 82-90
DOI 10.37547/tajet/Volume07Issue02-11

Check for updates

Methodology for rapid error detection in web applications

Maksim Zemskov

Lead Software Engineer, Yandex, Serbia, Belgrade

OPEN ACCESS

SUBMITTED 24 December 2024
ACCEPTED 26 January 2025
PUBLISHED 28 February 2025
VOLUME Vol.07 Issue02 2025

CITATION

Maksim Zemskov. (2025). Methodology for rapid error detection in web applications. The American Journal of Engineering and Technology, 7(02), 82–90. https://doi.org/10.37547/tajet/Volume07Issue02-11

Abstract: The article is aimed at researching and describing effective approaches and methods for quickly detecting and identifying errors in web applications while operating in production environments. This, in turn, is due to the fact that in modern conditions, the speed of detection and elimination of defects is important to ensure reliable operation of web applications.

The relevance of this topic is driven by the increasing transition of business processes to the online space. As more companies and human activities become dependent on software reliability, defects in commercial software products can lead to significant financial losses, reputational risks, and loss of user base. The growing complexity of web applications and their increasing role in critical business operations further emphasizes the importance of robust error detection methodologies. Therefore, timely detection and elimination of errors has become a vital business necessity.

The methodology described in this article represents a promising approach to rapid error detection in web applications, offering a systematic framework for monitoring and managing software errors in real-time. The methodology includes detailed recommendations and practices for identifying errors efficiently, even in software products handling high-volume error streams with substantial user loads.

The article will be useful for software developers, engineering managers, DevOps specialists, and researchers in the field of analysis and diagnostics of web applications. It provides a description of the techniques and tools used to improve the efficiency of working with web applications and improve their quality and security.

Keywords: Error monitoring, web applications, software reliability, javascript, error classification, error logging, error prioritization, real-time monitoring.

Introduction: In recent years, the development of web applications has become a key area of technology, encompassing a wide range of industries. These applications play a critical role in business, education, healthcare, and other sectors by providing access to services and information via the Internet. As software complexity and functionality increase, the likelihood of errors and defects also grows, negatively impacting system reliability, user experience and security.

The relevance of the topic lies in the need to ensure the reliable operation of web applications. Modern applications are often complex, multi-component systems interacting with various external services and databases. In such conditions, errors may occur in unexpected areas under specific operating scenarios. These errors can manifest in various forms: from syntax violations and runtime exceptions to logical errors in program execution. Timely detection of such defects is a critical task for developers and reliability engineers.

A particular challenge lies in the fact that under real operational conditions, especially under high load, web applications often experience a significant volume of errors that can reach thousands of errors per second. The frequency and severity of these errors directly depend on various factors, including the number of active users, architectural features, and the degree of integration with external systems.

The purpose of this article is to develop and analyze a methodology for the effective detection and prioritization of errors in web applications. The study focuses on exploring various approaches and tools that enable efficient and timely error identification, thereby improving the reliability and security of applications.

METHODS

In the study by Zhong H., Wang X., Mei H. [1], the author proposed a method that analyzes partial code corrections to identify bugs. This approach accelerates the error detection process without requiring a full analysis of the source code.

A similar method to improve the efficiency of bug detection was proposed in the work by Amankwah R. et al. [2], where a rapid defect detection algorithm was developed. It optimizes processing time and enhances the accuracy of static analysis, which is critical for large software systems. This algorithm identifies potential vulnerabilities and reduces the time spent on code review.

Kosińska J. et al. [3] describe approaches to observability in cloud systems, emphasizing the necessity of detailed monitoring and logging for timely error detection. The study examines tools embedded within cloud platform architectures, including distributed request tracing and microservices metrics. In a similar field, Sarika P. K. et al. [9] analyze the automation of failure diagnostics in microservices environments using Kubernetes cluster logs. The authors demonstrate how machine learning methods combined with log analysis enable failure cause identification, accelerating troubleshooting.

Camilli M., Janes A., and Russo B. [10] propose a methodology for training and validating performance models of microservices architectures through automated testing. This approach facilitates error detection during development and allows for the prediction of potential performance issues, reducing the likelihood of failures in production.

Samal U. and Kumar A. [7] present a model for assessing the reliability of software solutions, illustrating how development stages and release characteristics influence system resilience. Dhaka R., Pachauri B., and Jain A. [8] propose a two-dimensional model incorporating environmental variability and predictive analytics. This method helps evaluate failure probabilities, which is particularly relevant for web applications with dynamic architectures.

Mukwevho M. A. and Celik T. [4] examine strategies for enhancing fault tolerance in cloud environments. The study describes mechanisms such as data replication, dynamic load redistribution, and automated failure recovery. Herath J. D., Yang P., and Yan G. [11] analyze attack types targeting anomaly detection systems based on deep learning algorithms. The article discusses how adversaries bypass failure detection mechanisms, posing a threat to the stability of web applications.

Cheung G. W. et al. [12] focus on verifying the reliability and validity of models through structural equation modeling. Yagemann C. et al. [5] introduce a method utilizing symbolic state analysis to detect software errors. This approach accurately identifies root causes, which is critical for complex, multi-component systems.

Rathnayake R. M. D. S., Kumara B. T. G. S., and Ekanayake E. B. [6] apply deep learning to analyze bug reports and predict the severity of errors, enabling developers to respond promptly and prioritize issues effectively.

The reviewed studies present diverse approaches to error detection and monitoring in web applications. However, they provide limited coverage of integrating

error classification systems with real-time monitoring in high-load environments. Additionally, there is insufficient analysis of how error detection methodologies perform across different architectural patterns, particularly in distributed systems and microservices-based applications where error propagation patterns can be complex.

RESULTS AND DISCUSSION

Errors in web applications extend far beyond simple syntax violations. These applications frequently encounter issues associated with user interactions, asynchronous processes, network instability, security vulnerabilities, and failures of infrastructure components. Furthermore, errors are often contingent upon the specific execution environments, which include web browsers, operating systems, hardware specifications, or system load conditions at the time of occurrence. Such defects might remain undetected within controlled testing environments but emerge as significant problems under actual operational scenarios or during periods of elevated load. This situation imposes specific requirements on error monitoring methodologies [1].

Within the field of software development, an error is characterized as any deviation from the anticipated behavior that prevents the program from performing its intended function. Errors can manifest in various forms, ranging from syntax violations and runtime exceptions to logical errors within program execution.

To address these challenges, programming languages offer a variety of mechanisms for error handling and exception management. These mechanisms serve as the fundamental framework for error management within an application's codebase and simultaneously provide the essential components for developing comprehensive error monitoring strategies [7,8]. Figure 1 illustrates the error handling features provided by the JavaScript and TypeScript programming languages.

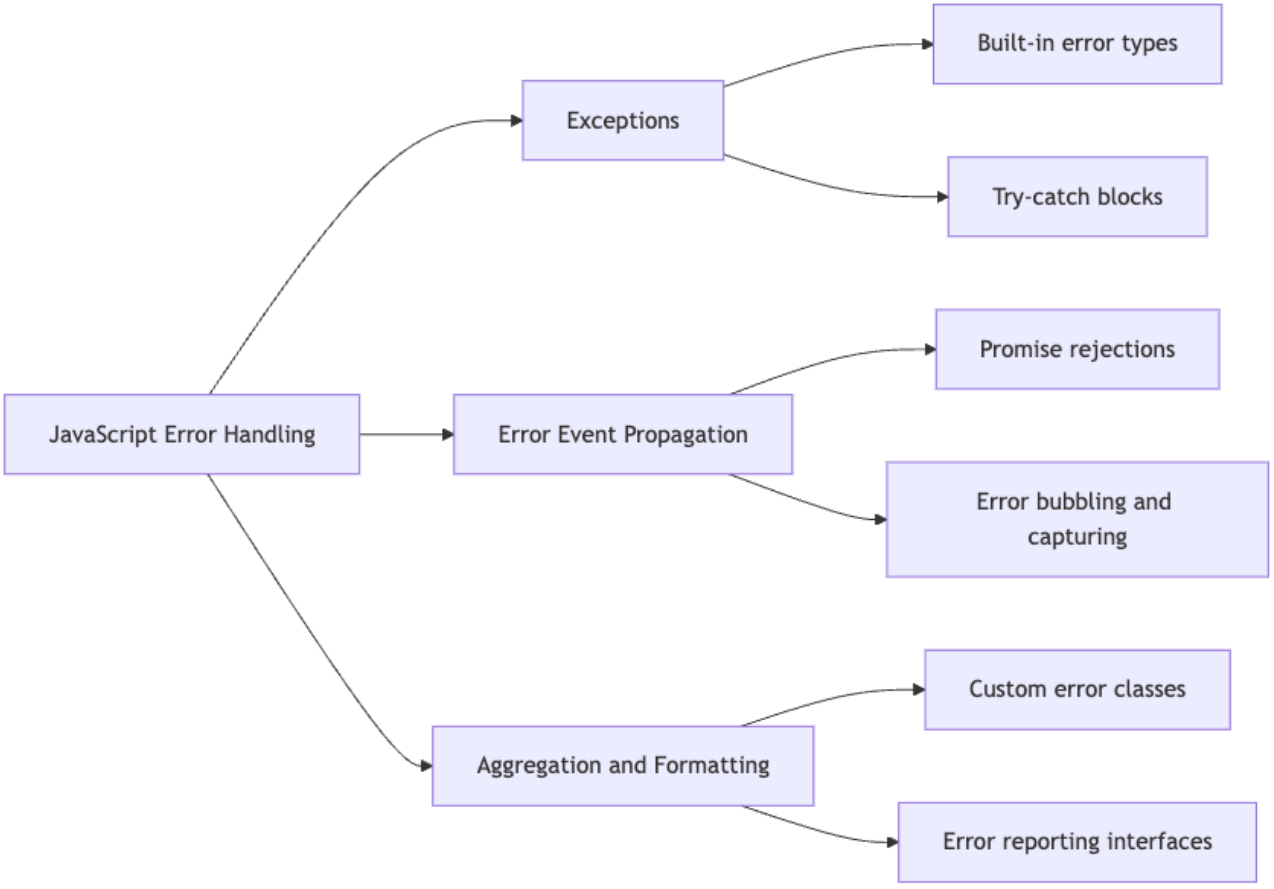

Figure 1. JavaScript Error Handling Components (compiled by the author)

While these built-in mechanisms form a solid foundation, they prove insufficient for effective error monitoring in modern web applications. Within the operational paradigm of real-world environments, especially under conditions of high load, web applications are often subjected to substantial error volumes, which may reach the order of thousands per second. The frequency and severity of these errors are

directly influenced by various factors including, but not limited to, the number of active users engaging with the application, the specific architectural framework employed, and the extent of integration with external systems.

A high error rate is, to a certain degree, an anticipated attribute of web applications due to external variables beyond the application's control. These variables encompass network reliability, the availability of infrastructure service providers, and hardware reliability issues. The distributed architecture of web applications inherently exposes them to numerous environmental influences capable of precipitating errors [3,9].

The primary challenge associated with error monitoring is not merely the detection of errors but rather the effective management and prioritization of these errors. Different types of errors have varying implications for system functionality and user experience. Attempting to investigate and respond to each individual error becomes impractical and sometimes impossible, especially when dealing with thousands of errors per second. Such an approach would consume excessive resources and potentially overlook critical issues.

Therefore, a crucial aspect of effective error monitoring is the identification and prioritization of errors that denote significant system malfunctions necessitating immediate intervention. This requires implementing sophisticated error classification and analysis methodologies that can distinguish between routine errors and those that pose significant risks to application functionality or user experience [12]. Figure 2 outlines the components required for implementing a comprehensive error monitoring strategy.

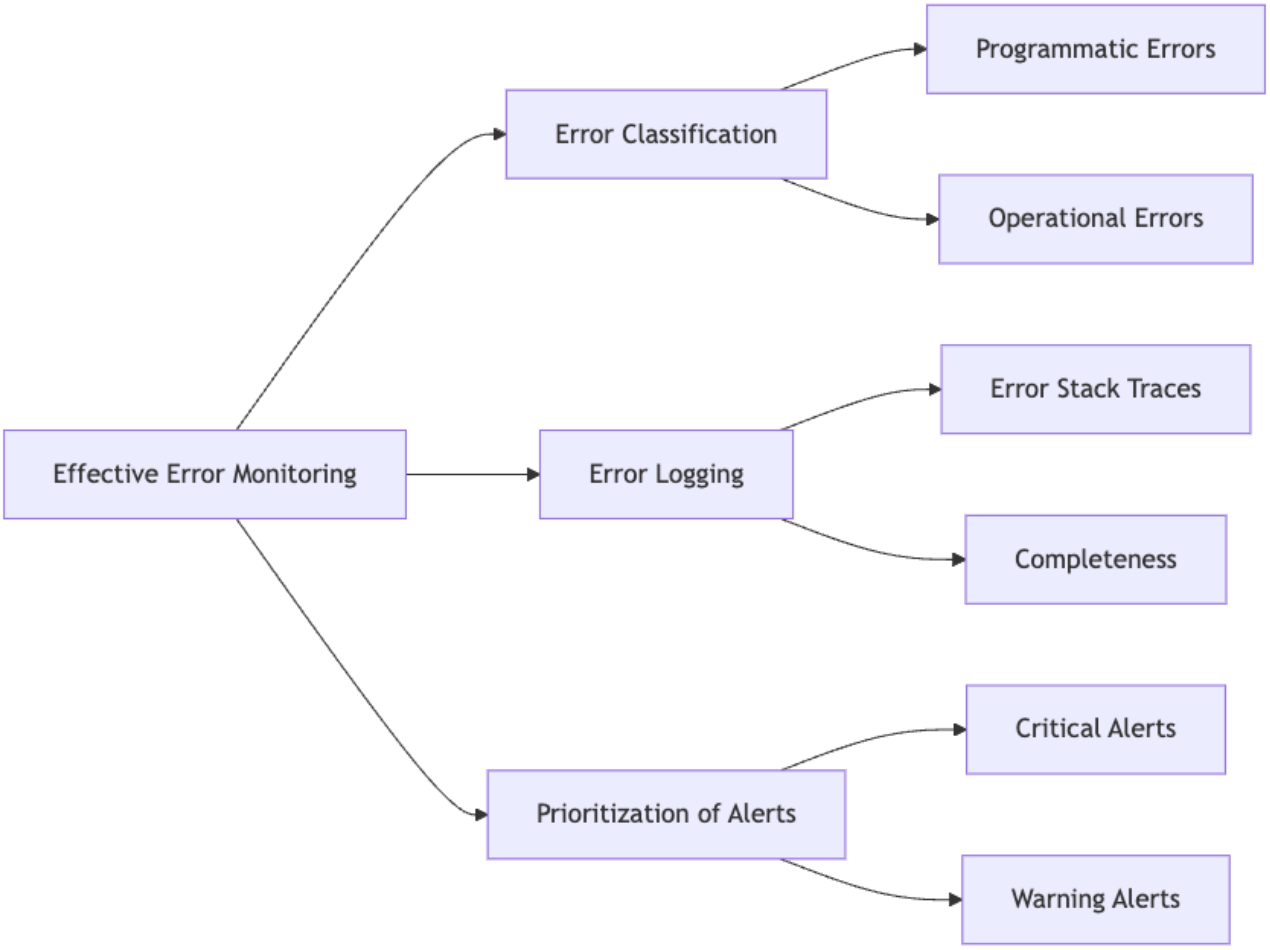

Figure 2. Components of Error Monitoring (compiled by the author)

A fundamental element of effective error monitoring involves the categorization of errors into two principal types: programmatic errors and operational errors.

Programmatic errors represent unexpected behavior within the application code itself. These errors should not occur under normal conditions and are typically indicative of faults or oversights in the coding process, necessitating immediate corrective measures. Examples of such errors include improper passing of function parameters or receiving unexpected data formats

within application components.

In contrast, operational errors originate from external factors affecting application functionality. These errors occur regularly during normal operation and are often related to infrastructure, network conditions, or user system configurations. Typical instances of operational errors encompass issues with network connectivity or incompatibility with certain web browser versions.

The importance of this classification becomes apparent when considering the practical aspects of error monitoring in high-load web applications. Operational errors generally constitute a persistent, underlying stream of issues that correlates with the level of application load. While these errors are expected and normal within certain thresholds, without proper separation and monitoring, they can obscure more serious programmatic errors [2,5].

To illustrate this classification system, consider the following JavaScript implementation:

```
// Programmatic Error Example
class ProgrammaticError extends Error {
  constructor(message) {
    super(message);
    this.name = 'ProgrammaticError';
  }
}

// Operational Error Example
class OperationalError extends Error {
  constructor(message) {
    super(message);
    this.name = 'OperationalError';
  }
}

// Error Handling Example
try {
  // Application logic
} catch (error) {
  if (error instanceof ProgrammaticError) {
    // Log and alert immediately
    console.error(error);
  } else if (error instanceof OperationalError) {
    // Log but alert with different strategy
    console.warn(error);
  }
}
```

This separation allows development teams and reliability engineers to maintain different sensitivity thresholds for different types of errors, ensuring that significant issues are not overshadowed by the routine operational failures. It provides a structured approach to error monitoring that aligns with the operational realities of modern web applications. Figure 3 illustrates the error class structure suitable for a typical web application.

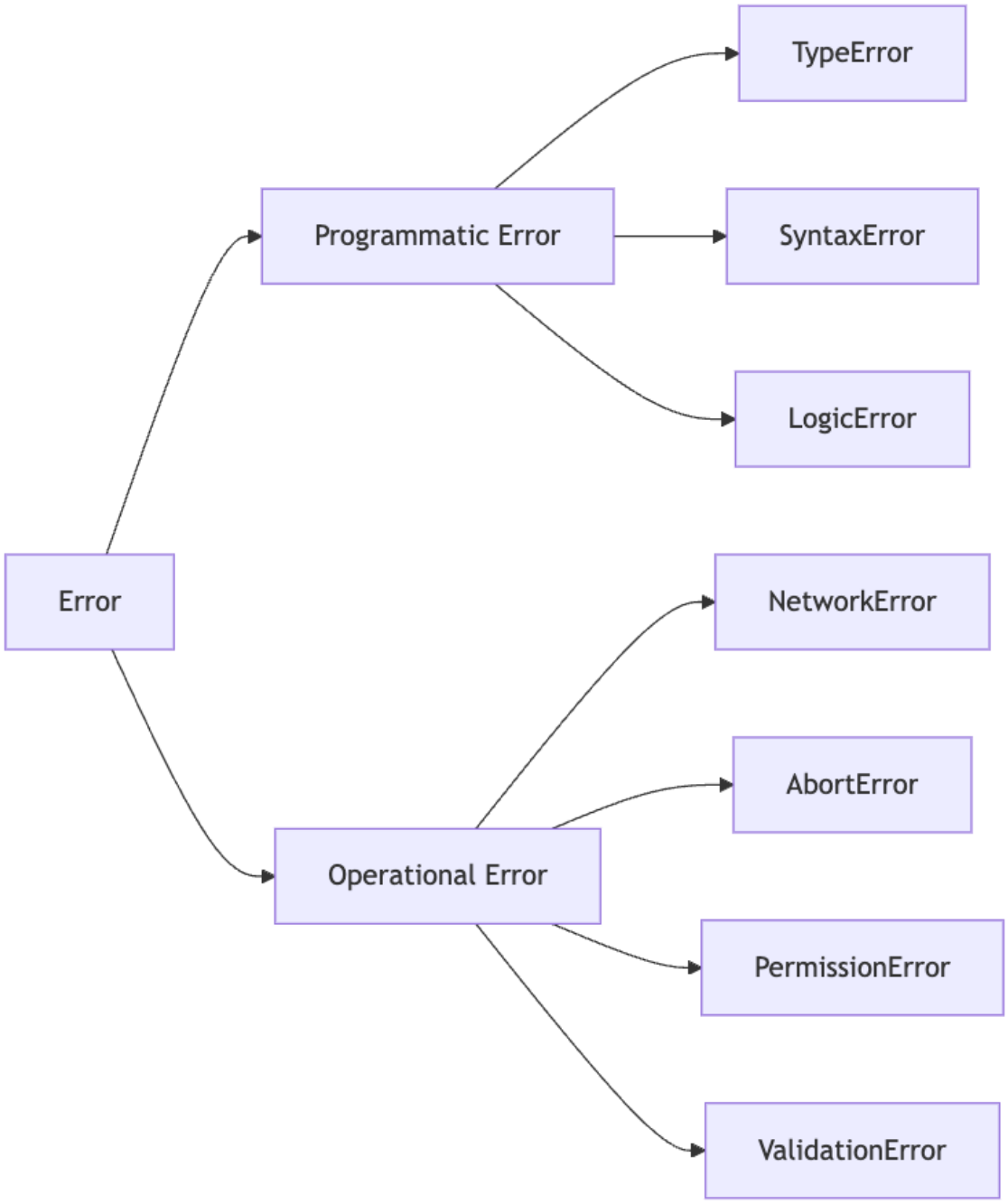

Figure 3. Example Error Class Hierarchy (compiled by the author)

Another fundamental component of proficient error monitoring is the provision of exhaustive error logging coverage. Although error classification establishes a resilient system for managing diverse error types, its effectiveness is entirely dependent on whether errors are actually being captured and logged within the monitoring system.

An example of code where errors could be silently caught and suppressed without proper logging:

```
try {
  // Application logic
} catch (error) {
  // Error is caught but not logged
  displayErrorToUser();
}
```

In the current scenario, although the error is detected, it does not yield any meaningful diagnostic information to the monitoring system. A more effective approach involves implementing centralized error logging facilitated by a specialized logging component. This approach ensures that errors are systematically recorded in a central repository, enabling comprehensive analysis and identification of underlying issues [11].

An example of code where errors are correctly handled:

```
import { errorLogger } from '#error-logger';

try {
  // Application logic
} catch (error) {
  displayErrorToUser();

  // Option 1: Log error through centralized logger
  errorLogger.logError(error);

  // Option 2: re-throw to maintain error propagation
  throw error;
}
```

Building on the foundation of these logging practices, the next crucial aspect of error monitoring involves the systematic analysis of error stream data. Two primary methodologies have proven particularly effective in this domain: threshold-based alerts and trend-based alerts [4,10].

Threshold-based alerts are activated when a monitored metric exceeds a specified threshold value. This mechanism is particularly valuable for monitoring programmatic errors because it facilitates the establishment of error budgets. These budgets play a crucial role in maintaining precise control over critical error metrics within the application, thereby ensuring operational integrity and stability.

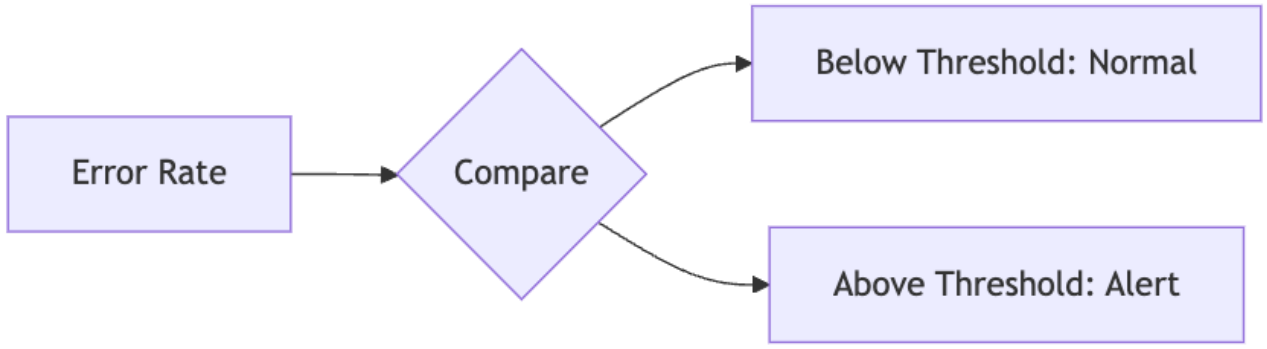

Figure 4. Threshold-Based Alert Mechanism (compiled by the author)

Trend-based alerts are activated when the value of a specified metric exhibits significant alteration over a temporal span, as compared to established historical patterns. For instance, a weekly comparison interval allows for the analysis of error quantities from the current timeframe against that from the same interval in the preceding week. This approach effectively accounts for seasonal variabilities in error rates, which may manifest differently across weekdays, weekends, and various times of the day.

This method is particularly effective for monitoring operational errors. By tailoring the sensitivity of monitoring mechanisms to accommodate both seasonal fluctuations and variations in application load, it effectively mitigates the occurrence of both false positives and false negatives.

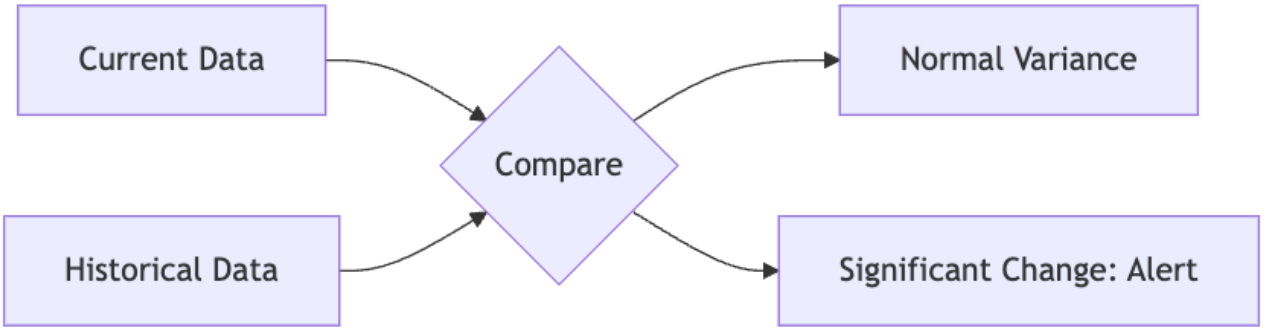

Figure 5. Trend-Based Alert Mechanism (compiled by the author)

By implementing both threshold-based and trend-based alerts, organizations can establish a comprehensive monitoring system that addresses various types of errors effectively [6]. Table 1 presents a four-tier alert system that offers a balanced approach to error detection, addressing both programmatic and operational errors.

Table 1. Four-Tier Alert System [1,3,7,9]

Alert	Example Use case
Threshold alert for new programmatic errors. Primary alert that enables quick response to new errors appearing in the system.	When a new deployment introduces a critical error, threshold alerts quickly detect the issue, enabling teams to roll back to a stable version and minimize user impact.
Trend alert for known program errors. Detects sharp increases in known errors that developers have analyzed but haven't fixed yet. Signals when issues become more critical and may need urgent attention.	When an application has known minor defects and a new deployment causes one defect's frequency to spike, the trend-based monitoring system will detect this increase even when threshold alerts don't trigger. This allows the team to quickly roll back or prioritize a fix.
User impact threshold alert. Quickly identifies mass errors affecting large numbers of users, even if all errors were previously known.	When previously identified errors affect a substantial portion of users, even without new errors or frequency changes, the system triggers an immediate alert due to the high user impact rate.
Operational errors trend alert. Identifies changes in operational error frequencies to detect mass issues quickly.	Network-related errors typically have a baseline level due to ISP issues. A significant spike in error rates across multiple users often indicates underlying problems with infrastructure or critical services. This alert efficiently identifies these abnormal patterns and enables quick emergency response.

The results demonstrate that effective error monitoring in web applications requires a multi-faceted approach combining sophisticated error classification, comprehensive logging practices, and intelligent alert mechanisms. This methodology enables development teams to maintain high software quality while efficiently managing resources through prioritized error monitoring.

CONCLUSION

The analysis presented in this paper demonstrates the critical importance of implementing advanced error monitoring systems in modern web applications. While programming languages provide basic error handling mechanisms like try-catch blocks and error objects, these standard tools are insufficient for effective error monitoring in production environments. Through the systematic classification of errors into programmatic and operational categories, organizations can effectively prioritize and manage the high rate of errors that occur in high-load environments. This classification serves as a foundation for developing targeted response strategies.

The emphasis on exhaustive error logging and centralized monitoring systems highlights the importance of maintaining detailed error records, thus facilitating in-depth analysis and continuous improvement. By implementing proper error handling practices and utilizing sophisticated monitoring tools that go beyond standard language features, organizations can significantly enhance their ability to detect, analyze, and resolve issues within web

applications. This systematic approach not only improves system reliability but also contributes to better user experience and overall application stability in production environments.

Furthermore, the proposed methodology, which integrates both threshold-based and trend-based alert mechanisms, establishes a comprehensive paradigm for error detection and management. The implementation of a four-tier alert system offers a well-rounded approach to addressing disparate error scenarios, ranging from critical programmatic errors demanding immediate intervention to the gradual progression of operational error patterns. This multi-faceted approach ensures that development teams can efficiently allocate resources while maintaining high standards of software reliability.

REFERENCES

Zhong H., Wang H., Mei H. (2020). Defining error signatures to detect real errors //IEEE Transactions on Software Engineering. 48 (2), 571-584.

Amankwa R. et al. (2020). An algorithm for rapid error detection to identify potential vulnerabilities in juliet test cases //8th IEEE 2020 International Conference on Smart City and Informatization (iSCI). pp. 89-94.

Kosińska J. et al. (2023). Toward the observability of cloud-native applications: The overview of the state-of-the-art //IEEE Access. 11, 73036-73052.

Mukwevho M. A., Celik T. (2018). Toward a smart cloud: A review of fault-tolerance methods in cloud systems //IEEE Transactions on Services Computing. 14 (2), 589-605.

Jagemann S. et al. (2021). Automated error detection using symbolic root cause analysis based on data //Proceedings of the ACM SIGSAC 2021 Conference on Computer and Communication Security, 320-336.

Ratnayake R. M. D. S., Kumara B. T. G. S., Ekanayake E. B. (2021). Predicting the severity of error messages based on CNN//2021 From Innovation to Result (FITI). 1, 1-6.

Samal U., Kumar A. (2024). A software reliability model incorporating fault removal efficiency and it's release policy //Computational Statistics. 39 (6), 3137-3155.

Dhaka R., Pachauri B., Jain A. (2022). Two-dimensional software reliability model with considering the uncertainty in operating environment and predictive analysis //Data Engineering for Smart Systems: Proceedings of SSIC 2021. – Springer Singapore. 57-69.

Sarika P. K. et al. (2023). Automating Microservices Test Failure Analysis using Kubernetes Cluster Logs //Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering. 192-195.

Camilli M., Janes A., Russo B.(2022). Automated test-based learning and verification of performance models for microservices systems //Journal of Systems and Software,187.1-10..

Herath J. D., Yang P., Yan G. (2021). Real-time evasion attacks against deep learning-based anomaly detection from distributed system logs //Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy. 29-40.

Cheung G. W. et al. (2024). Reporting reliability, convergent and discriminant validity with structural equation modeling: A review and best-practice recommendations //Asia Pacific Journal of Management. 41 (2),745-783.