**Software Architecture Document (SAD)**

**1. Introduction**

**1.1 Purpose**

This document describes the architecture of an Automated Event Log Monitoring System implemented using PowerShell. The system scans Windows Event Logs, filters critical errors, and sends alerts via email.

**1.2 Scope**

The system is designed for Windows environments, utilizing PowerShell scripting to automate event log monitoring with modular, reusable components.

**2. Software Design Principles**

**2.1 Abstraction**

* The system abstracts log retrieval, filtering, and alerting into distinct functionalities.

**2.2 Encapsulation**

* Related tasks are grouped into PowerShell functions, ensuring better maintainability.

**2.3 Modularity**

* Independent functions for retrieving logs, filtering errors, logging output, and sending alerts.

**2.4 Cohesion & Coupling**

* High cohesion is maintained within each function.
* Low coupling is ensured by minimizing dependencies between modules.

**3. System Architecture**

**3.1 Data Flow Diagram (DFD)**

(Insert a DFD illustrating the flow from log retrieval to alerting)

**3.2 Class Diagram (if applicable)**

(Insert a UML class diagram if using OOP in PowerShell or Python)

**3.3 Deployment Design**

* Install PowerShell script on a Windows machine.
* Configure email credentials for alerts.
* Schedule script execution using Windows Task Scheduler.

**4. Functional Components**

**4.1 Log Retrieval**

* Uses Get-EventLog to fetch Windows Event Logs.

**4.2 Error Filtering**

* Filters logs with severity levels like "Error" or "Critical".

**4.3 Logging**

* Saves filtered log details into a file for record-keeping.

**4.4 Alerting**

* Sends email alerts using Send-MailMessage when critical errors are found.

**5. Software Configuration Management (SCM)**

* Git repository with version tracking.
* Branching strategy for feature updates.
* Changelog maintenance.

**6. Performance and Risk Management**

**6.1 Performance Testing**

* Measure execution time using Measure-Command.
* Monitor system resource usage with Get-Process.

**6.2 Risk Identification**

* **Technical Risks**: Script failure due to missing permissions.
* **Operational Risks**: Incorrect log file parsing.

**6.3 Mitigation Strategies**

* Implement error handling using Try-Catch.
* Validate log parsing results before sending alerts.

**7. Conclusion**

This document outlines the architecture and design principles for an efficient Windows-based event log monitoring system, ensuring maintainability, scalability, and reliability.