Harish Govinda Gowda, 2022, 10:6
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology

An Open Access Journal

From Alerts to Automation: Building Self-Healing SRE Frameworks with Runbooks and Intelligent Triggers

Harish Govinda Gowda
Engineer, Cardinal Health International India

Abstract - Modern Site Reliability Engineering (SRE) is evolving from traditional alert-driven operations toward autonomous, intelligent systems capable of self-healing. This article explores the architecture, principles, and implementation of self-healing SRE frameworks using runbooks and intelligent triggers. It presents a structured approach to transitioning from reactive incident response to proactive and automated remediation by integrating observability tools, codified operational knowledge, and decision-making logic. Through real-world case studies and best practices, the article demonstrates how organizations like Netflix, Google, and Shopify are leveraging automation to reduce toil, improve mean time to resolution (MTTR), and increase service resilience. The discussion also covers toolchains, implementation challenges, and a future outlook where AI and machine learning further enhance the capabilities of self-healing infrastructure. This work serves as a comprehensive guide for SRE teams aiming to build scalable, reliable systems with minimal human intervention.

Keywords - Site Reliability Engineering, self-healing systems, automation, runbooks, intelligent triggers.

I. INTRODUCTION

The complexity of modern software systems has grown exponentially with the rise of microservices, container orchestration, cloud-native applications, and global-scale user demand. Site Reliability Engineering (SRE) emerged as a discipline to manage this complexity by applying software engineering principles to operations tasks. However, even with robust monitoring and alerting systems in place, engineers often face alert fatigue and inefficiencies caused by manual intervention in incident response processes. The gap between incident detection and resolution continues to widen due to scale, variability, and human error.

In many organizations, alerts are generated around the clock, but only a subset requires urgent attention. Many alerts end up being repetitive, non-critical, or even false positives. This overload not only slows down response time but also numbs engineering teams to truly critical issues. In turn, this increases mean time to recovery (MTTR) and jeopardizes system availability. Runbooks—documents containing pre-defined steps for resolving known issues—are commonly used to help, but they are often static and rely on human execution.

The future of operational resilience lies in self-healing systems: architectures and processes that automatically detect, diagnose, and resolve issues without human intervention. This shift from reactive to proactive and eventually autonomous recovery is enabled through a combination of codified knowledge (runbooks), sophisticated telemetry, and intelligent triggers that initiate automated actions.

In this article, we explore how SRE teams can evolve from traditional alert-response models to self-healing frameworks. We'll start by discussing the history and current challenges of incident management. Then, we'll outline the core components of a self-healing system: reliable monitoring, standardized runbooks, and intelligent automation triggers. We'll walk through how to build such a framework step-by-step, and look at real-world tools and case studies that showcase its implementation. We'll also consider the trade-offs and best practices for scaling these systems effectively and safely.

By the end of this guide, you'll understand how to begin your journey toward operational automation, and how to design self-healing infrastructure that reduces toil, improves reliability, and allows your team to focus on innovation rather than firefighting.

II. THE EVOLUTION OF INCIDENT RESPONSE

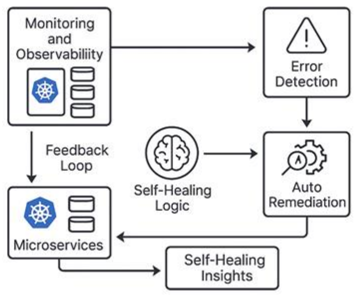
Incident response has come a long way from its early days of ad hoc troubleshooting and manual recovery. In traditional IT environments, a system failure would trigger a ticket or phone call, prompting an engineer to log in, diagnose the issue, and take corrective action—often under pressure and with incomplete information. As systems have scaled and become more complex, this model has become increasingly unsustainable.
With the rise of DevOps and SRE practices, the need for structured and efficient incident response has led to more formalized alerting systems, runbooks, and postmortems. However, even with these tools in place, many organizations find themselves overwhelmed by the sheer volume of alerts and the complexity of their systems. It's not uncommon for engineers to be woken up by alerts that ultimately don't require any action, leading to burnout and decreased focus during real incidents.

One of the early innovations in modern incident response was the use of playbooks or runbooks: step-by-step guides that help engineers diagnose and resolve known issues. These documents help standardize processes and reduce the cognitive load during high-stress situations. However, runbooks are often static, require human execution, and can become outdated if not regularly maintained.

At the same time, the proliferation of observability tools—such as Prometheus, Grafana, Datadog, and ELK—has made it easier to collect telemetry and understand system behavior. But increased observability has also led to more alerts, dashboards, and data points to monitor, compounding the challenge of determining which issues actually need intervention.

To address these challenges, forward-thinking organizations are moving toward automated incident response. This involves coupling monitoring systems with intelligent triggers that can execute predefined remediation actions without human involvement. The goal is not just to react faster, but to prevent incidents altogether through predictive analytics, anomaly detection, and automated correction mechanisms.

III. FOUNDATIONS OF A SELF-HEALING SRE FRAMEWORK



A self-healing SRE (Site Reliability Engineering) framework

Building a self-healing SRE framework requires a strong foundation rooted in observability, process codification, and automation. These foundations enable a system to detect, understand, and remediate problems without human intervention. The three main pillars of this framework are robust monitoring and alerting, codified operational knowledge through runbooks, and intelligent triggers that determine when and how to act.

The first foundational component is monitoring and alerting. Without deep visibility into the health and behavior of services, it's impossible to know when something has gone wrong. Observability tools provide the necessary telemetry—metrics, logs, traces, and events—that help detect anomalies and performance regressions. Traditional threshold-based alerts are useful for known failure modes, while modern systems increasingly incorporate anomaly detection, trend analysis, and machine learning to surface unknown issues. An effective alerting system should be actionable, relevant, and enriched with context.

Next are runbooks, which act as the brains behind automated remediation. A runbook is a structured document or script that defines how to diagnose and resolve a specific class of issues. In their most basic form, runbooks are written guides that engineers follow during incidents. However, the true power of runbooks emerges when they are automated—turned into scripts or workflows that can be executed by systems in response to certain triggers. This enables repeatable, reliable recovery actions with minimal delay or variation.

The third foundational piece is the intelligent trigger. A trigger is what decides when an automated runbook should be executed. Intelligent triggers go beyond simple conditions like "CPU > 90% for 5 minutes." They take into account historical data, service dependencies, user impact, and contextual signals to determine whether an issue is real, urgent, and automatable. These triggers can be rule-based, statistical, or powered by machine learning algorithms trained on past incident data.

Together, these elements form the groundwork for a self-healing system. Monitoring provides awareness, runbooks encode knowledge, and intelligent triggers provide decision-making. When built cohesively, this trio can reduce alert fatigue, lower mean time to resolution (MTTR), and enable proactive operations. Most importantly, they free SRE teams to focus on long-term improvements instead of firefighting repeat incidents.

Monitoring and Alerting

Monitoring and alerting form the sensory and signaling system of a self-healing infrastructure. They provide visibility into the internal state of systems and notify operators—or automated agents—when predefined conditions are met. Without effective monitoring, self-healing is impossible because the system cannot perceive what is going wrong.

Effective monitoring begins with understanding what to observe. Metrics such as latency, error rates, request volumes, and system resource utilization are core indicators of health. Logs offer granular, timestamped event data for deep inspection. Distributed tracing helps visualize how requests flow through microservices, identifying bottlenecks and errors across boundaries. When combined, these signals create a comprehensive observability stack.

Traditional alerting relies heavily on thresholds. For instance, “Alert if memory usage exceeds 90%” is a simple, useful rule—but it can lead to false positives or missed failures. Modern systems enhance this with more sophisticated methods like dynamic baselining, statistical deviation, and outlier detection. These techniques reduce noise by detecting abnormal behavior relative to historical trends, rather than static numbers.

However, good alerts are not just about signal detection—they must also be actionable. An alert should clearly describe the issue, its severity, affected components, and recommended steps (or automated responses). Alert fatigue often stems from irrelevant, ambiguous, or low-priority alerts. A key design principle is to ensure that every alert represents a condition that requires an immediate response—whether by a human or by automation.
In a self-healing system, alerts are not just notifications—they are triggers. They initiate the flow that leads to automatic diagnosis and remediation. This makes the design and quality of alerts critical. Engineers should carefully define which metrics correlate with meaningful failures and under what conditions automation should respond.

Alert routing and deduplication are also important. Grouping related alerts reduces cognitive load and helps the system identify patterns. Integration with incident management platforms (e.g., PagerDuty, Opsgenie) or automation tools (e.g., StackStorm, Rundeck) allows for seamless execution of predefined actions.

Ultimately, monitoring and alerting are not just operational tools—they are decision-making enablers for automation. When well-configured, they empower systems to act quickly, consistently, and safely in response to known problems.

Runbooks as Codified Expertise

Runbooks are the backbone of operational consistency and a key ingredient in self-healing systems. Traditionally, a runbook is a document that outlines the step-by-step process an engineer should follow to resolve a known issue. These might include restart commands, diagnostic steps, configuration checks, or remediation procedures. However, in the context of self-healing frameworks, runbooks evolve from static documents to executable workflows that systems can act upon automatically.

Codifying expertise into runbooks has several immediate benefits. First, it reduces the reliance on tribal knowledge. Engineers may come and go, but their expertise—when written down and versioned—becomes part of the organization's operational memory. Second, standardized runbooks ensure consistency in how issues are handled, eliminating guesswork and reducing the risk of human error during stressful incidents.

Runbooks come in various forms. At one end of the spectrum are simple wiki pages or knowledge base articles. These are useful for manual reference but not directly automatable. On the other end are dynamic, script-based runbooks built using tools like StackStorm, Rundeck, or custom automation engines. These can be triggered automatically in response to alerts, execute predefined logic, and even incorporate decision trees based on real-time system data.

To make runbooks automation-friendly, they must be structured, parameterized, and idempotent. For example, a runbook for restarting a failed service should include safety checks (e.g., confirming the service is indeed down), rollback mechanisms, and logging of all actions for traceability. Parameters such as service names, environments, and instance IDs should be passed dynamically, ensuring the runbook is reusable across multiple use cases.

Version control is also vital. Storing runbooks in Git or similar repositories allows for change tracking, peer reviews, and integration with CI/CD pipelines. Just like application code, infrastructure automation should follow modern software engineering practices.

Ultimately, runbooks serve as the operational intelligence of your infrastructure. When coupled with monitoring and intelligent triggers, they transform alerts into automated recovery actions. As the system evolves, these runbooks should be continuously refined based on incident outcomes, postmortems, and feedback loops—leading to more robust, adaptable self-healing capabilities over time.

Intelligent Triggers

Intelligent triggers are what differentiate a basic automation script from a truly self-healing system. While runbooks provide the "what" of remediation, intelligent triggers determine the "when" and "why." These triggers are responsible for initiating automated actions in response to system anomalies, errors, or threshold breaches—ideally without requiring human intervention.

Traditional alerting systems rely on static thresholds and binary conditions. For example, “trigger an alert if CPU usage is over 90%.” While simple to configure, these rules often lack context and can lead to noisy or irrelevant alerts. Intelligent triggers go beyond simple metrics. They incorporate contextual awareness, historical data, dependency graphs, and even business-level metrics to make smarter decisions.

For instance, an intelligent trigger might correlate high CPU usage with increased traffic, recent deployments, and error logs to determine whether the spike is expected or abnormal. If it finds supporting evidence of a problem—such as a service crash or degraded user experience—it can confidently initiate a remediation runbook. This reduces false positives and ensures that automation is only activated when truly needed.

These triggers can be implemented in various ways. Rule-based systems provide deterministic control with complex conditional logic. For example, "If service latency > 500ms AND error rate > 5% for 5 minutes, then trigger a restart runbook." More advanced systems use machine learning to identify patterns, trends, or anomalies that humans might miss. AIOps platforms like Moogsoft or BigPanda are designed to perform this kind of intelligent correlation and signal processing.

Dependency mapping also enhances trigger intelligence. By understanding which services depend on each other, a trigger can trace the origin of a failure and avoid redundant actions. For example, if a downstream service fails due to an upstream outage, the system can avoid restarting both and instead focus remediation where it's needed.

Ultimately, the goal of intelligent triggers is to minimize unnecessary noise and maximize targeted, effective automation. When implemented well, they not only reduce MTTR but also increase confidence in automated systems. By accurately interpreting system signals and launching the appropriate runbook, intelligent triggers bring self-healing infrastructure to life—ensuring systems recover gracefully without human intervention.

IV. BUILDING THE FRAMEWORK: STEP-BY-STEP

Creating a self-healing SRE framework is not a one-time implementation—it's an iterative process that evolves alongside your systems and operational maturity. While each organization may approach this differently based on its tools, culture, and architecture, the following five steps provide a structured path to designing and deploying automated recovery.

Step 1: Audit Your Current Alerting System
Start by reviewing your existing alerts. Identify which ones are noisy, repetitive, or routinely ignored. Classify alerts into categories: actionable, informational, or redundant. The goal is to reduce alert fatigue by refining thresholds, suppressing duplicates, and consolidating related alerts. Tag alerts that consistently lead to the same manual actions—these are prime candidates for automation.

Step 2: Collect and Standardize Runbooks
Next, gather the existing operational knowledge your team already uses to handle incidents. These may be wiki pages, internal documents, or even tribal knowledge passed verbally. Consolidate them and begin standardizing their format. Focus on common, high-frequency incidents first. Runbooks should include diagnostics, decision logic, commands, and rollback steps. Store them in version control for maintainability.

Step 3: Define Trigger Conditions and Logic
Once runbooks are in place, determine the conditions under which they should execute. These triggers should be based on refined alerts from your observability stack. Define thresholds, conditions, or even ML models that can detect specific failure patterns. Always validate trigger accuracy before automating remediation, to avoid acting on false positives.

Step 4: Automate Remediation Actions Safely
With triggers and runbooks defined, build automation pipelines. Use tools like StackStorm, Rundeck, or custom scripts. Start with safe, read-only automations—like diagnostics or log gathering. Then gradually introduce active remediations, such as restarting services or scaling resources. Include guardrails such as rate limits, circuit breakers, and human-in-the-loop approvals for sensitive actions.

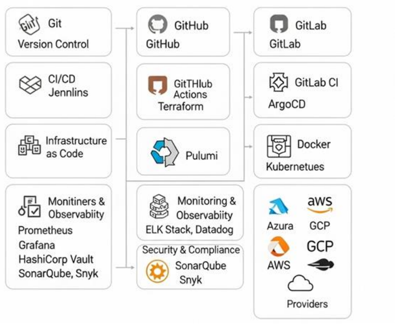
Step 5: Create a Feedback Loop
Finally, implement mechanisms to measure the effectiveness of automation. Track metrics like incidents auto-resolved, MTTR reduction, and human intervention rate. Run periodic reviews to improve runbooks and trigger logic. Postmortems should include an assessment of whether the issue

could have been self-healed, and if so, add that capability to the framework.

This iterative process transforms your incident response from reactive to proactive and eventually autonomous. Over time, you'll build a robust, intelligent system that handles common issues independently, freeing up your engineers to work on higher-value problems.

V. TOOLS AND TECHNOLOGIES

Implementing a self-healing SRE framework requires selecting the right tools to support observability, automation, and intelligent decision-making. While no single tool does it all, there is a mature ecosystem that provides powerful building blocks for each stage of the framework. These tools typically fall into three categories: monitoring/alerting, automation/runbook execution, and intelligent correlation.



A modern DevOps or cloud-native toolchain
Monitoring and Alerting

Key observability tools form the backbone of intelligent automation. Prometheus is a widely-used monitoring solution that collects time-series metrics and supports custom alert rules via Alertmanager. Grafana integrates well with Prometheus to visualize system health and define alert thresholds. Datadog, New Relic, and Dynatrace offer commercial alternatives with built-in anomaly detection, alerting, and dashboards that work across cloud-native environments.

ELK Stack (Elasticsearch, Logstash, Kibana) or OpenSearch are essential for centralized log aggregation and querying. These help trigger actions based on log patterns and textual anomalies.

Runbook Automation and Execution
For executing automated tasks, tools like StackStorm and Rundeck are popular choices. They enable defining workflows with triggers and conditions that execute shell scripts, API calls, or infrastructure commands. Ansible, Terraform, and Pulumi provide declarative automation for infrastructure-level healing—such as recreating failed VMs or updating configurations.

AWS Systems Manager Automation, Google Cloud Workflows, and Azure Automation provide cloud-native options to execute runbooks based on monitoring alerts. These tools often include secure access to infrastructure and detailed logging.

Intelligent Triggers and AIOps
For smarter, context-aware automation, AIOps platforms like Moogsoft, BigPanda, and PagerDuty Event Intelligence provide machine learning-powered correlation, noise reduction, and root cause suggestions. These tools reduce alert storms and determine when automation should be invoked based on signal analysis, historical incidents, and system dependencies.

Orchestration and Integration
To tie everything together, use platforms like Apache Airflow, Temporal, or Argo Workflows for orchestrating complex multi-step automation tasks. Event buses like Kafka or NATS allow you to build event-driven architectures where alerts can publish to a topic and automation systems can consume and act on them in real time.
The best self-healing systems are not built by choosing the fanciest tools, but by thoughtfully integrating these technologies to fit your environment. Focus on extensibility, auditability, and security, ensuring every automated action is observable, reversible, and well-documented.

VI. Case Studies and Real-World Applications

The concept of self-healing systems is no longer theoretical—it's actively being implemented by companies across industries that prioritize uptime, scalability, and operational efficiency. Let's explore a few real-world case studies that demonstrate how self-healing frameworks work in practice, what benefits they deliver, and what challenges organizations faced in adopting them.

Case Study 1: Netflix – Chaos Engineering and Auto-Remediation
Netflix is a pioneer in resilient architecture. Known for its Chaos Engineering practices, Netflix intentionally introduces failures into its systems to test and improve resilience. But beyond chaos testing, Netflix has invested heavily in self-healing. For instance, it uses a tool called Hystrix (now retired, but foundational) to implement circuit breakers around service calls. If a dependent service becomes unresponsive, Hystrix automatically short-circuits the call and returns fallback data or gracefully degrades the service.

More recently, Netflix has developed systems that detect failures in real time—such as container crashes or traffic spikes—and automatically trigger remediation. This might involve restarting a container, re-routing traffic, or auto-scaling infrastructure—all without engineer intervention. These actions are guided by service-level objectives (SLOs) and constant telemetry analysis.

Case Study 2: Google – Site Reliability Engineering at Scale
Google's SRE teams practice self-healing as a core principle. Their use of autonomous repair bots and auto-remediation pipelines allows them to maintain reliability across massive systems like Gmail, Search, and Kubernetes. One notable approach is Google's Error Budget Policy, which tightly integrates SLO monitoring with automation. When error budgets are at risk, systems can automatically rollback deployments or shift traffic between clusters.

Google also employs tools that automatically restart failed jobs, redeploy misbehaving services, or quarantine faulty instances, ensuring that issues are remediated before users are impacted. These systems are built on years of telemetry data, extensive runbook development, and internal platforms like Borg and Monarch.

Case Study 3: Shopify – Automated Incident Response
Shopify has embraced automation to handle the scale and complexity of its global commerce platform. It uses chat-based automation tools that integrate with Slack and PagerDuty. When alerts fire, automated runbooks can be launched directly from Slack using bots. These bots perform diagnostics, gather logs, and execute scripts—all tracked and logged for transparency. This speeds up triage and reduces MTTR significantly.

These examples show that self-healing systems are not only possible—they're practical. The common thread is the thoughtful combination of observability, automation, and intelligent decision-making tailored to each organization's needs and scale.

VII. BEST PRACTICES AND PITFALLS

While self-healing systems offer immense promise in reducing toil and improving uptime, they also come with a new set of challenges. Success depends on careful design, iteration, and cultural alignment. By following best practices and avoiding common pitfalls, teams can build automation that is not only effective but also safe and sustainable.

Best Practices
Start Small and Iterate Begin with low-risk, high-frequency incidents that are already well-understood. Automate diagnostics first—like gathering logs or checking service status—before moving into full remediation. Small wins help build confidence and refine your automation approach.
Ensure Observability and Auditability Every automated action should be logged and traceable. This allows engineers to understand what happened, when, and why. Observability tools should be integrated with your automation system so you can measure impact, track metrics like MTTR, and diagnose failures in the automation itself.

Add Safeguards Include safety checks, circuit breakers, and timeouts in all automated actions. For example, if a trigger initiates a service restart, the system should confirm the service is actually degraded and not already in the process of recovery. Prevent cascading failures by adding thresholds, caps, and rollback logic.

Maintain and Version Runbooks Runbooks must be treated as living documents. They should be stored in version control and updated regularly based on postmortems and incident reviews. This ensures automation remains aligned with the current system state.

Use Human-in-the-Loop for Sensitive Tasks For high-risk operations—like data deletion or production-level changes—include human approvals or validation steps. Over time, as confidence in automation grows, these steps can be relaxed or refined.

Common Pitfalls
Over-Automation Without Context Automating everything too quickly can lead to blind spots. If triggers lack proper context, automation may respond to non-critical events or even exacerbate issues. Intelligent decision-making is key.

Neglecting Postmortem Insights Many organizations automate based on assumptions. Instead, use real-world incident data to identify what can be automated. Postmortems are rich sources of improvement opportunities.

Static Runbooks for Dynamic Systems Systems change frequently. If runbooks don't evolve with them, they can become dangerous. Automation that worked a month ago may fail silently—or worse, create new issues.

Lack of Cross-Team Collaboration Self-healing is not just an SRE task. It involves developers, security, QA, and product teams. Without alignment, automation may solve symptoms without addressing root causes.
Following these best practices while being aware of pitfalls ensures your self-healing system is reliable, secure, and scalable.

VIII. CONCLUSION AND FUTURE OUTLOOK

Self-healing systems represent a critical shift in how organizations manage reliability and scale. By transitioning from reactive incident response to proactive, automated remediation, teams can achieve faster recovery, fewer outages, and a significant reduction in operational overhead. But more importantly, they gain time and space to innovate, focus on user experience, and build more resilient architectures.

The journey begins with a mindset change—moving from firefighting to engineering reliability. This includes evaluating alerts not just for their urgency but for their automability, documenting tribal knowledge into runbooks, and linking those runbooks to intelligent triggers. Over time, patterns emerge: the same failures happen in the same ways, and automation becomes the obvious path forward. Looking ahead, the future of self-healing systems is likely to become even more intelligent and adaptive. With advances in artificial intelligence, large language models, and predictive analytics, systems will increasingly anticipate failures before they occur. Instead of reacting to problems, infrastructure will evolve to self-optimize, auto-tune configurations, and perform preemptive fixes based on learned behavior.

Integration with generative AI will also change how runbooks are written and updated. Future systems might use incident data to generate new runbooks on demand or simulate responses to hypothetical failures before deploying real changes. Developers may begin to "code" reliability using higher-level intents rather than scripts or logic flows.

However, as with any powerful capability, responsibility and governance must follow. Transparency, audit trails, and human oversight will remain crucial—especially as automation begins to impact more critical parts of the business. Ensuring that automation is safe, explainable, and controllable will be just as important as making it fast and efficient.

In summary, building self-healing frameworks isn't just about writing scripts or setting up alerts. It's a strategic investment in reliability engineering. Organizations that embrace it now will not only reduce toil but will also lay the groundwork for highly autonomous, scalable operations that can meet the demands of the next decade.

REFERENCES

1. Pournaras, E., Ballandies, M.C., Acharya, D., Thapa, M.J., & Brandt, B. (2018). Prototyping Self-Managed Interdependent Networks - Self-Healing Synergies against Cascading Failures. 2018 IEEE/ACM 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 119-129.
2. Jain, L., Pothireddy, N.K., Malikireddy, S.K., Parekh, V., & Algubelli, B. (2019). January 03 , 2019 AUTOMATED FAILURE PREDICTION AND SELF-HEALING OF EDGE DEVICES IN INTERNET OF THINGS INFRASTRUCTURE.
3. Kumar, G., Muthusamy, C., & Vinaya Babu, A. (2013). Intelligent Enterprise Application Servers: A Vision for Self-Managing Performance.
4. Pigliautile, I., Castaldo, V.L., & Pisello, A.L. (2019). 7th International Building Physics Conference IBPC 2018 Proceedings: Healthy, Intelligent and Resilient Buildings and Urban Environments.
5. Samah, K.A., Hussin, B., & Basari, A.S. (2014). Modelling Conceptual Framework For Autonomous Intelligent Evacuation Wayfinding In Unfamiliar Building: Replacing Human With Intelligent Agent.
6. Frank, M., Johnstone, M.D., & Clever, G.H. (2016). Interpenetrated Cage Structures. Chemistry, 22 40, 14104-25 .
7. Giri, P., Ng, K., & Phillips, W. (2019). Wireless Sensor Network System for Landslide Monitoring and Warning. IEEE Transactions on Instrumentation and Measurement, 68, 1210-1220.
8. Alberti, S.F., Soler-Illia, G.J., & Azzaroni, O. (2015). Gated supramolecular chemistry in hybrid mesoporous silica nanoarchitectures: controlled delivery and molecular transport in response to chemical, physical and biological stimuli. Chemical communications, 51 28, 6050-75 .
9. Kumar, M., & Sharma, A. (2017). An integrated framework for software vulnerability detection, analysis and mitigation: an autonomic system. Sādhanā, 42, 1481 - 1493.
10. Brennan, R., Tai, W., O'Sullivan, D., Aslam, M.S., Rea, S., & Pesch, D. (2009). Open Framework Middleware for intelligent WSN topology adaption in smart buildings. 2009 International Conference on Ultra Modern Telecommunications & Workshops, 1-7.
11. Stephanidis, C. (2009). Intelligent and ubiquitous interaction environments.
12. Wang, J., Huang, R., Wei, Z., Xi, X., Dong, X., & Zang, S. (2019). Linker Flexibility-Dependent Cluster Transformations and Cluster-Controlled Luminescence in Isostructural Silver Cluster-Assembled Materials (SCAMs). Chemistry, 25 13, 3376-3381 .
13. Derix Bsc, C., Thum, R., Dipl, M., & Ing (2012). Self-Organising Space (SOS) : artificial neural network spaces.
14. Trivedi, M.M. (2003). Human movement capture and analysis in intelligent environments. Machine Vision and Applications, 14, 215-217.