INSTANA
an IBM Company

Achieving Software Health in the Microservices Age

Table of Contents

Introduction

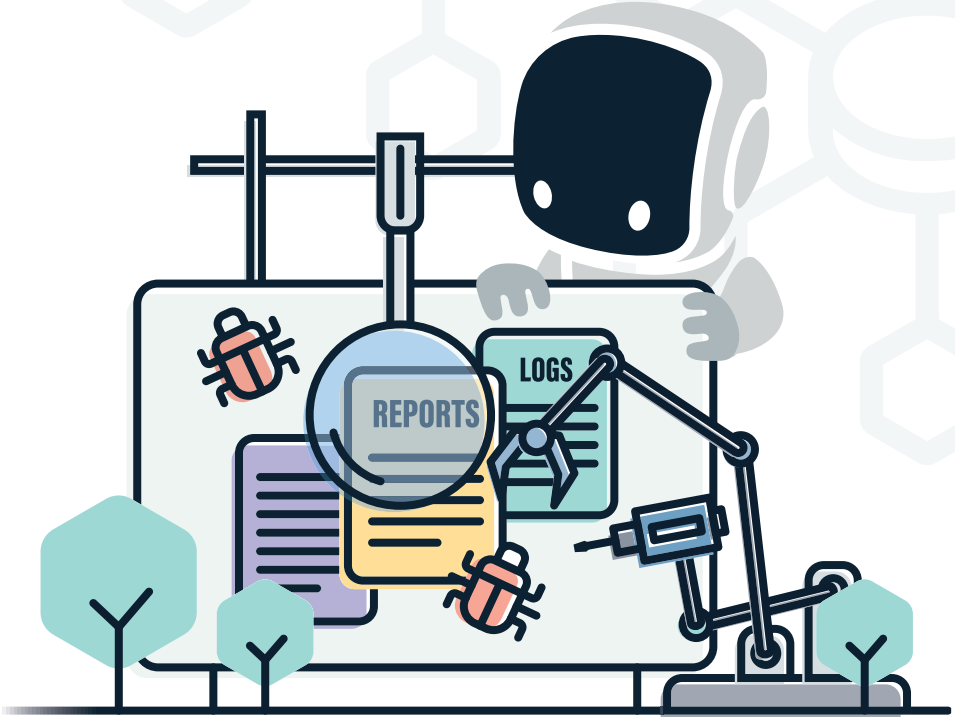
Software health maintenance is most commonly associated with software repair. If the health of any software container, service, or application degrades, the immediate focus of the enterprise shifts to repairing the software – **fast**.

Software repair has existed since the first software program was written and the code didn't work the way the programmer wanted it to do. The repair process, known as debugging, was established and is still the process of making software functionality that doesn't work correctly, work correctly.

Debugging is the process of detecting and removing existing and potential errors in software code that causes unexpected behavior. This can be done with a code profiler or other debugging tools. The nature of software repair and the tools and methods available to debug software have changed and improved over time, but the end goal is the same. Repair whatever isn't working right.

Every enterprise that creates software — whether it's applications, tools, libraries, or any other implementation — uses debugging to remediate software issues. These methods are manual and the time it takes to resolve code issues is MTTR or Mean Time to Repair. MTTR is the amount of time it takes to detect (MTTD), notify (MTTN), and initiate and validate code repairs that resolve the issue(s). It's how it has been and how it has seemed it will always be.

But there's good news on the horizon. New methods of resolving software issues are evolving rapidly. They use Artificial Intelligence/Machine Learning (AI/ML) and AIOps, wherever possible, to resolve software performance and reliability issues that were previously only resolved by manual triage. These methods will lead to better and more reliable software and applications because automated machine procedures can resolve many issues much faster than manual triage.



Software Issue Categories

There are two primary software issue categories. ‘They are:

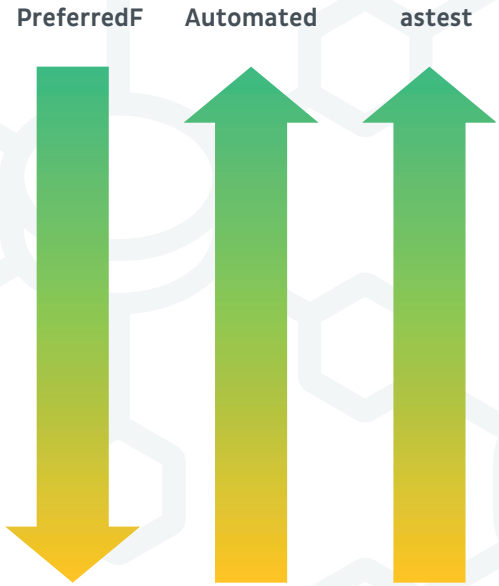
- **Operational**
- **Functional**

Operational issues occur when the application components are working properly and as expected but are caused by infrastructure issues that impact application performance. These could be from a lack of resources, such as CPU, memory, storage, or network bandwidth and most frequently during application scaling. They are the type of issues addressed by System Reliability Engineers (SREs).

Functional issues are typically application code anomalies that impact one or more application components. They can occur in one component or cascade among multiple components along the application transaction path. Functional issues always require manual triage to resolve the code issues and are usually addressed by DevOps teams and Developers.

The Software Issue Remediation Spectrum for the Microservices Age

	Issue Detection	Problem Resolution Method
A	1 second	Automated prevention
B	1 second	Automated remediation
C	1 second	Runbook semi-automated procedures
D	1 second	Manual triage



The Software Issue Remediation Spectrum describes the range of options that are emerging or are now available for software repair. It's a compilation of a range of options available for either automatically, semi-automatically, or manually repairing operational and functional software issues in production and pre-production.

The expanded use of AI and machine learning (ML) are providing new means to simplify and automated software repair, up to and including preventing certain types of software and infrastructure issues.

The most important measure for automated, semi-automated or manual issue remediation is precise real-time metrics and traces. The repair types listed in the Remediation Spectrum are driven by those measurements. The faster they are measured, the more rapidly any remediation operation can begin. They're particularly critical for any of the automated resource management or runbook procedures to avoid delays or disruptions that affect user experience.

In the Microservices Age, slow metrics and trace aggregation is contrary to successful Cloud DevOps and SRE initiatives. No DevOps or SRE team wants to say to users or their enterprise “we’re sorry, your transaction failed because our Observability platform takes too long to detect a problem.”

The foundation for automated issue remediation is an observability platform that measures and aggregates precision metrics and traces with context in real-time without compromise. It’s the only way to keep up with and vanquish issues thrown by highly distributed cloud-and scalable based microservices architectures.

To be the leading observability platform for ensuring software health requires that it must support both precise, automated Operational ***and*** Functional software issue remediation.

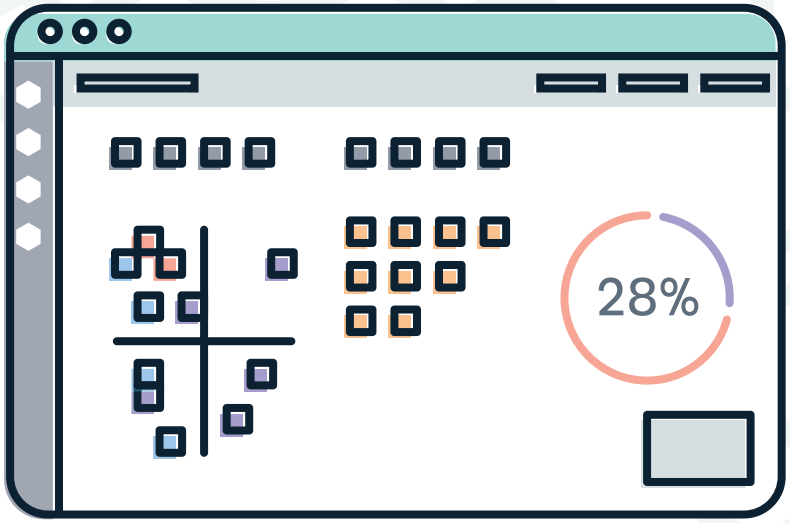
The Software Repair Paradigms

New and Emerging Remediation Methods

Observability plus AI and AIOps is the vanguard of next-gen software health technology. With AI and AIOps, the new forms of software remediation are automatic or semi-automatic prevention and repair. The decision between whether repair procedures that you use are fully automated or semi-automatic/advisory depends upon the level of trust you have that your software systems will always apply the right repair for specific issues.

Automated Incident Prevention/MTTP

Automated incident prevention is used for remediating application resource issues in the underlying application infrastructure which can be absolutely handled reliably by an observability plus application resource management (ARM) combination. The measure for determining how fast a system can act to prevent an application resource issue is Mean Time to Prevention or MTTP.



MTTP requires the fastest and most precise observability metrics to ensure real-time remediation using automated resource management. ARM uses AI/machine learning to evaluate application performance based upon baselines that you define and then acts to resolve issues that occur when those baselines aren't met. Those baselines can be performance metrics as well as cost.

In the event that a performance degradation is detected by the observability platform, the ARM platform is notified and if a threshold is exceeded, it automatically implements procedures to fix the problem. For example, if a cloud application slows down due to under-allocated CPU, memory, storage or other resources, the ARM platform increases the resources allocated to resolve the performance degradation. This usually occurs when an application scales up to respond to increased user demand.

Conversely, if the ARM platform detects that application resources are over allocated after the application scales down when the user demand decreases, it will reduce the allocated resource amount to reduce the cost of the cloud-based resources.

The fastest metrics gathering rate is currently one second, which enables ARM to engage and potentially provide an MTTP of two seconds or less. Slower observability platforms that deliver 10+ seconds or sampled metrics aren't effective precursors for providing effective MTTP.

Why is this important? Because rapid response to maintain application health has never been more critical than it is now for cloud-based microservices applications. Two seconds versus 10-12 seconds is the difference between users not noticing a problem and many dissatisfied users. The vastly distributed nature of microservices and endpoints has made maintaining the health of the application services grid more complex than ever.

To ensure that the level of service desired can be maintained, intelligent automation has become the new imperative to rapidly handle the increased issues in a highly distributed microservices environment. Fully automated MTTP drives MTTR times to zero. In an enterprise with many microservices-based applications, it would require dozens if not hundreds of skilled professionals to deal with all the issues that could arise.

That's why automated remediation is increasing in popularity. AI/Machine Learning capabilities, such as ARM, can remediate certain classes of issues, such resource management, much faster and more efficiently than humans. It also requires the fastest observability plus AI/ML to ensure minimum disruption from its managed issues.

Runbook Procedures

Runbooks are compilations of procedures and operations that are carried out to either automatically, semi-automatically, or manually resolve issues that occur. Typically, a runbook contains procedures to begin, stop, supervise, and debug a system or software. It may also describe procedures for handling special requests and contingencies. An effective runbook allows operators to manage and troubleshoot a system. Runbooks are also used to help with onboarding new or junior DevOps and SRE team members to quickly familiarize them with existing recovery and resiliency policies and procedures.

With runbook automation, these processes can be carried out in a predetermined manner. In addition to automating specific processes, such as resource management and optimization, the runbook results can be presented back to the user for further action. Multiple runbooks can also be linked together with a machine learning to provide interactive troubleshooting and guided or automated procedures.

Runbook automation is the process of defining, building, orchestrating, managing, and reporting workflows that support system and network operational processes. A runbook workflow can interact with all types of infrastructure elements, such as applications, databases, containers, endpoints, and hardware using a variety of communication methods such as command-line interfaces (CLI), HTTP REST and SOAP API's, SSH sessions, scripts, utilities, and code libraries.

The Automated Resource Management (ARM) capabilities described in the previous section are a specific Runbook use case for optimizing distributed system resources.

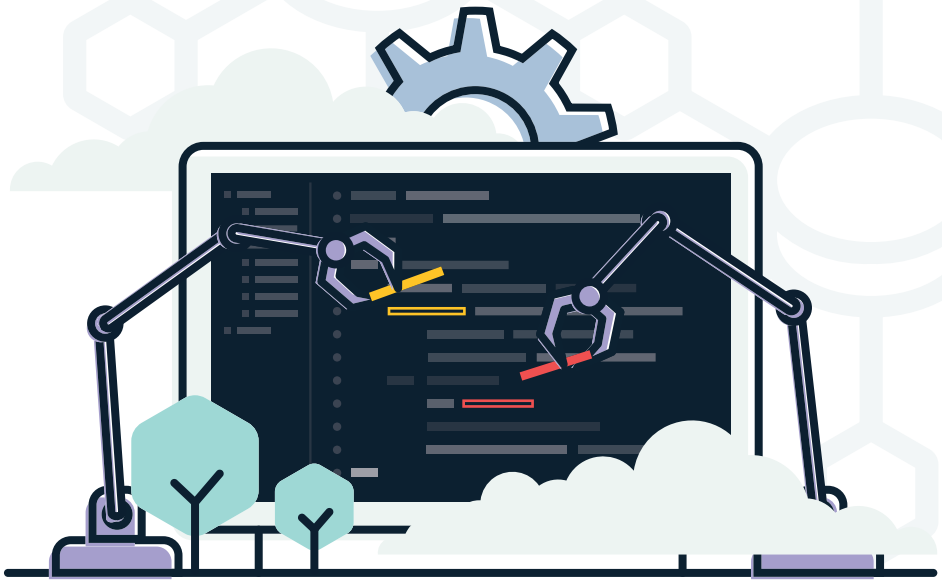
Automated ML driven Code Completion Tools

A new software repair and development capability is ML-powered code completion tools. These tools accelerate software implementation by providing automatic code recommendations based on the code and comments you input within your IDE. They're capable of generating entire functions and logical code blocks without the developer having to search for and customize sample code snippets.

There are two notable recent entrants: GitHub Copilot and Amazon CodeWhisperer. There are also a range of open-source offerings such as Asm-Dude, Atom, Captain Stack, GPT-Code-Clippy, Kite, Second Mate, and YouCompleteMe.

Amazon CodeWhisperer is an AI pair programming tool that can autocomplete entire functions based on only a comment or a few keystrokes. CodeWhisperer is based on billions of lines of publicly available open-source code and its own codebase, as well as publicly available documentation and code on public forums. Software engineers can choose between different code suggestions, autocomplete comments, and accept functions based on those comments.

GitHub Copilot, from Microsoft, uses AI to assist users by autocompleting code. Like CodeWhisperer, it's trained on billions of lines of code and turns natural language prompts into coding suggestions across dozens of languages.



Code completion tools can be applied for both testing and development initiatives to help speed the code implementation process. Code completion automation simplify provides code suggestions, but the final decisions about and integration of the suggestions are up to the practitioner, to ensure proper code function and security. In that respect, code completion is a semi-automated process.

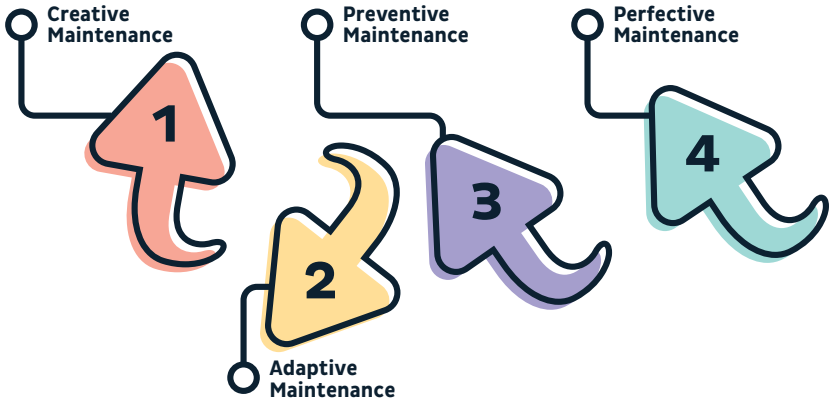
Overall, the ML-driven code completion tools make adding new or improving existing software functionality faster and, in many cases, more reliable and secure. ML-based code completion tools are evolving quickly and as indicated by the number of open-source options, many new capabilities will continue to become available to aid your software development and maintenance efforts.

The Traditional Software Issue Remediation Paradigm

Manual Repair - MTTR

Manual code repair has been the gold standard method for software repair and resiliency since the inception of programming languages. It has led to the most common software remediation term – MTTR, which stands for Mean Time to Repair.

MTTR is a maintenance metric that measures the average time required to troubleshoot and repair failed software.



Software MTTR and code triage methods are so commonplace that they've been codified as Software Lifecycle processes by the ISO and IEC. The ISO/IEC 14764 categories software maintenance into four categories:

- **Corrective maintenance:** reactive modification of a software product performed after delivery to correct discovered problems. Corrective maintenance can be automated with automatic bug fixing.
- **Adaptive maintenance:** modification of a software product performed after delivery to keep a software product usable in a changed or changing environment.
- **Perfective maintenance:** modification of a software product after delivery to improve performance or maintainability.
- **Preventive maintenance:** modification of a software product after delivery to detect and correct latent faults in the software product before they become effective faults.

Corrective Software Maintenance:

Corrective software maintenance is really any software maintenance of any kind. It addresses the errors and faults within software applications that could impact various parts the software, including the design, logic, and code. These corrections usually come from bug reports form users or customers. but corrective software maintenance can help to spot them before your customers do, which can reduce complaints. Corrective maintenance is the most well-known maintenance type and is the one most commonly referred to as debugging.

Adaptive Software Maintenance:

Adaptive software maintenance occurs when your software environment changes. This could be from changes to the operating system, hardware, software dependencies, or even cloud storage. It can also be from changes to organizational policies or rules that require updating services, making modifications to vendors, or changing payment processors.

Perfective Software Maintenance:

Perfective software maintenance is when the requirements and features of your software evolve to add new or enhanced features. As users interact with your applications, they may suggest new features that they would like as part of the software. Perfective software maintenance entails adding new features that enhance user experience and removing features that are not effective. The incremental improvement plans implemented in successive Agile sprints is an example of perfective software maintenance.

Preventive Software Maintenance:

Preventative Software Maintenance is making software changes and adaptations that it can work optimally for a longer period to reduce the Meant Time Between Failure (MTBF) and have increased resiliency. The type of maintenance is used to prevent software deterioration as you continuously adapt and change it in successive Agile sprints. These procedures can include optimizing code and updating documentation as needed and helps your software more stable, understandable, and maintainable. These changes are usually to improve software service level indicators (SLIs) as part of your overall service level objectives (SLOs).

All the manual software maintenance categories defined by the ISO/IEC 14764 definitions have a MTTR aspect and the urgency of reducing the MTTR for each is predicated by the significance and urgency of the maintenance issue. In all cases, reducing the issue MTTR is important for both software functional and operational health. MTTR procedures benefit from Observability metrics and traces that direct the remediation teams to issue root cause in real-time. In the cases where automation can be used for issue remediation, the MTTR=0.

Tips and Tricks

1) Build a solid software issue remediation plan to address all issues before they impact your users. Software maintenance and reliability is more complex in the containerized microservices era than at any other time in history, so a well-articulated plan is a necessity.

2) Use the method or methods that suit your needs. You are no longer constrained by the one-size fits all manual repair strategy, but now have access to a range of automated options that can help you repair faster or prevent issues.

3) Verify your software repairs and maintenance in pre-production before you release new or updated software into production. This helps avoid unknown surprises.

4) Use software remediation automation wherever you can and feel comfortable with to ensure peak health for your applications and infrastructure. The automated procedures that defined in this book are always optional, i.e., at the discretion of the practitioner. Automated issue response, code completion, et al will only be invoked when activated by the remediation practitioner.

The Instana Software Health Advantage

Instana provides critical observability functionality to help ensure software health and resiliency. Instana is the only observability platform that delivers one second metrics and retains them for 24 hours. These high precision and performance instantly drive industry-leading and unparalleled operational and functional issue remediation. The key Instana attributes are:

AI-Driven: AI is an integral part of the Instana Observability platform. Instana AI builds on its precise metrics and full traces to provide advanced capabilities such as Smart Alerts, trace context, Unbounded Analytics, and continuous automated discovery. It uses these intelligent features to simplify MTTR and to engage AIOps to automatically or semi-automatically remediate operational and functional software issues.

Precise One-second Metrics: Precise one second metrics provides industry leading Mean Time to Detection of one second and Mean Time to Notification of three seconds. Instana retains these metrics at that granularity for 24 hours. This ensures up-to-date measurements that enable a Mean Time to Prevention (MTTP) that never delays automated ARM and Runbook procedures.

Full End to End Transaction Traces: Instana maps every end-to-end trace without sampling for every transaction. This means no gaps caused by sampling. With that information it shows upstream and downstream dependencies in real-time to pinpoint the root cause of issues.

Automated discovery: Instana automatically discovers every application and infrastructure element the moment it is installed. Instana then instantly collects application components, nodes, containers, and architectural entity metrics and traces.

Automated context: Instana provides a context guide driven by the continuously updated Dynamic graph, which tracks all the physical components of your infrastructure, associates them, and visualizes them with their logical counterparts. It features an Upstream/Downstream button that lets you navigate to the dependencies of an Application, Service, Endpoint, or infrastructure or Kubernetes entity.

Architecture monitoring: By monitoring your architecture as well as your applications, Instana gives a better view into the impact of your applications on your architectural components and the effects of your architecture on your applications. Healthy software operation depends on healthy infrastructure, and Instana's architecture monitoring provides the precision metrics to enable automatic ARM and rapid MTTR.

Instana is the only Observability platform that support both precise, automated Operational and Functional software issue remediation in real-time.
It's **Always Accurate**[SM] interface helps ensure the fastest automated remediation and MTTR.

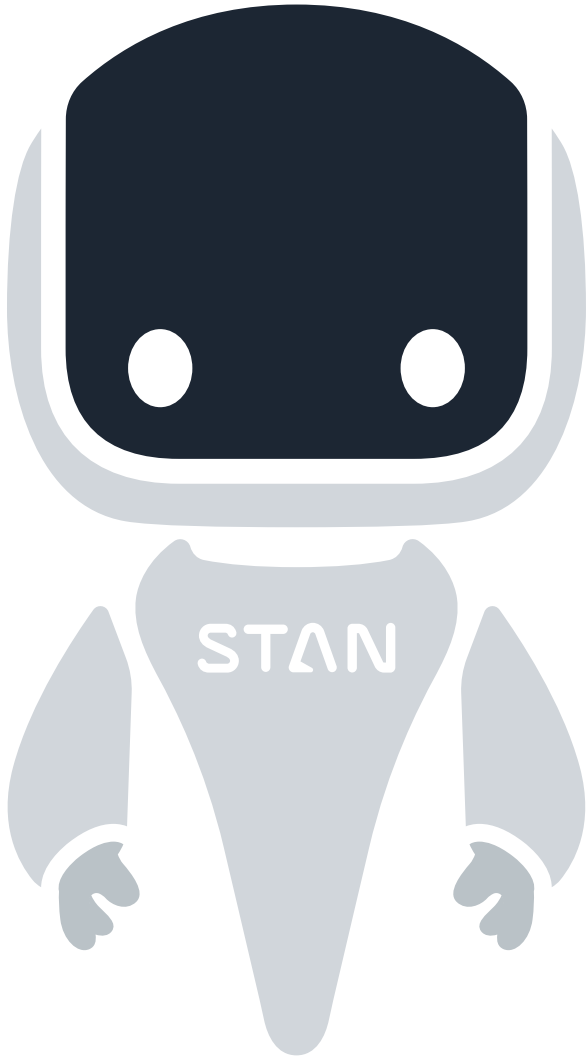
About Instana, an IBM Company

Instana, an IBM Company, provides an **Enterprise Observability Platform** with **automated application performance monitoring** capabilities to businesses operating complex, modern, cloud-native applications no matter where they reside—on premises or in public and private clouds, including mobile devices or IBM Z® mainframe computers.

Control modern hybrid applications with Instana's AI-powered discovery of deep contextual dependencies inside hybrid applications. Instana also provides visibility into development pipelines to help enable closed loop DevOps automation and helps ensure SLI/SLO compliance with precise, reliable observability information.

These capabilities provide actionable feedback needed for clients as they optimize application performance, enable innovation, and mitigate risk, helping DevOps increase efficiency and add value to software delivery pipelines while meeting their service and business level objectives.

For more information, visit **instana.com**.



Stan
Your Intelligent
DevOps and SRE

INSTANA

an IBM Company