



Revised: January 11, 2021

Service Level Agreement for AEC Solutions Inc. Products and Services

1. Agreement Overview

This Agreement represents a Service Level Agreement (“SLA” or “Agreement”) between *AEC Solutions Inc.* and [Subscriber](#) for the provisioning of IT services required to support and sustain [AEC Solutions Products](#).

- This Agreement remains valid until superseded by a revised agreement mutually endorsed by the stakeholders.
- This Agreement outlines the parameters of all IT services covered as they are mutually understood by the primary stakeholders. This Agreement does not supersede current processes and procedures unless explicitly stated herein.

2. Goals & Objectives

- The purpose of this Agreement is to ensure that the proper elements and commitments are in place to provide consistent IT service support and delivery to the Subscriber(s) by the Service Provider(s).
- The goal of this Agreement is to obtain mutual agreement for IT service provision between the Service Provider(s) and Subscriber(s).

The objectives of this Agreement are to:

- Provide clear reference to service ownership, accountability, roles and/or responsibilities.
- Present a clear, concise and measurable description of service provision to the Subscriber.
- Match perceptions of expected service provision with actual service support & delivery.

3. Stakeholders

The following Service Provider(s) and Subscriber(s) will be used as the basis of the Agreement and represent the primary stakeholders associated with this SLA:

IT Service Provider(s): [AEC Solutions Inc.](#) (“Provider”)

IT Subscriber(s): [Subscriber](#) (“Subscriber”)



Revised: January 11, 2021

4. Periodic Review

This Agreement is valid from the Effective Date outlined herein and is valid until further notice. This Agreement should be reviewed at a minimum once per fiscal year; however, in lieu of a review during any period specified, the current Agreement will remain in effect.

The Business Relationship Manager (“Document Owner”) is responsible for facilitating regular reviews of this document. Contents of this document may be amended as required, provided mutual agreement is obtained from the primary stakeholders and communicated to all affected parties. The Document Owner will incorporate all subsequent revisions and obtain mutual agreements / approvals as required.

Business Relationship Manager: [AEC Solutions Inc.](#)

Review Period: [Annually \(12 months\)](#)

Previous Review Date: [05-01-2020](#)

Next Review Date: [05-01-2021](#)

5. Service Agreement

The following detailed service parameters are the responsibility of the Service Provider in the ongoing support of this Agreement.

Service Scope

The following Services are covered by this Agreement;

- Manned telephone support
- Monitored email support
- Remote assistance using Remote Desktop and a Virtual Private Network where available
- Planned or Emergency Onsite assistance (extra costs apply)
- Daily System Health check – see Appendix A – Health Monitoring Checklist from Microsoft Azure

Subscriber Requirements

Subscriber responsibilities and/or requirements in support of this Agreement include:

- Payment for all support costs at the agreed interval.
- Reasonable availability of Subscriber representative(s) when resolving a service related incident or request.

Service Provider Requirements

Service Provider responsibilities and/or requirements in support of this Agreement include:

- Meeting response times associated with service related incidents.
- 48 hours advanced notice to Subscriber for all scheduled maintenance.
 - If Scheduled Maintenance is required, time for updates will be between 2am EST and 5am EST or on Weekends

Service Assumptions

Assumptions related to in-scope services and/or components include:

- Changes to services will be communicated and documented to all stakeholders.



Revised: January 11, 2021

6. Service Management

Effective support of in-scope services is a result of maintaining consistent service levels. The following sections provide relevant details on service availability, monitoring of in-scope services and related components.

Service Availability

Coverage parameters specific to the service(s) covered in this Agreement are as follows:

- Telephone support : 9:00 A.M. to 5:00 P.M. Monday – Friday (EST)
 - Calls received out of office hours will be forwarded to a mobile phone and best efforts will be made to answer / action the call, however there will be a backup answer phone service
- Email support: Monitored 9:00 A.M. to 5:00 P.M. Monday – Friday (EST)
 - Emails received outside of office hours will be collected, however no action can be guaranteed until the next working day
- Onsite assistance guaranteed within 72 hours during the business week

Service Requests

In support of services outlined in this Agreement, the Service Provider will respond to service related incidents and/or requests submitted by the Subscriber within the following time frames:

- 0-8 hours (during business hours) for issues classified as High priority.
 - Error when submitting Data or Photos with App
 - Error when generating Reports with Web Portal
 - Mobile Data collected using App not appearing in Web Portal
 - Payment Reporting Calculations Incorrect
 - User UI Support Onsite which results in stopping data collection or reporting
- Within 48 hours for issues classified as Medium priority.
 - Custom reporting formatting issues
 - User UI Support in addition to formal training
- Within 5 working days for issues classified as Low priority.
 - User has defined issue as a low priority and provided us a timeline

Remote assistance will be provided in-line with the above timescales dependent on the priority of the support request.

7. Contact Information

If you have any questions about this please contact AEC Solutions at support@aecolutions.ca

AEC Solutions Inc.

17 Keble Court, Richmond Hill, ON, L4E 5E5

Cell: 289-928-8676

Web: www.aecolutions.ca



8. Appendix A – Health Monitoring Checklist provided by Microsoft Azure

Health monitoring

A system is healthy if it is running and capable of processing requests. The purpose of health monitoring is to generate a snapshot of the current health of the system so that you can verify that all components of the system are functioning as expected.

Requirements for health monitoring

An operator should be alerted quickly (within a matter of seconds) if any part of the system is deemed to be unhealthy. The operator should be able to ascertain which parts of the system are functioning normally, and which parts are experiencing problems. System health can be highlighted through a traffic-light system:

- Red for unhealthy (the system has stopped)
- Yellow for partially healthy (the system is running with reduced functionality)
- Green for completely healthy

A comprehensive health-monitoring system enables an operator to drill down through the system to view the health status of subsystems and components. For example, if the overall system is depicted as partially healthy, the operator should be able to zoom in and determine which functionality is currently unavailable.

Data sources, instrumentation, and data-collection requirements

The raw data that's required to support health monitoring can be generated as a result of:

- Tracing execution of user requests. This information can be used to determine which requests have succeeded, which have failed, and how long each request takes.
- Synthetic user monitoring. This process simulates the steps performed by a user and follows a predefined series of steps. The results of each step should be captured.
- Logging exceptions, faults, and warnings. This information can be captured as a result of trace statements embedded into the application code, as well as retrieving information from the event logs of any services that the system references.
- Monitoring the health of any third-party services that the system uses. This monitoring might require retrieving and parsing health data that these services supply. This information might take a variety of formats.



Revised: January 11, 2021

- Endpoint monitoring. This mechanism is described in more detail in the "Availability monitoring" section.
- Collecting ambient performance information, such as background CPU utilization or I/O (including network) activity.

9. Analyzing health data

The primary focus of health monitoring is to quickly indicate whether the system is running. Hot analysis of the immediate data can trigger an alert if a critical component is detected as unhealthy. (It fails to respond to a consecutive series of pings, for example.) The operator can then take the appropriate corrective action.

A more advanced system might include a predictive element that performs a cold analysis over recent and current workloads. A cold analysis can spot trends and determine whether the system is likely to remain healthy or whether the system will need additional resources. This predictive element should be based on critical performance metrics, such as:

- The rate of requests directed at each service or subsystem.
- The response times of these requests.
- The volume of data flowing into and out of each service.

If the value of any metric exceeds a defined threshold, the system can raise an alert to enable an operator or autoscaling (if available) to take the preventative actions necessary to maintain system health. These actions might involve adding resources, restarting one or more services that are failing, or applying throttling to lower-priority requests.

10. Availability monitoring

A truly healthy system requires that the components and subsystems that compose the system are available. Availability monitoring is closely related to health monitoring. But whereas health monitoring provides an immediate view of the current health of the system, availability monitoring is concerned with tracking the availability of the system and its components to generate statistics about the uptime of the system.

In many systems, some components (such as a database) are configured with built-in redundancy to permit rapid failover in the event of a serious fault or loss of connectivity. Ideally, users should not be aware that such a failure has occurred. But from an availability monitoring perspective, it's necessary to gather as much information as possible about such failures to determine the cause and take corrective actions to prevent them from recurring.



The data that's required to track availability might depend on a number of lower-level factors. Many of these factors might be specific to the application, system, and environment. An effective monitoring system captures the availability data that corresponds to these low-level factors and then aggregates them to give an overall picture of the system. For example, in an e-commerce system, the business functionality that enables a customer to place orders might depend on the repository where order details are stored and the payment system that handles the monetary transactions for paying for these orders. The availability of the order-placement part of the system is therefore a function of the availability of the repository and the payment subsystem

Requirements for availability monitoring

An operator should also be able to view the historical availability of each system and subsystem, and use this information to spot any trends that might cause one or more subsystems to periodically fail. (Do services start to fail at a particular time of day that corresponds to peak processing hours?)

A monitoring solution should provide an immediate and historical view of the availability or unavailability of each subsystem. It should also be capable of quickly alerting an operator when one or more services fail or when users can't connect to services. This is a matter of not only monitoring each service, but also examining the actions that each user performs if these actions fail when they attempt to communicate with a service. To some extent, a degree of connectivity failure is normal and might be due to transient errors. But it might be useful to allow the system to raise an alert for the number of connectivity failures to a specified subsystem that occur during a specific period.

Data sources, instrumentation, and data-collection requirements

As with health monitoring, the raw data that's required to support availability monitoring can be generated as a result of synthetic user monitoring and logging any exceptions, faults, and warnings that might occur. In addition, availability data can be obtained from performing endpoint monitoring. The application can expose one or more health endpoints, each testing access to a functional area within the system. The monitoring system can ping each endpoint by following a defined schedule and collect the results (success or fail).

All timeouts, network connectivity failures, and connection retry attempts must be recorded. All data should be timestamped.



Revised: January 11, 2021

11. Performance monitoring

As the system is placed under more and more stress (by increasing the volume of users), the size of the datasets that these users access grows and the possibility of failure of one or more components becomes more likely. Frequently, component failure is preceded by a decrease in performance. If you're able to detect such a decrease, you can take proactive steps to remedy the situation.

System performance depends on a number of factors. Each factor is typically measured through key performance indicators (KPIs), such as the number of database transactions per second or the volume of network requests that are successfully serviced in a specified time frame. Some of these KPIs might be available as specific performance measures, whereas others might be derived from a combination of metrics.

Note:

Determining poor or good performance requires that you understand the level of performance at which the system should be capable of running. This requires observing the system while it's functioning under a typical load and capturing the data for each KPI over a period of time. This might involve running the system under a simulated load in a test environment and gathering the appropriate data before deploying the system to a production environment.

You should also ensure that monitoring for performance purposes does not become a burden on the system. You might be able to dynamically adjust the level of detail for the data that the performance monitoring process gathers.



Revised: January 11, 2021

Requirements for performance monitoring

To examine system performance, an operator typically needs to see information that includes:

- The response rates for user requests.
- The number of concurrent user requests.
- The volume of network traffic.
- The rates at which business transactions are being completed.
- The average processing time for requests.

It can also be helpful to provide tools that enable an operator to help spot correlations, such as:

- The number of concurrent users versus request latency times (how long it takes to start processing a request after the user has sent it).
- The number of concurrent users versus the average response time (how long it takes to complete a request after it has started processing).
- The volume of requests versus the number of processing errors.

Along with this high-level functional information, an operator should be able to obtain a detailed view of the performance for each component in the system. This data is typically provided through low-level performance counters that track information such as:

- Memory utilization.
- Number of threads.
- CPU processing time.
- Request queue length.
- Disk or network I/O rates and errors.
- Number of bytes written or read.
- Middleware indicators, such as queue length.

All visualizations should allow an operator to specify a time period. The displayed data might be a snapshot of the current situation and/or a historical view of the performance.

An operator should be able to raise an alert based on any performance measure for any specified value during any specified time interval.



Revised: January 11, 2021

Data sources, instrumentation, and data-collection requirements

You can gather high-level performance data (throughput, number of concurrent users, number of business transactions, error rates, and so on) by monitoring the progress of users' requests as they arrive and pass through the system. This involves incorporating tracing statements at key points in the application code, together with timing information. All faults, exceptions, and warnings should be captured with sufficient data for correlating them with the requests that caused them. The Internet Information Services (IIS) log is another useful source.

If possible, you should also capture performance data for any external systems that the application uses. These external systems might provide their own performance counters or other features for requesting performance data. If this is not possible, record information such as the start time and end time of each request made to an external system, together with the status (success, fail, or warning) of the operation. For example, you can use a stopwatch approach to time requests: start a timer when the request starts and then stop the timer when the request finishes.

Low-level performance data for individual components in a system might be available through features and services such as Windows performance counters and Azure Diagnostics.

Analyzing performance data

Much of the analysis work consists of aggregating performance data by user request type and/or the subsystem or service to which each request is sent. An example of a user request is adding an item to a shopping cart or performing the checkout process in an e-commerce system.

Another common requirement is summarizing performance data in selected percentiles. For example, an operator might determine the response times for 99 percent of requests, 95 percent of requests, and 70 percent of requests. There might be SLA targets or other goals set for each percentile. The ongoing results should be reported in near real time to help detect immediate issues. The results should also be aggregated over the longer time for statistical purposes.

In the case of latency issues affecting performance, an operator should be able to quickly identify the cause of the bottleneck by examining the latency of each step that each request performs. The performance data must therefore provide a means of correlating performance measures for each step to tie them to a specific request.

Depending on the visualization requirements, it might be useful to generate and store a data cube that contains views of the raw data. This data cube can allow complex ad hoc querying and analysis of the performance information.



Revised: January 11, 2021

12. Security monitoring

All commercial systems that include sensitive data must implement a security structure. The complexity of the security mechanism is usually a function of the sensitivity of the data. In a system that requires users to be authenticated, you should record:

- All sign-in attempts, whether they fail or succeed.
- All operations performed by—and the details of all resources accessed by—an authenticated user.
- When a user ends a session and signs out.

Monitoring might be able to help detect attacks on the system. For example, a large number of failed sign-in attempts might indicate a brute-force attack. An unexpected surge in requests might be the result of a distributed denial-of-service (DDoS) attack. You must be prepared to monitor all requests to all resources regardless of the source of these requests. A system that has a sign-in vulnerability might accidentally expose resources to the outside world without requiring a user to actually sign in.

Requirements for security monitoring

The most critical aspects of security monitoring should enable an operator to quickly:

- Detect attempted intrusions by an unauthenticated entity.
- Identify attempts by entities to perform operations on data for which they have not been granted access.
- Determine whether the system, or some part of the system, is under attack from outside or inside. (For example, a malicious authenticated user might be attempting to bring the system down.)

To support these requirements, an operator should be notified if:

- One account makes repeated failed sign-in attempts within a specified period.
- One authenticated account repeatedly tries to access a prohibited resource during a specified period.
- A large number of unauthenticated or unauthorized requests occur during a specified period.

The information that's provided to an operator should include the host address of the source for each request. If security violations regularly arise from a particular range of addresses, these hosts might be blocked.

A key part in maintaining the security of a system is being able to quickly detect actions that deviate from the usual pattern. Information such as the number of failed and/or successful sign-in requests can be displayed visually to help detect whether there is a



Revised: January 11, 2021

spike in activity at an unusual time. (An example of this activity is users signing in at 3:00 AM and performing a large number of operations when their working day starts at 9:00 AM). This information can also be used to help configure time-based autoscaling. For example, if an operator observes that a large number of users regularly sign in at a particular time of day, the operator can arrange to start additional authentication services to handle the volume of work, and then shut down these additional services when the peak has passed.

Data sources, instrumentation, and data-collection requirements

Security is an all-encompassing aspect of most distributed systems. The pertinent data is likely to be generated at multiple points throughout a system. You should consider adopting a Security Information and Event Management (SIEM) approach to gather the security-related information that results from events raised by the application, network equipment, servers, firewalls, antivirus software, and other intrusion-prevention elements.

Security monitoring can incorporate data from tools that are not part of your application. These tools can include utilities that identify port-scanning activities by external agencies, or network filters that detect attempts to gain unauthenticated access to your application and data.

In all cases, the gathered data must enable an administrator to determine the nature of any attack and take the appropriate countermeasures.

Analyzing security data

A feature of security monitoring is the variety of sources from which the data arises. The different formats and level of detail often require complex analysis of the captured data to tie it together into a coherent thread of information. Apart from the simplest of cases (such as detecting a large number of failed sign-ins, or repeated attempts to gain unauthorized access to critical resources), it might not be possible to perform any complex automated processing of security data. Instead, it might be preferable to write this data, timestamped but otherwise in its original form, to a secure repository to allow for expert manual analysis.



13. SLA monitoring

Many commercial systems that support paying customers make guarantees about the performance of the system in the form of SLAs. Essentially, SLAs state that the system can handle a defined volume of work within an agreed time frame and without losing critical information. SLA monitoring is concerned with ensuring that the system can meet measurable SLAs.

Note

SLA monitoring is closely related to performance monitoring. But whereas performance monitoring is concerned with ensuring that the system functions *optimally*, SLA monitoring is governed by a contractual obligation that defines what *optimally* actually means.

SLAs are often defined in terms of:

- Overall system availability. For example, an organization might guarantee that the system will be available for 99.9 percent of the time. This equates to no more than 9 hours of downtime per year, or approximately 10 minutes a week.
- Operational throughput. This aspect is often expressed as one or more high-water marks, such as guaranteeing that the system can support up to 100,000 concurrent user requests or handle 10,000 concurrent business transactions.
- Operational response time. The system might also make guarantees for the rate at which requests are processed. An example is that 99 percent of all business transactions will finish within 2 seconds, and no single transaction will take longer than 10 seconds.

Note

Some contracts for commercial systems might also include SLAs for customer support. An example is that all help-desk requests will elicit a response within five minutes, and that 99 percent of all problems will be fully addressed within 1 working day. Effective [issue tracking](#) (described later in this section) is key to meeting SLAs such as these.

Requirements for SLA monitoring

At the highest level, an operator should be able to determine at a glance whether the system is meeting the agreed SLAs or not. And if not, the operator should be able to drill down and examine the underlying factors to determine the reasons for substandard performance.



Typical high-level indicators that can be depicted visually include:

- The percentage of service uptime.
- The application throughput (measured in terms of successful transactions and/or operations per second).
- The number of successful/failing application requests.
- The number of application and system faults, exceptions, and warnings.

All of these indicators should be capable of being filtered by a specified period of time.

A cloud application will likely comprise a number of subsystems and components. An operator should be able to select a high-level indicator and see how it's composed from the health of the underlying elements. For example, if the uptime of the overall system falls below an acceptable value, an operator should be able to zoom in and determine which elements are contributing to this failure.

Note

System uptime needs to be defined carefully. In a system that uses redundancy to ensure maximum availability, individual instances of elements might fail, but the system can remain functional. System uptime as presented by health monitoring should indicate the aggregate uptime of each element and not necessarily whether the system has actually halted. Additionally, failures might be isolated. So even if a specific system is unavailable, the remainder of the system might remain available, although with decreased functionality. (In an e-commerce system, a failure in the system might prevent a customer from placing orders, but the customer might still be able to browse the product catalog.)

For alerting purposes, the system should be able to raise an event if any of the high-level indicators exceed a specified threshold. The lower-level details of the various factors that compose the high-level indicator should be available as contextual data to the alerting system.



Revised: January 11, 2021

Data sources, instrumentation, and data-collection requirements

The raw data that's required to support SLA monitoring is similar to the raw data that's required for performance monitoring, together with some aspects of health and availability monitoring. (See those sections for more details.) You can capture this data by:

- Performing endpoint monitoring.
- Logging exceptions, faults, and warnings.
- Tracing the execution of user requests.
- Monitoring the availability of any third-party services that the system uses.
- Using performance metrics and counters.

All data must be timed and timestamped.

Analyzing SLA data

The instrumentation data must be aggregated to generate a picture of the overall performance of the system. Aggregated data must also support drill-down to enable examination of the performance of the underlying subsystems. For example, you should be able to:

- Calculate the total number of user requests during a specified period and determine the success and failure rate of these requests.
- Combine the response times of user requests to generate an overall view of system response times.
- Analyze the progress of user requests to break down the overall response time of a request into the response times of the individual work items in that request.
- Determine the overall availability of the system as a percentage of uptime for any specific period.
- Analyze the percentage time availability of the individual components and services in the system. This might involve parsing logs that third-party services have generated.

Many commercial systems are required to report real performance figures against agreed SLAs for a specified period, typically a month. This information can be used to calculate credits or other forms of repayments for customers if the SLAs are not met during that period. You can calculate availability for a service by using the technique described in the section [Analyzing availability data](#).

For internal purposes, an organization might also track the number and nature of incidents that caused services to fail. Learning how to resolve these issues quickly, or eliminate them completely, will help to reduce downtime and meet SLAs.



Revised: January 11, 2021

14. Auditing

Depending on the nature of the application, there might be statutory or other legal regulations that specify requirements for auditing users' operations and recording all data access. Auditing can provide evidence that links customers to specific requests. Nonrepudiation is an important factor in many e-business systems to help maintain trust between a customer and the organization that's responsible for the application or service.

Requirements for auditing

An analyst must be able to trace the sequence of business operations that users are performing so that you can reconstruct users' actions. This might be necessary simply as a matter of record, or as part of a forensic investigation.

Audit information is highly sensitive. It will likely include data that identifies the users of the system, together with the tasks that they're performing. For this reason, audit information will most likely take the form of reports that are available only to trusted analysts rather than as an interactive system that supports drill-down of graphical operations. An analyst should be able to generate a range of reports. For example, reports might list all users' activities occurring during a specified time frame, detail the chronology of activity for a single user, or list the sequence of operations performed against one or more resources.

Data sources, instrumentation, and data-collection requirements

The primary sources of information for auditing can include:

- The security system that manages user authentication.
- Trace logs that record user activity.
- Security logs that track all identifiable and unidentifiable network requests.

The format of the audit data and the way in which it's stored might be driven by regulatory requirements. For example, it might not be possible to clean the data in any way. (It must be recorded in its original format.) Access to the repository where it's held must be protected to prevent tampering.

Analyzing audit data

An analyst must be able to access the raw data in its entirety, in its original form. Aside from the requirement to generate common audit reports, the tools for analyzing this data are likely to be specialized and kept external to the system.



15. Usage monitoring

Usage monitoring tracks how the features and components of an application are used. An operator can use the gathered data to:

- Determine which features are heavily used and determine any potential hotspots in the system. High-traffic elements might benefit from functional partitioning or even replication to spread the load more evenly. An operator can also use this information to ascertain which features are infrequently used and are possible candidates for retirement or replacement in a future version of the system.
- Obtain information about the operational events of the system under normal use. For example, in an e-commerce site, you can record the statistical information about the number of transactions and the volume of customers that are responsible for them. This information can be used for capacity planning as the number of customers grows.
- Detect (possibly indirectly) user satisfaction with the performance or functionality of the system. For example, if a large number of customers in an e-commerce system regularly abandon their shopping carts, this might be due to a problem with the checkout functionality.
- Generate billing information. A commercial application or multitenant service might charge customers for the resources that they use.
- Enforce quotas. If a user in a multitenant system exceeds their paid quota of processing time or resource usage during a specified period, their access can be limited or processing can be throttled.

Requirements for usage monitoring

To examine system usage, an operator typically needs to see information that includes:

- The number of requests that are processed by each subsystem and directed to each resource.
- The work that each user is performing.
- The volume of data storage that each user occupies.
- The resources that each user is accessing.

An operator should also be able to generate graphs. For example, a graph might display the most resource-hungry users, or the most frequently accessed resources or system features.



Revised: January 11, 2021

Data sources, instrumentation, and data-collection requirements

Usage tracking can be performed at a relatively high level. It can note the start and end times of each request and the nature of the request (read, write, and so on, depending on the resource in question). You can obtain this information by:

- Tracing user activity.
- Capturing performance counters that measure the utilization for each resource.
- Monitoring the resource consumption by each user.

For metering purposes, you also need to be able to identify which users are responsible for performing which operations, and the resources that these operations use. The gathered information should be detailed enough to enable accurate billing.

16. Issue tracking

Customers and other users might report issues if unexpected events or behavior occurs in the system. Issue tracking is concerned with managing these issues, associating them with efforts to resolve any underlying problems in the system, and informing customers of possible resolutions.

Requirements for issue tracking

Operators often perform issue tracking by using a separate system that enables them to record and report the details of problems that users report. These details can include the tasks that the user was trying to perform, symptoms of the problem, the sequence of events, and any error or warning messages that were issued.

Data sources, instrumentation, and data-collection requirements

The initial data source for issue-tracking data is the user who reported the issue in the first place. The user might be able to provide additional data such as:

- A crash dump (if the application includes a component that runs on the user's desktop).
- A screen snapshot.
- The date and time when the error occurred, together with any other environmental information such as the user's location.

This information can be used to help the debugging effort and help construct a backlog for future releases of the software.



Analyzing issue-tracking data

Different users might report the same problem. The issue-tracking system should associate common reports.

The progress of the debugging effort should be recorded against each issue report. When the problem is resolved, the customer can be informed of the solution.

If a user reports an issue that has a known solution in the issue-tracking system, the operator should be able to inform the user of the solution immediately.

17. Tracing operations and debugging software releases

When a user reports an issue, the user is often only aware of the immediate effect that it has on their operations. The user can only report the results of their own experience back to an operator who is responsible for maintaining the system. These experiences are usually just a visible symptom of one or more fundamental problems. In many cases, an analyst will need to dig through the chronology of the underlying operations to establish the root cause of the problem. This process is called *root cause analysis*.

Note:

Root cause analysis might uncover inefficiencies in the design of an application. In these situations, it might be possible to rework the affected elements and deploy them as part of a subsequent release. This process requires careful control, and the updated components should be monitored closely.

Requirements for tracing and debugging

For tracing unexpected events and other problems, it's vital that the monitoring data provides enough information to enable an analyst to trace back to the origins of these issues and reconstruct the sequence of events that occurred. This information must be sufficient to enable an analyst to diagnose the root cause of any problems. A developer can then make the necessary modifications to prevent them from recurring.

Data sources, instrumentation, and data-collection requirements

Troubleshooting can involve tracing all the methods (and their parameters) invoked as part of an operation to build up a tree that depicts the logical flow through the system when a customer makes a specific request. Exceptions and warnings that the system generates as a result of this flow need to be captured and logged.



To support debugging, the system can provide hooks that enable an operator to capture state information at crucial points in the system. Or, the system can deliver detailed step-by-step information as selected operations progress. Capturing data at this level of detail can impose an additional load on the system and should be a temporary process. An operator uses this process mainly when a highly unusual series of events occurs and is difficult to replicate, or when a new release of one or more elements into a system requires careful monitoring to ensure that the elements function as expected.