

## PROBLEM SOLVE

## How to handle root cause analysis of software defects

Root cause analysis plays a major role in how software teams can fix their defects. Here's how to employ root cause analysis and how teams can get the most out of the process.

**Stephen J. Bigelow**, Senior Technology Editor

Published: 26 Jan 2021

Enterprise software involves a complex interplay of instructions, data, associated services and dependencies. When software defects occur -- as they inevitably do -- developers must identify and understand the underlying reasons for those glitches.

Root cause analysis (RCA) for software defects is an approach developers use to better understand why a fault occurred and to take steps to drive improvements. The process is akin to how a medical team wants to diagnose and cure a patient's illness rather than simply treat the symptoms.

In a broad sense, root cause analysis is a process to identify underlying causes -- the *whys* -- of defects or failure events. Once the underlying cause is clear, a team can remediate the problem at its source. When software professionals perform the process properly, the team can use RCA results to improve product design, testing and overall quality. Let's take a closer look at some organizational benefits of root cause analysis for software defects.

### What are defects?

From a software development perspective, a defect isn't just an error message or system crash because of a coding mistake. Defects are any deviation between an actual and expected result, such as when software works perfectly, but doesn't do what the user expects.

A defect represents a departure from the expectations [outlined in a software requirements specification](#). Defects also occur in live software when pre-production tests fail to detect functional or performance problems. Here are six examples:

1. **Errors, oversights or gaps in the original software requirements.** These defects can occur when a requirement is omitted or forgotten, phrased poorly, not properly understood by stakeholders or misunderstood by developers.
2. **Errors in the design or architecture of the software.** These problems occur when software designers create an inefficient software algorithm or process, or when that algorithm or process doesn't yield the required precision in its results.
3. **Errors in the coding or implementation.** These defects include traditional bugs caused by everything from missing brackets to ungraceful error handling.
4. **Errors in the test planning or test activities.** These defects stem from inadequately tested features and functions.
5. **Errors or oversights in the deployment.** An example of these defects would be when a team provisions inadequate VM resources.
6. **Errors in the process or policies a team uses to govern the development cycle.** These defects crop up when, for example, a team obtains signoffs or approvals without adequate design, coding or testing review.

Once root cause analysis discovers the issue, the team can take proactive steps to remediate the defect and prevent it from future occurrences. If the defect resulted from design error, for example, developers can review the design and requirements documents to make corrections. If a testing mistake caused the defect, developers can update the test cases and metrics.

## Troubleshooting vs. root cause analysis

RCA and troubleshooting are different processes. Troubleshooting and general problem-solving methodologies solve specific problems. For example, if an [application's health monitoring](#) reveals that a software instance crashed and is unresponsive, the team may resolve the problem by restarting the software instance or rebooting the server.

Root cause analysis for software defects, however, might reveal that the software becomes unresponsive because of a certain error condition. Perhaps the application can't access data and the software isn't designed to handle such errors gracefully. In response, the team can release a software patch that addresses the error handling and will likely prevent the problem from recurring.

## Benefits of root cause analysis for software defects

Root cause analysis can save an organization money by helping to find and address problems earlier in the SDLC.

If a business uses RCA to find and fix problems earlier in the development cycle, the enterprise can create better quality software in a faster, more cost-effective way. Root cause analysis that prevents problems in live software also promotes customer satisfaction and protects company reputation. Some advantages of root cause analysis in software development include:

- lower software defect rates;

- improved software quality (eliminates the same defects and repetitive mistakes);
- reduced development costs;
- shortened development cycles by reducing troubleshooting fixes and remediations;
- improved user and customer satisfaction;
- [improved developer productivity](#) (allows a team to focus its effort on new features and improvements rather than fixes); and
- identification of problems elsewhere in the development and production environments.

## How to perform a root cause analysis

A team can perform RCA in a wide variety of ways, but an organized, logical and objective approach is usually considered most appropriate and effective. The analysis will typically examine log data, help desk and trouble ticket details, and other evidence from an incident. As an RCA team scrutinizes this information, its members can begin to understand a defect's underlying causes and formulate strategies and recommendations to address them.

For the purposes of this discussion, consider an RCA team to be any group that gathers to discuss or determine root causes in search of corrective actions.

**Prepare for the meeting.** RCA meetings can be held as-needed -- perhaps in the wake of an unexpected, critical fault -- or as regularly scheduled occurrences within the software development team. The RCA team leader will usually gather details and data about each fault including logs, screenshots, reporting and other resources.

RCA team members can include representatives involved in each stage of the software's lifecycle, such as requirements, design, implementation, testing, operations and anybody else involved in development. The team can also consist of individuals who worked on and fixed the initial problem. Each RCA team member reviews the details and comes to the meeting prepared to discuss the issue from their own lifecycle stage.

## Blameless reporting and recommendations

Root cause analysis for software defects only has value if a team objectively receives and implements RCA results. The biggest challenges with RCA initiatives involve the human concepts of [blame perception](#) and responsibility assignment. In other words, no one wants it on the record that a defect was their fault. Unfortunately, when an analysis points fingers, the resentment and morale loss that follows can often undermine the benefits of root cause analysis, and in turn lead to resistance from team members, managers and business leaders.

It's crucial that all RCA efforts include blameless objectivity. Reporting and recommendations should always be framed as actionable steps that don't solely place the blame on an individual or team. When reporting and recommendations are blameless, a team is more likely to receive and implement changes without resentment or resistance.

**Define the problem.** With details available, the RCA team can meet to collectively assess the defect and its effect on the software. This phase of the discussion focuses on what happened by answering a variety of common analytical questions, including:

- What is the problem?
- What events or triggers led to the issue?
- What systems or services did the issue affect?
- How long did the issue last?
- What effects did the issue have?
- Who (if anyone) was involved?

**Brainstorm the underlying causes.** After RCA team members review the evidence and clearly define the problem, they can consider the possible root cause or causes. Focus on *why* the defect happened and [brainstorm with tools](#) to identify the root cause. The RCA team leader typically moderates this part of the meeting and ensures that all members can contribute ideas.

**Select corrective action.** Once the RCA team identifies the likely root cause of a defect, it can decide on the most appropriate root cause corrective action. Here, the team should determine how to address the underlying causes. Corrective actions can vary dramatically depending on the RCA finding, such as updating requirements, enforcing coding styles and standards, making specific changes or fixes to the software, adding test cases or making changes to the deployment environment.

The team should decide if it will add to the codebase fixes already made at the software level, and if those changes require retesting. Be sure that a fix doesn't affect any other features and functionality.

**Select preventative action.** The real value of root cause analysis is ongoing improvement. Defects cost money to find and fix. By understanding the underlying cause of a problem, an RCA team's recommendations can show how to prevent similar problems in the same or other applications. The final part of an RCA process should result in explicit guidance on how to prevent similar issues from recurring. These suggestions are known as root cause preventative actions (RCPAs), and may involve a wide range of ongoing recommendations such as better documentation, more team training or skill set enhancements, process changes or IT infrastructure improvements.

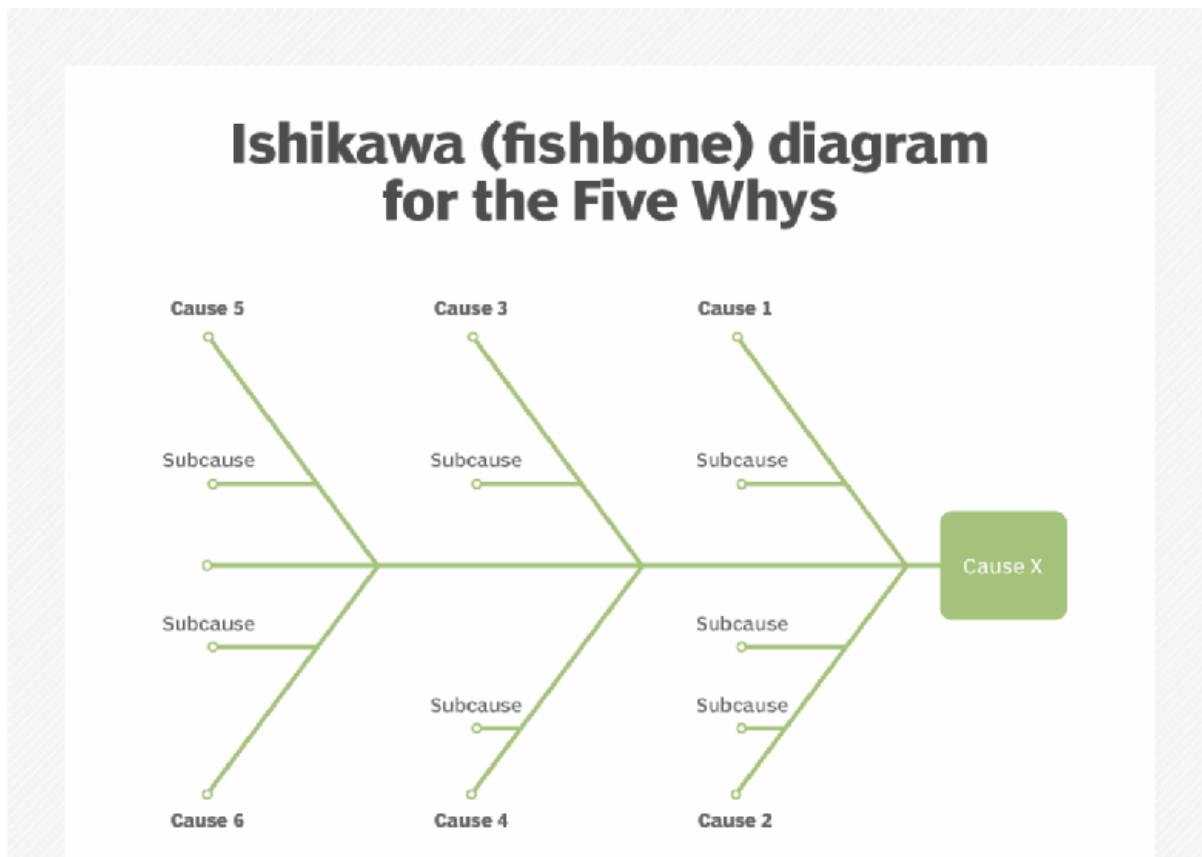
## Approaches to root cause analysis for software defects


Software teams can draw from numerous tools to approach root cause analysis tasks, including:

- [fishbone diagrams](#)
- [Five Whys](#)
- scatter plots
- Failure Mode and Effects Analysis (FMEA)
- Pareto charts

Fishbone diagrams and Five Whys are the most popular techniques.

**Fishbone analysis.** A fishbone analysis -- also called an Ishikawa diagram or cause-and-effect diagram -- is designed to help analysts visualize a root cause by sorting possible causes into categories that branch off from the original issue. The resulting diagram resembles the skeleton of a fish -- thus the name. In practice, the underlying problem or issue is written at the "head" of the fish. The visual's "bones" are categories of possible causes. Analysts then identify the primary causes under each category; if necessary, the diagram-makers can add secondary and tertiary causes.



 Example of Fishbone (or Ishikawa) diagram

**Five Whys analysis.** Asking *why* allows people to drill down into successive layers of a problem. The answer to each why becomes the basis for the next successive question. The process is similar to a child asking successive why questions -- each time the adult answers, the child uses that answer as the basis for the next question. The technique relies on brainstorming.

Five Whys analysis can be subjective since it doesn't use data or statistics, so the approach isn't suited for complex cases. The analysis may also demand more than five questions to reach a root cause, but five questions is often a starting point. Consider a simple example with just four "whys":

Problem: *The log file from a software application is missing.*

- Why is the log file missing?

- The log file is not present in the logical unit number or folder where it was anticipated.
- Why was the log file not present?
  - The log file was not enabled in the software application.
- Why was the log feature not enabled?
  - The software application was not configured properly.
- Why was the software not configured properly?
  - A team inadequately documented the application or failed to complete a process to set up and use the software. The ultimate answer may be to enable the log and provide better documentation and user training.

## Next Steps

[How to write a good software bug report](#)

[10 exploratory testing techniques for QA professionals](#)

## Dig Deeper on Software test design and planning

Six Sigma

By: Emily McLaughlin

The root cause analysis process needs all IT hands on deck

By: Tom Nolle

How AI can help with requirements analysis tools

## Why flaky tests are a problem you can't ignore

By: **Gerie Owen**

CLOUD COMPUTING

SEARCHAPPARCHITECTURE

SEARCHITOPERATIONS

JAVA

AWS



## SearchCloudComputing

### Hybrid cloud connectivity best practices and considerations

Private and public clouds stress networks in different ways and don't always play well together. Here's what to know to set up a ...

### Explore edge computing services in the cloud

Learn what you need to know about edge computing and how it compares to cloud computing. Also, discover related management and ...

[About Us](#) [Meet The Editors](#) [Contact Us](#) [Advertisers](#) [Business Partners](#) [Media Kit](#) [Corporate Site](#)

[Contributors](#) [Reprints](#) [Answers](#) [Definitions](#) [E-Products](#) [Events](#) [Features](#)

[Guides](#) [Opinions](#) [Photo Stories](#) [Quizzes](#) [Tips](#) [Tutorials](#) [Videos](#)

All Rights Reserved,  
Copyright 2006 - 2021, TechTarget

[Privacy Policy](#)

[Do Not Sell My Personal Info](#)