SauceLabs

More Data, More Problems, No More

A Software Testing Prompt Guide

Software Testing Prompt Guide

More Data, More Problems, No More

Engineering leaders and QA managers often spend hours combing through logs, dashboards, and spreadsheets to answer simple questions about software release health. This guide provides a framework for effective software testing prompts to help make the most out of your test data.

The C.E.T.F. Method

An effective prompt follows this structure for maximum clarity and results:

Context:

Are you asking about jobs, builds, or test cases?

Execution:

Do you want to trend, compare, summarize, or list?

Timeframe:

"Last 30 days," "this quarter," or "since last release."

Filters:

Specifics like OS, platform, or team name.

Putting It Into Practice

Vague prompts yield vague results. See how applying the C.E.T.F. framework transforms a simple question into an actionable insight.

FROM VAGUE TO PRECISE

Compare these two approaches to see the C.E.T.F. difference.

VAGUE PROMPT

"Show me the latest test failures."

BETTER PROMPT

"Compare pass rates between Android and iOS (Filter) for the last 90 days (Timeframe) to identify disparity in quality (Context/Goal)."

Why it works:

The better prompt specifies the Filter (Android vs iOS), the Timeframe (90 days), and the Context/Goal (identify quality disparity). This gives the AI everything it needs to deliver actionable results.

Pro Tip The more structured your prompt, the more precise and actionable the response will be.

Additional Principles

When using AI-powered testing insights, these additional principles will help you get the most useful outputs.

PROMPTING PRINCIPLES

Apply these principles for better AI-driven testing insights.

Trust the context

DON'T:
~~"Show me failures for the 'Login' test in the widget."~~

DO:
"Why are these tests failing?"

Seek patterns

DON'T:
~~"List every failed test."~~

DO:
"Group failures by error message."

Use comparison

DON'T:
~~"Is this bad?"~~

DO:
"Compare this failure rate to the last 30 days."

Drill down

DON'T:
~~"Tell me everything about the errors."~~

DO:
"Filter by 'Timeout' errors, then group by Browser."

If you'd like to drill down into optimizing Sauce AI for Insights usage, be sure to read the product-specific prompting guide found within the Sauce Labs documentation.

Engineering Directors Prompt Guide

Optimizing for Strategy and Speed

Engineering directors should not be bogged down in individual test logs. Instead, they must optimize for macro patterns, cross-team velocity, and release risk. The end goal is to direct resources to areas with the greatest impact on delivery.

What Engineering Directors Should Optimize For

Strategic visibility and delivery performance across the organization.

Tracking quality trends across teams

Performance and efficiency comparisons

Evaluating release risk proactively

Directing resources to highest-impact areas

Prompt Pattern 1

Tracking Quality Trends

Trend analysis highlights systemic risk rather than just tactical failures.

QUALITY TRENDS

These prompts help engineering directors identify macro patterns and systemic risk across teams.

Show monthly pass rate trends for the last 6 months.

Chart weekly failed job counts across all teams.

Identify sudden spikes in failed jobs this month.

Why it works:

Trend analysis surfaces systemic risk rather than individual failures, enabling strategic decision-making.

Pro Tip Tracking trends over time helps identify degradation patterns before they impact releases.

Prompt Pattern 2

Performance & Efficiency Comparisons

Cross-team and cross-platform comparisons reveal disparity in execution effectiveness and help identify where investment is needed.

PERFORMANCE COMPARISONS

Compare execution effectiveness across platforms and teams to identify investment opportunities.

Show failure rates for Chrome vs Safari.

Show the average job duration trends over the last quarter.

Compare the runtime changes between this month and last month.

Why it works:

Cross-platform comparisons reveal where execution effectiveness diverges and where investment is needed.

Key Insight Cross-team and cross-platform comparisons reveal disparity in execution effectiveness.

Prompt Pattern 3

Evaluating Release Risk

Catching declining quality before a release prevents costly rollbacks. These prompts help anticipate instability before it reaches production.

RELEASE RISK

Catch declining quality before a release to prevent costly rollbacks.

List builds with failure rates above 20% in the last 30 days.

Show builds with declining pass rates over time.

Why it works:

These prompts help anticipate instability before it reaches production, preventing costly rollbacks.

Critical Catching declining quality before a release prevents costly rollbacks and production incidents.

QA Managers Prompt Guide

Operational Clarity and Daily Test Health

While engineering directors focus on longer time horizons, QA managers are responsible for day-to-day efforts. They need to focus on reducing flakiness, monitoring coverage gaps, and shortening feedback loops by identifying any regression and reliability issues as quickly as possible.

What QA Managers Should Optimize For

Move from "What failed?" to "Why is this happening?"

- **Monitoring stability over time**
- **Identifying recurring failures and flakiness**
- **Investigating root causes efficiently**
- **Shortening feedback loops**

Prompt Pattern 1

Stability Monitoring

Surface instability before it becomes a release blocker by tracking daily execution patterns.

These prompts surface instability before it becomes a release blocker.

Show the daily pass/fail trend over the last 30 days.

Compare this week's pass rate to last week's.

Chart flaky test cases detected in the last 60 days.

Why it works:

Tracking daily execution patterns helps surface instability before it escalates into a release blocker.

Pro Tip Tracking trends over time helps identify degradation patterns before they impact releases.

Prompt Pattern 2

Flakiness & Persistent Failures

A flaky test is worse than no test -- it erodes confidence. Managers must isolate recurring issues from one-off noise.

FLAKINESS & PERSISTENT FAILURES

Isolate recurring issues from one-off noise to focus on systemic problems.

Which test cases failed more than 3 times in the last 14 days?

Compare flaky vs stable test case counts by platform.

List consistently failing iOS test cases from the past month.

Why it works:

Isolating recurring failures from one-off noise lets you focus remediation efforts on systemic problems.

Remember A flaky test is worse than no test -- it erodes confidence in the entire test suite.

Prompt Pattern 3

Root Cause Investigation

Use specific identifiers to move from detection to diagnosis instantly.

ROOT CAUSE INVESTIGATION

Always include job IDs or build names for effective failure analysis.

Why did job [Job ID] fail?

Analyze differences between passing and failing runs of build [Build Name].

What changed between successful and failed runs?

Why it works:

Specific identifiers enable the AI to compare configurations, timelines, and logs for faster root cause analysis.

Critical Specific identifiers enable faster, more accurate root cause analysis.

Best Practices

Prompting Checklist

Before submitting a query, confirm:

Define specific timeframes

Align queries to your sprint cycles or quarterly reviews.

Clearly identify the entity

Specify whether you're querying jobs, builds, test cases, or other entities.

Filter context

Narrow the scope to specific teams or environments to make insights actionable.

Ask for trends and comparisons

Use keywords such as "trends," "comparisons," or "over time" for richer insights.

Visualize

Ask for charts and graphs (e.g., "Chart weekly job execution...") to communicate trends to stakeholders effectively.

Summary

Converting Insights to Action

By mastering these prompt patterns, engineering organizations can shift from reactive debugging to proactive quality management.

The True Value of AI in Testing

The true value of AI in testing is its ability to facilitate a shift from reactive troubleshooting to proactive quality management. Like any AI product, the quality of the insight depends entirely on the prompt.

Proactive quality management

Shift from reactive debugging to proactive quality management

Faster delivery

Enable faster delivery with greater confidence in release health

Surface macro patterns

Quickly surface macro patterns, diagnose root causes, and anticipate release risks

Build release confidence

Ship with certainty backed by reliable quality metrics and insights

Start applying these patterns to your daily workflows and track the improvement in your team's efficiency. The more you prompt, the better the AI will get at context, details, and ultimately value.