

# The Blast Radius Test

Trusted BI. Prove every number you publish.

Run the checklist against your own environment, not against a vendor's feature list. Work through it with the two or three people who would actually have to answer, and time the answers where noted. The time it takes to answer is usually the finding.

Score at the end of each one. The score tells you whether the problem is a workflow annoyance or an architecture gap, and the two need different budgets.

A column gets renamed. A table gets restructured. The damage surfaces in production, when a dashboard breaks or a report goes to a customer with the wrong number in it.

Then root cause analysis begins, which means tracing by hand through logs, tribal knowledge, and whoever is awake. The analysis that should run before the change runs after the incident instead.

## Section A: Recall your last incident

- ☐ Date and one-line description of your most recent data incident: [FILL IN]
- ☐ How was it discovered? Monitoring / a user complained / an executive complained / other: [FILL IN]
- ☐ Time from the change deploying to the incident being noticed: [FILL IN]
- ☐ Time from noticing to identifying root cause: [FILL IN]
- ☐ Number of people pulled into the investigation: [FILL IN]
- ☐ Was a pre-change impact analysis run? Yes / No / There's no way to run one

## Section B: Test it live

*Pick a real column in a real production table.*

- ☐ Time to list every downstream consumer of that column, across every system: [FILL IN]
- ☐ Did the answer include systems whose jobs haven't run in the last 30 days? Yes / No / Unknown
- ☐ Did the answer include reports and dashboards, or only pipelines? [FILL IN]
- ☐ How confident is the team that the list is complete, 1 to 5? [FILL IN]

## Section C: Requirements to test on any vendor call

- ☐ Is lineage computed by parsing the actual code, or inferred from observing queries that happened to run? Ask which, and ask what happens to coverage for a job that runs quarterly.
- ☐ Does impact analysis reach column level across systems, before the change deploys?
- ☐ When something breaks, can the same engine trace the wrong number back to the line of code that produced it?
- ☐ Ask the vendor to trace one of your report fields back to source, live, during the first demo.
- ☐ Does the lineage carry version history, so you can see what the dependency graph looked like on the day of the incident?

## Score

1 point each for: discovery by user or executive complaint rather than monitoring, over 2 hours to root cause, four or more people pulled in, no pre-change impact analysis available, a confidence score of 3 or lower on the completeness question.

**0 to 1.** **Your dependency map is real** and your team trusts it.

**2 to 3.** **You're finding out about the blast radius after the blast.** Each incident is survivable and the pattern is expensive.

**4 to 5.** **Changes ship blind.** A definition you can't trace is a definition you can't trust. Right now the tracing runs after the ticket lands.

## Next step

Know the blast radius before the first ticket lands. Take the column you tested in Section B into a vendor demo and ask for the complete downstream list, live, in the session.

### Get in touch.

Request a demo to learn more about MetaKarta and how it helps you with impact and root cause analysis.

[Request Demo](#)