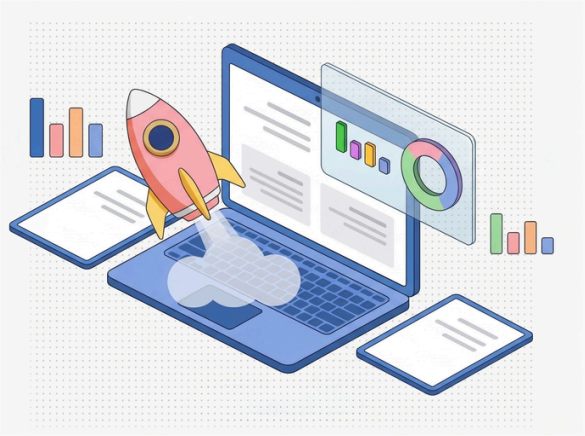
Why bugs still reach production when your dashboard is green [Cheat Sheet]



You want to ship on time. You don't want to explain to management why the release is delayed again, or why customers found bugs that should have been caught.

This cheat sheet gives you a testing strategy that works regardless of your setup, manual, automated, or mixed. Same approach, any tool. Focus your effort on the tests that actually matter, drop the ones that don't, and catch critical bugs before they reach production.

1 Why bugs slip through

You're measuring the wrong thing

A typical test report tells you: 800 passed, 50 failed, 150 not executed. Out of 1,000 test cases.

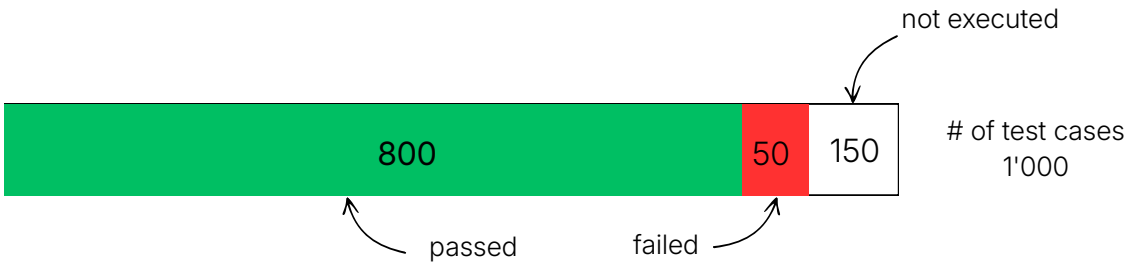


Figure 1: Software testing report today

That report says nothing about the quality of your application. It tells you how many tests ran. Not whether you tested the right things.

This is why bugs show up after release even when the report looked green. The numbers gave you confidence, but it was false confidence. You had data, just not the data that matters.

The metric you actually need is business risk coverage. We'll get to how it works in a second.

© progile AG. All Rights Reserved.

Testing is the bottleneck, not the safety net

Testing is the last step before release. And in most companies, it's not fast enough. You'll recognize one of these situations:

Tests are automated, but building and maintaining them takes so long that results come too late in the cycle. Or your team gave up on automation and tests manually, either partially or stretched over weeks. Or automation was never in place to begin with.

In all three cases, testing slows you down instead of protecting you. Two things fix this: a smarter test strategy (next section) and the right tooling (we'll cover that too).

2 Business risk coverage

Stop counting test cases. Start measuring risk

Most companies track test coverage: how many functions did we test out of all functions? The problem with that metric is simple. It treats every function as equally important. A login page and a buried settings toggle get the same weight. That makes no sense.

Business risk coverage fixes this by weighing each function according to its actual risk: how often does it get used, and how bad is the damage if it breaks?

How to calculate risk weight

Risk = how frequently something is used x how much damage a failure causes.

You probably don't have exact numbers for these two dimensions. That's fine. You don't need them. Assign each function a class from 1 (lowest) to 5 (highest) for both frequency and damage. Then look up the weight in this table:

	Damage 1	Damage 2	Damage 3	Damage 4	Damage 5
Freq 1	4	8	16	32	64
Freq 2	8	16	32	64	128
Freq 3	16	32	64	128	256
Freq 4	32	64	128	256	512
Freq 5	64	128	256	512	1024

Risk weight based on frequency and damage class

© progile AG. All Rights Reserved.

A function with Frequency 4 and Damage 4 gets a weight of 256. A function with Frequency 1 and Damage 1 gets a weight of 4. That's a 64x difference. This is why treating all functions equally is misleading.

A quick example
Take a web shop with two main areas: Customer Tasks and Handle Products.

Name	Frequency class	Damage class	Weight	Relative Weight (%)	Contribiution (%)
Example Web Shop					
Customer Tasks	**4**	**4**	**256**	**10.81**	**10.81**
Register	3	4	128	18.18	1.97
Login	5	4	512	72.73	7.86
Modify Customer Data	3	2	32	4.55	0.49
Check Orders	2	3	32	4.55	0.49
Handle Products	**3**	**3**	**64**	**2.70**	**2.70**
Product Configuration	3	3	64	32.65	0.88
Modify Products View	4	2	64	32.65	0.88
Compare Products	1	1	4	2.04	0.06
Search for Products	4	2	64	32.65	0.88

Login (Frequency 5, Damage 4) carries 72.73% of the relative weight within Customer Tasks. Compare Products (Frequency 1, Damage 1) carries 2.04%. If you've tested Compare Products but not Login, your test coverage looks decent. Your risk coverage tells a completely different story.

Your business risk coverage is the sum of the relative weights of all tested functions. That's the number that tells you whether you're ready to ship.

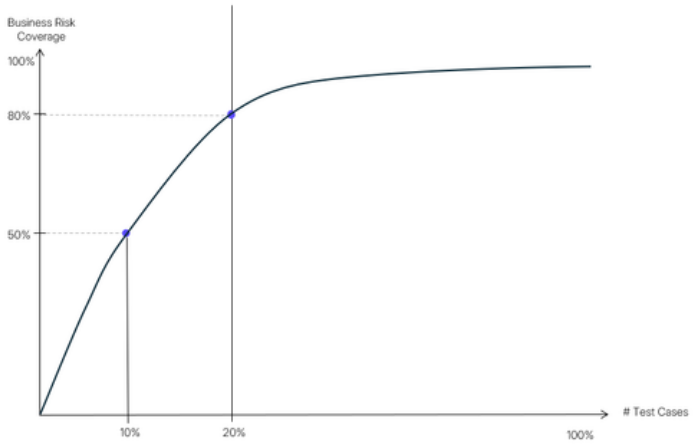
Practical tip: Run this assessment with your business analysts or testers. It takes about 1.5 days. Use silent approval (if nobody objects to a rating, it stands) so people give honest answers instead of playing it safe.

© progile AG. All Rights Reserved.

3 The 80:20 rule

20% of your test cases cover 80% of your risk

Once you know the risk weight of each function, something becomes obvious: a small number of test cases carries most of the risk coverage. The Pareto principle applies almost perfectly here.



With the top 20% of your test cases, you typically cover 80% of your business risk. The other 80% of test cases? They don't add much coverage.

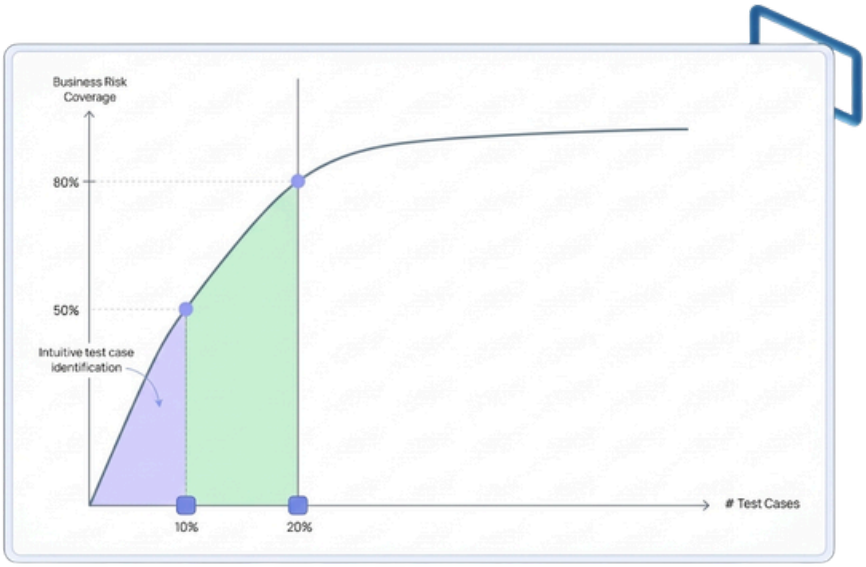
This means you can stop burning your team's time on hundreds of test cases that barely move the needle. Focus on the ones that actually reduce risk. Fewer tests, better results, less maintenance.

© progile AG. All Rights Reserved.

4 Methodology

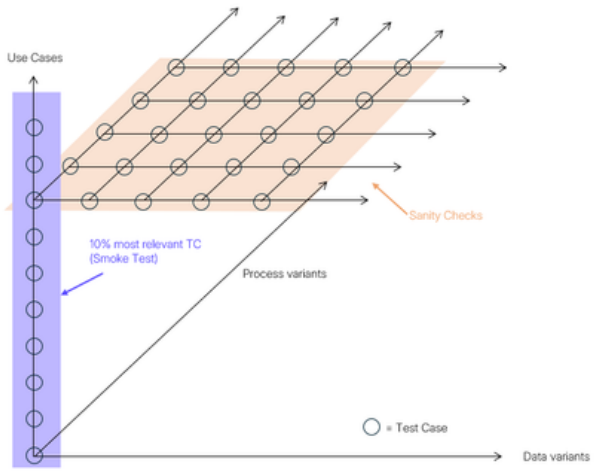
Your team already knows the top 10%. For the rest, they need a method
Most testers can pick the 10% most important test cases without any formal training. They know the application, they know what breaks. These are your smoke tests.

But after that first 10%, it gets tricky. Without a method, teams tend to create test cases that overlap with what they already have instead of closing actual gaps. You end up with more tests but not more coverage.



The 3 dimensions of software testing

Every function can be tested along three dimensions: use cases (the happy path), process variants (what happens when users deviate from the happy path), and data variants (different input data).



© progile AG. All Rights Reserved.

Your smoke tests cover the use case dimension. Your sanity checks add the process and data variants for your highest-risk functions. Together, that's how you get from 50% to 80%+ risk coverage with roughly 20% of the total possible test cases.

5 Who owns test automation?

Developers can't. A dedicated team is expensive. So who does it?
Developers need to build features. They don't have time to automate tests on top of that. A dedicated test automation team sounds good until you realize they spend 25-30% of their time just keeping existing tests alive because every release breaks something.

Going fully manual isn't an option either. It's too slow, and it doesn't scale as your application grows.

Your domain experts
Your domain experts already spend up to 40% of their time testing manually or explaining requirements to automation engineers. They know the application better than anyone. If they can automate tests directly, you skip the handover, cut testing effort in half, and free up developers.

The obvious objection: they're not technical enough. And the no-code tools you've seen can't handle complex enterprise applications.

That used to be true. With today's computing power and visual recognition technology, there are tools that actually deliver on this promise. Tools where a business user automates a process once and the system keeps running it, even when the UI changes. No coding, no selectors, no maintenance overhead every time something moves two pixels.

If you've been burned by this promise before, that's fair. But the technology has caught up. It's worth a fresh look.

© progile AG. All Rights Reserved.

6 Putting it together

Every piece cuts cost. Combined, they change how you release.
Run a rapid risk assessment to know which functions matter most. Apply the 80:20 rule to focus your effort where it counts. Use the three dimensions to design test cases that close real gaps instead of adding noise. And give the people who know your application best the tools to automate their own tests.

Companies that combine these strategies don't just test more efficiently. They ship with confidence because they know exactly how much of their business risk is covered before every release.

We run a strategy workshop covering all of this for new customers. If you want to see how this works in practice, including the tooling side, book a call. If it's not a fit, no hard feelings, and we won't follow up.

Get a demo including a short strategy session.

© progile AG. All Rights Reserved.