Software Releases Without Last-Minute Surprises [Cheat Sheet]

You want to ensure you don't have to explain to management why the release is delayed again, or how the defects customers are experiencing could have slipped through.

This cheat sheet provides IT leaders with a strategy to decide with certainty whether you're ready to ship while meeting your timeline and maintaining agile speed.

Let's transform your software quality from a roadblock into your competitive advantage.

Metrics and business risk

1 Why last-minute surprises occure

Flying blind without knowing it

Today a typical report looks like the one in figure 1. It's an overview of the number of passed, failed, and not executed test cases. It says nothing about the quality of your application. Relying on the wrong metrics is like flying blind without knowing it. That's the worst part. You feel like you know and have control, but you don't.

Therefore, issues appear last-minute or post-release even if the report was all green. To make a more accurate decision about whether you can go live, you need to look at business risk coverage. We dive into what this metric is and how to calculate it in the next section.

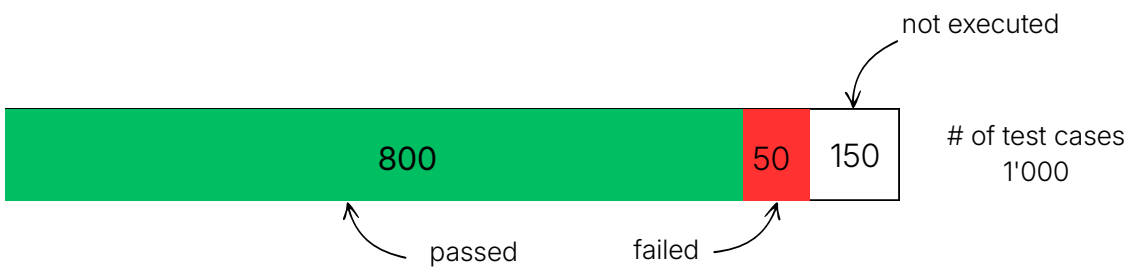

Figure 1: Software testing report today

Software Testing as bottleneck instead of value driver

Since testing is the last step in the process and in mostly not efficient enough, we see a pattern throughout all companies with the following three clusters:

1. Tests are automated, but automating and maintaining test cases takes too much time, resulting in having results late in the development process
2. No longer automate for this reason; domain experts test the software manually, either partially or over several weeks
3. Don't test automated (yet). Testing only partially or over several weeks

In all three cases, software testing becomes a bottleneck, slowing releases down. There are two levers to shift from bottleneck to value driver by improving efficiency and getting results earlier in the process:

- Test strategy and methodology → We dive into that in the next section
- Test automation tooling → get your team a tool that enables domain experts to automate testing. They spend the time going through the application manually anyway, and devs are free to handle the high feature pressure. We also cover how to become more efficient if you have a dedicated test automation team

2 Business risk coverage

Without solid information about your business risk coverage, you keep your team busy with testing without getting real insights from it. But how do you get from counting the number of test cases to a business risk coverage metric?

Some companies relay on test coverage (TC). So, let's dissect first what test coverage is and why it isn't the metric you should aim for:

Functions	$F_1, \dots, F_n$	
Tested	$F_1, \dots, F_m$	$m \leq n$

$$TC = \frac{m}{n}\,[\%]$$

As an example: If you have 200 functions and have only tested 120 of those functions, you have a test coverage of 60%.

$$TC = \frac{120}{200}\,[60\%]$$

The implicit assumption of test coverage is that you can count functions while **assuming that all functions have equal value.** But that's not the case. It's insufficient to talk about test coverage when we want to assess the behavior of the application. We need to talk about risk coverage and therefore assign weights to the functions.

Risk is frequency times damage. The higher the damage, the higher the risk. The higher the probability that things happen, the higher the risk.

$$\text{Risk} = \text{frequency} \times \text{damage}$$

The problem: Usually, you can't measure it. You don't have any numbers that give you clear information about frequency and damage.

Rapid risk assessment is the answer. Assign classes from 1-5 to these two dimensions.
The lowest risk = 1, the highest risk = 5.

$$\text{Risk} = \text{frequency} \times \text{damage}$$

lowest	1	=	1	x	1
highest	25	=	5	x	5

To get a more realistic granulation of the risk, we stretch out the relevance. An established and successful mechanism.

$$\text{Absolute weight} = 2^{\text{frequency}} \times 2^{\text{damage}}$$

lowest	4	=	2^1	x	2^1
highest	1024	=	2^5	x	2^5

Risk based approach example

You gain clear insight into how each individual capability contributes to your application's overall value, and consequently, its associated risk profile.

Countless projects have demonstrated this approach delivers the most efficient workflow with the most accurate risk weighting results.

From these absolute weights, you can derive relative weights. At any hierarchical level, the relative weights always total 100%.

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

Figure 2: Rapid risk assessment example

Actionable tips:

- Conduct the rapid risk assessment with your business analysts or testers. It can be completed in 1.5 days.
- Use the silent approval method (if no one objects to the suggested rating, it's automatically approved) to prevent people from defaulting to politically correct responses instead of sharing their genuine assessments.

Business risk coverage is the percentage of your business risk covered by test cases. It recognizes that some tests are more relevant than others.

When you sum the risk weights of the *m* tested requirements, you get your risk coverage (RC):

Functions $F_1, \dots, F_n$
Weighted $W_1, \dots, W_n$ $m \leq n$

$$\sum_{i=1}^{n} W_i = 100$$

$$RC = \sum_{i=1}^{m} W_i$$

Let's go back to our previous web shop example from figure 2. If you fully cover *Customer Tasks* requirement, you'll achieve 10.81% risk coverage.

Key takeaway

Risk coverage shifts the metric from how many requirements you've tested to how much risk those requirements represent. This gives you the needed foundation for making far more informed decisions about whether to ship.

Test strategy = efficiency

1 The 80:20 rule

You now understand the business risk for each function as well as the associated business risk coverage after each test run. Armed with this information, you can test much smarter and far more efficiently. You no longer need to test everything. Instead, you can focus on the requirements and test cases that contribute most to business risk coverage.

The well-known Pareto principle applies perfectly here: with just 20% of your most relevant test cases, you typically cover 80% of your business risk.

Since we've already identified which test cases are most relevant, you can cover the majority of your business risk with only 20% of the effort.

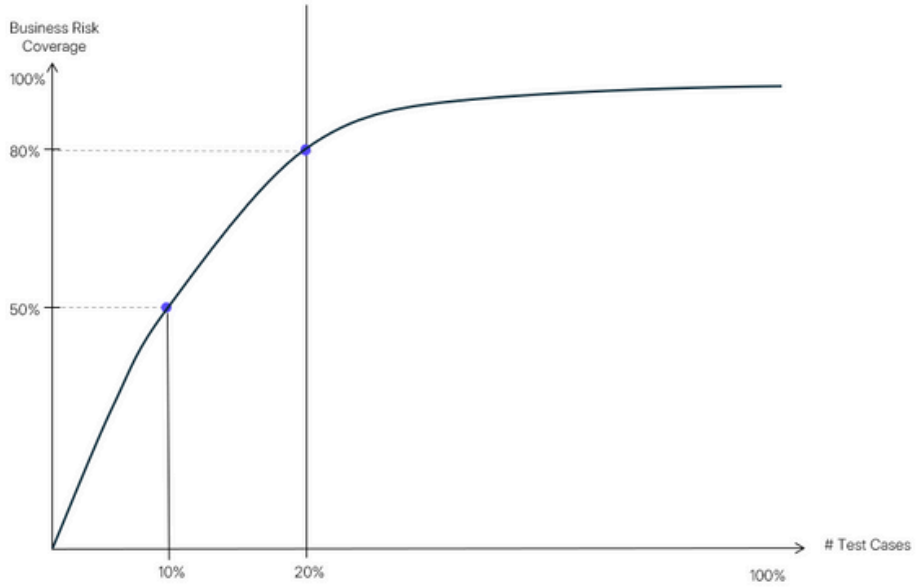

Figure 3: The 20:80 rule business risk coverage

Key takeaway

The 20:80 rule makes you drastically more efficient. You're testing the right things, tracking the metrics that actually matter, and you're no longer burning out your team with thousands of unnecessary test cases. These don't provide meaningful insights, they just keep everyone busy automating and maintaining them.

2 Methodology, smart instead of more

Your team can identify the 10% most relevant test cases intuitively, no training on test case design required. They instinctively know which ones matter most.

However, beyond that initial 10%, you risk falling into the trap of creating countless test cases that add redundancy rather than meaningful business risk coverage. To avoid this, you need proper test case design methodology and training.

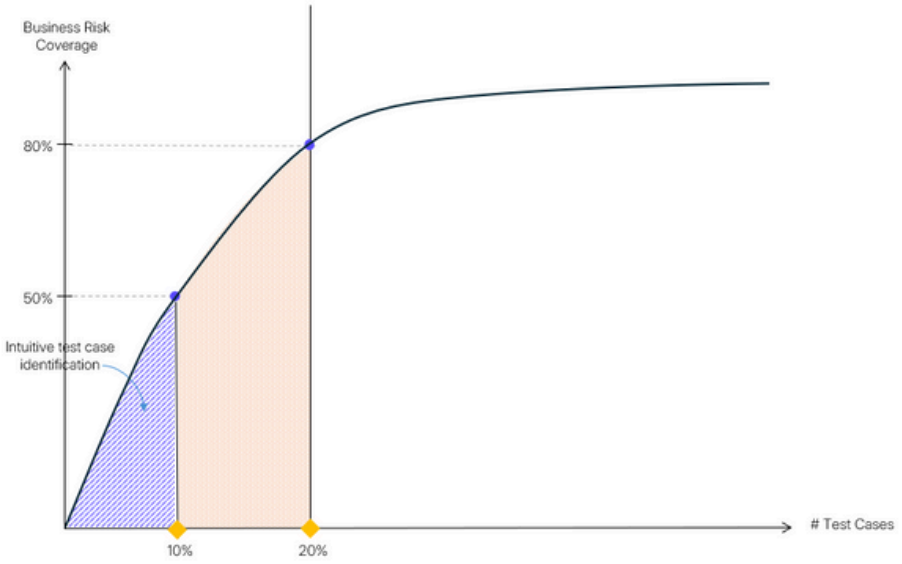

Figure 4: Software test design to increase business risk coverage

The 3 dimensions of software testing

Software testing operates in three dimensions: use cases, process variants (deviations from the happy path of your use cases), and data variants.

The test cases from the use case dimension represent that intuitive 10%, your smoke tests, as shown in Figure 5 on the next page. However, each use case also has process variants and data variants that require testing. These orange-marked test cases in Figures 4 and 5, your sanity checks, boost your business risk coverage to 80% and more.

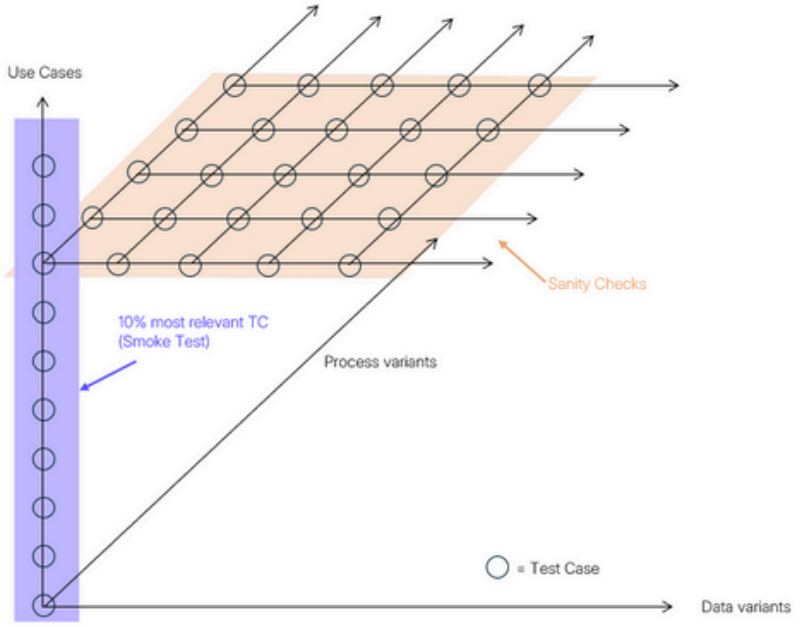

Figure 5: The 3 dimensions of software testing

Test automation ownership

1 Who should be in charge of test automation?

Market pressure demands constant feature releases. You need to maintain quality standards, but testing grows more complex with every release as there's more software to test.

As discussed earlier, most companies fall into one of these scenarios:

- Tests are automated, but automating and maintaining test cases takes too much time, resulting in having results late in the development process
- No longer automate for this reason; domain experts test the software manually, either partially or over several weeks
- Don't test automated (yet). Testing only partially or over several weeks

The key issue: who handles test automation? Developers need to focus on building new features to keep pace with market demands. They typically don't have time for test automation.

With a dedicated test automation team, you hit a different wall: automation takes too long, and they spend 25-30% of their time just keeping existing tests working as each release breaks something.

Or you go manual. Either because automation isn't in place, or because you've ditched automation tools when the effort to automate matched the effort to build the feature itself. So domain experts test manually.

What's the ideal setup to match agile speed without hiring expensive specialists or pulling developers away from feature work?

Enable your domain experts to automate tests (keep on ready, it's not what you expect)

Your domain experts know the application best and already spend up to 40% of their time either testing manually or coordinating with automation teams. By enabling them to automate directly, they can cut their testing efforts significantly, reducing testing costs by half. Plus, you won't need to hire specialists and you'll gain flexibility for future hires.

Now let's address the elephant in the room: they lack technical knowledge and therefore aren't qualified to automate software tests. And all those no-code tools can't handle the complex applications and user flows you're dealing with.

Therefore, to implement this strategy successfully, it's essential to have the right tool stack. You've heard it before, and I don't blame you if you don't believe this initially: there are tools specifically designed for business users that can handle even the most complex applications and make automation effortless. I'm aware of how many vendors have promised this but failed to deliver. We're in a new era of tools now, thanks to today's available computing power and advanced technology. So stay with me...

2 Tool stack

What tool delivers on these promises? TestResults. We've done extensive research and are admittedly biased. But as confirmed by numerous customers who've conducted even more thorough research and evaluation, TestResults is the only test automation tool they've found capable of delivering these results.

TestResults is a test automation platform built specifically for business testers and designed for complex enterprise and medtech software. It can test complete user journeys across different applications, systems, and devices, including medical and lab devices.

Since efficiency is crucial for cutting costs and catching issues early in the development process, here are some numbers our customers have reported:

- 90% faster than Selenium
- Up to 74% faster than Tricentis Tosca
- 0 flaky test cases, compared to 3-5 out of 10 test cases being flaky before

Ready for efficient software releases without last-minute surprises? Explore how TestResults can help you achieve the same results listed above by scheduling a demo.

If TestResults is what you were looking for, great! If it's not a match, no worries, we won't bother you after the call.

Conclusion

That was a lot of information and mathematical formulas.

Let's recap what we've covered:

- Counting test cases doesn't give you insights into your application's health
- **Business risk coverage is the most important metric** for deciding whether your application is ready for release
- The **rapid risk assessment** and **test design methodology** (The 3 dimensions of software testing) are your satring points
- **You only need 20% of your test cases** and can eliminate the rest, less effort, maximum efficiency
- Your domain experts should own test automation
- With a tool like TestResults, you enable non-technical testers to handle test automation seamlessly. TestResults makes you more efficient by design (schedule a demo)
- Every strategy covered in this cheat sheet **cuts costs and boosts efficiency. Combine them all, and you become unstoppable**

We see how powerful these strategies are for our customers and how dramatically more efficient they become. That's why we kick off new customers with a strategy workshop covering all these topics and more.