No. 03
July 2025

testing experience

AI, automation, culture, and the road to inclusive, resilient software.

Meaningful Metrics

Software accessibility includes gender diversity

AI vs Gilded Rose: Future of Testing or Just a Thorny Path?

How I build my own GenAI test data generator


AGILE
TESTING DAYS
Nov. 24 – 27, 2025
Potsdam, GER & Online
THE CONFERENCE
FOR SOFTWARE TESTERS
agiletestingdays.com

Summer Reflections and Learning Opportunities

The summer has arrived, and the solstice has quietly passed. Life begins to slow down a little, and so does the pace of work.

I've been part of this profession for nearly three decades, and I can't remember a time when so much was shifting and so quickly. It's not easy to keep up, especially when daily work, family life, social connections, and your own space to breathe are all happening at once.

With this edition, we simply want to offer you a few reflections and stories. Something to take in at your own rhythm, without pressure.

Learning has always been important, but these days it feels absolutely essential. If you're curious, have a look at the trainings we offer at trendig. And just so you know, when you book a training with us, you also receive a free online pass to the Agile Testing Days. For many teams with limited budgets, that's a valuable way to make learning accessible and inspiring.

Wishing you a beautiful summer. May you find time for sunshine, for your people, and for the things that matter most.

All the best,

Impressum

EDITOR

trendig technology services GmbH
Kleiststraße 35
10787 Berlin
Germany

Phone: +49 (0)30 74 76 28-0
Fax: +49 (0)30 74 76 28-99

E-mail: hi@trendig.com
Website: www.trendig.com

EDITORIAL
José Díaz

LAYOUT & DESIGN
Yalhen Rudolph

WEBSITE
www.testingexperience.media

ARTICLES AUTHORS
editorial@testingexperience.media

ADVERTISEMENTS
editorial@testingexperience.media

PICTURE CREDITS

CONTENT

6 **Meaningful Metrics**
Lisa Crispin

10 **AI vs Gilded Rose: Future of Testing or Just a Thorny Path?**
Christian Baumann

14 **Software accessibility includes gender diversity**
Tobias Geyer

18 **Business-Driven Test Automation with Sahi Pro**
Joerg Sievers

20 **Guarding the Gates: Why Security Testing Cannot Replace a Strong Security Culture**
Yvonne Johnson

24 **The role of a Product Owner in Quality Engineering and Testing**
Huib Schoots & Niek van Malsen

28 **Little's Law: on ready-for... queues**
Antony Marcano & Andy Palmer

32 **3 Lens Quality Coach Model**
Anne-Marie Charrett

38 **How I build my own GenAI test data generator**
Stephan Dreher

Meaningful Metrics

Author: Lisa Crispin

What comes to your mind when someone mentions metrics? Before you keep reading, stop and think for a minute. What metrics does the team you're working with now, or the last one you worked with, use? Are they helping (or did they help?) Organizations often gather metrics without much thought about how they will be useful.

Sadly, metrics are often used in destructive ways. I encountered a company where testers' performance reviews were based on the number of bugs they found. And, developers' performance reviews were based on how many bugs were found in their code. How well do you think testers and developers got along in that company?

That said, if our team or our organization wants to improve how we work, we need to try experiments. To know whether an experiment's hypothesis is proving to be correct, we have to measure progress in some way. When used with thought and care, metrics guide our efforts to continually improve.

Track your improvement experiments - step by step

Metrics measure progress. Modern software teams want to continually improve. The first step has to be deciding what you want to improve next. In my experience, the most effective way to decide this is through retrospectives.

After everyone has a chance to vent about their frustrations, and hopefully to celebrate achievements and contributions, figure out the biggest blocker to your improvement goals. You can do this by dot voting. Next, discuss what you might try to make that problem a little smaller.

When you come up with a change or technique to try to chip away at the biggest obstacle, design a small experiment to do together for the next two to four weeks. Create a hypothesis that includes a measurement to show whether your experiment is succeeding.

Review progress at the next retrospective. Based on your hypothesis measurement, decide whether to continue the experiment, tweak it, try a new experiment, or start working on another problem which is now the biggest one. Move towards your team improvement goal, one step at a time.

"Lead time indicates how fast a team can get feedback from production, once they have finished a new change."

Metrics that help in many contexts

All that said, I do know of some basic metrics that are useful in a lot of different organizations. Many organizations of all sizes find value in using these four "key metrics" identified by extensive research over the past ten years by Google's DORA research program. These provide a practical way to measure process quality, and they have been proven to correlate with team performance. Here's a quick summary:

LEAD TIME FOR CHANGES

This includes the time between committing a change (code or configuration) to the repository, and deploying it to production. I found this one confusing at first, because my own teams had a different definition of lead time. DORA's definition includes the time from the start of a merge request: code review, merging to trunk, running the continuous integration and deployment pipeline, and other stages needed through the deploy to production. Note that deploying does not mean that the change is released to customers. Release strategies such as feature flags and canary releases enable safe deployments.

This metric is focused on the deployment pipeline, because that is a key indicator of team and organizational performance. Lead time indicates how fast a team can get feedback from production, once they have finished a new change. The goal here is that over time, this lead time should decrease, while your team's performance improves. Some ways to shorten lead time include:

- Identify bottlenecks in the processes
- Breaking the changes down into smaller batches
- Adding automation
- Increasing tester/developer collaboration
- Improving the performance of your pipelines

DEPLOYMENT FREQUENCY

This metric measures how frequently and consistently an organization successfully deploys to production. By shipping small batches of changes frequently, an organization reduces risk.

Small batches allow teams to work at a sustainable pace, and have a better focus on what is valuable to customers. Shipping small batches frequently - weekly, twice a week, daily, even multiple times per day - is the secret known by high-performing software teams. Practices that lead to more frequent deployment include:

- Adding automated tests
- Adding automated code validation
- Breaking changes down into smaller batches

FAILED DEPLOYMENT RECOVERY TIME

How long does it take your team to recover when a deployment to production causes a failure? This is the time from when you notice the problem, investigate it, and do whatever is needed to correct it. A wide range of development and testing practices affects this metric. It's a good indicator of software stability and team agility.

Delivering small batches frequently lowers risk, and allows the team to recover from failed deployments more quickly. That means happier customers. Organizations need the right telemetry so that they learn about any production incidents quickly. They need a good safety net of trustworthy automated regression tests. Teams need good working agreements on how they respond to production issues. And they need to practice using those practices with simulations.

So many quality considerations go into a team's ability to respond quickly to production failures. High quality documentation is a must. A code base that is easy to understand and update is also essential. These metrics are all intertwined - you need a short lead time for changes in order to get fixes out to production fast. All of these factors mean less pain for customers. Here are some steps to take to shorten this recovery time:

- Improving the observability into the production environment
- Improving response workflows
- Improving deployment frequency and lead time for changes so fixes can get into production more efficiently

CHANGE FAILURE RATE

This metric, which reflects both process and product quality, is defined as a percentage of deployments that cause a failure in production such as downtime, degraded service, or a need to rollback the deployment. In other words, how often does a deployment to production fail? Teams with a high change failure rate may be spending more time fixing problems than developing new features.

"Teams that spend a lot of time fixing problems have less time to devote to new features."

Teams practicing continuous delivery may see more failures, but an overall better failure rate. For example, if a team deploys five changes a day, that means 25 changes in a week. If five of those 25 changes fail, the rate is 20%. If a team deploys only once a week, and that change fails, they have a 100% failure rate. So, don't aim for fewer deployments.

This metric reflects both product and process quality. The synergy of combining change failure rate with time to restore service is powerful. Teams that spend a lot of time fixing problems have less time to devote to new features. All the leading development practices that help teams produce maintainable, testable, operable code, building quality in and testing effectively, lower the change failure rate.

Again, delivering small batches of changes more frequently is the key for better results. Other ways to improve this metric:

- Finding the right balance between stability (as measured by change failure rate and failed deployment recovery time) and throughput (deployment frequency and lead time for changes)
- Improving the efficacy of code review processes
- Adding automated tests (this helps with so much!)
- Breaking changes down into smaller batches

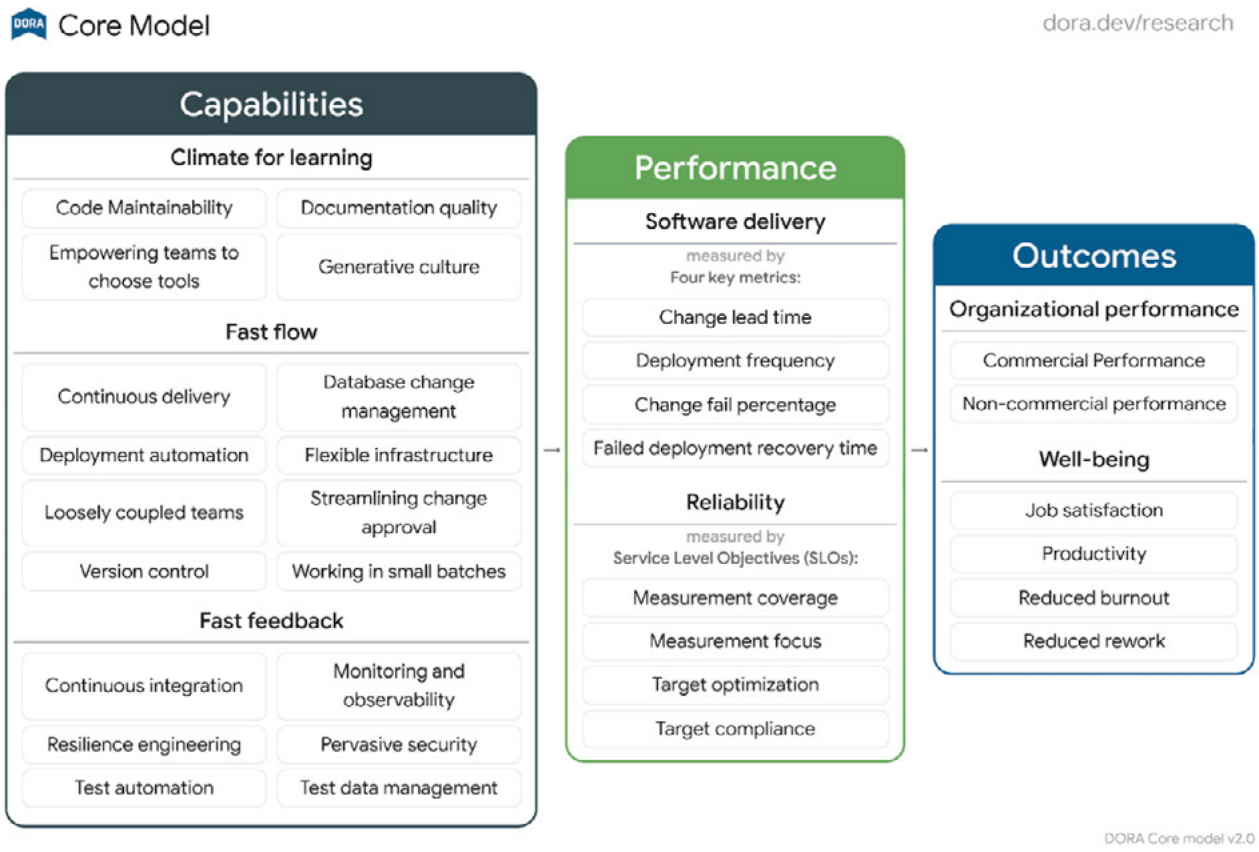
DO YOU NOTICE ANY PATTERNS?

Process and product quality, as reflected in these key metrics, can be greatly improved by deploying smaller batches of changes more frequently to production. That is the basis for modern development practices that are iterative and incremental, what many people call agile.

We've known for decades that having trustworthy automated regression tests gives teams confidence to make small changes quickly. Looking for bottlenecks in your team's path to production, such as code reviews, is a great way to start improving.

TRACKING THE DATA

Measuring doesn't have to be complicated. Sure, it is nice to instrument your code to capture data and



The DORA Core Model, showing where the key metrics reflect performance

events in log files. There are many tools available to turn that information into highly visible dashboards, alerts and analytics. And, you can simply track information such as the four key metrics or whatever you are using to track progress in a small experiment in a spreadsheet or a document. If you do decide to try the four keys, DORA provides a form where you simply answer four questions and see how your team or organization compares with similar ones.

WHAT TO AIM FOR

The 2024 DORA research report notes that elite improvement is more important than elite performance. Start conversations in your team about what you'd like to improve next. Brainstorm ways you could make your biggest problem a bit smaller. Try small experiments together, making sure to specify your expected result in a measurable way.

See if metrics that have proven to be useful across many types of organizations might apply in your own context. Use your retrospective workshops to evaluate whether the measurements being tracked are helpful.

Lisa Crispin
Co-founder
@ Agile Testing Fellowship, Inc.

AI vs Gilded Rose: Future of Testing or Just a Thorny Path?

Christian Baumann
Principal Test Architect
@ MaibornWolff GmbH

Author: Christian Baumann

Picture AI stepping into the role of testers, creating test cases and exploring their potential to support the testing process. This article puts them to the test with the quirky and infamous Gilded Rose Kata, to see if these AI tools can actually lend a hand in testing or if they'll trip over their own code.

THE GILDED ROSE TREASURES

The Gilded Rose Kata, a popular coding exercise, is about an inventory management system with complex rules and changing business logic. Its unclear requirements are similar to real situations, making it a good way to test how well test cases are designed. For testers, the task is to make automated regression tests that keep new features working without breaking old ones. The different items and their special behaviors, such as the legendary "Sulfuras" or the fragile "Aged Brie," add challenges to the testing process.

This task not only checks developers' ability to add features without breaking the old ones but also pushes testers to think about edge cases, tricky rules, and keeping the system working as expected. By solving these problems, the Gilded Rose shows what happens in real software systems, where unexpected interactions often appear. For testers, this means using tools and methods that go beyond manual work, opening the door for approaches like AI.

THE TESTER'S GRIMOIRE: GOLDEN MASTERS AND AI VALIDATION

To evaluate the outcomes of the different LLMs, we used golden masters: Here, reference results under set conditions serve as the standard to compare later test results. This method allowed for a steady check of AI-created test cases. Each test case was carefully made to include starting and final values, expected behaviors, and the rules for each item type. This strong setup gave a good basis for checking how AI performs.

Golden master testing works well when requirements are complex or changing. It acts as a safety check to make sure any system changes can be compared to an existing standard. This method not only makes it easier to find problems but also gives a way to check if AI-made test cases match what is expected. In the Gilded Rose test, these golden masters were key in seeing how AI understood and applied tricky requirements.

MACHINE DIVINATIONS

The main focus of this work was on three different tasks, each meant to test a specific part of AI's abilities in testing. In the first task, LLMs were asked to create test cases using only the Gilded Rose's requirements. The second task was about making automated tests from these requirements, and the third skipped the

Criterion/Model	ChatGPT 01	ChatGPT 4o	ChatGPT 4o mini	Google Gemini	Llama3.2 3B	Mistral Nemo 12B
Described reqs?	Yes	Yes	Yes	Yes	Yes	Yes
Described test?	Yes	No	No	Yes	No	No
Grouping	Yes	Yes	Yes	Yes	Yes	No
Number of tests	17	-	-	13	-	-
Grouping auto	Yes	Yes	Yes	Yes	No	No
Numer of tests auto	16	16	17	9		
Runnable?	Yes	Yes	Yes	No	Yes	Yes
Passing?	No	Yes	No	Yes	No	No

requirements, asking the AI to automate pre-made reference test cases.

The results were revealing. When creating test cases from requirements, the LLMs showed
mixed results. Larger models, like ChatGPT 4.0 and Google Gemini, showed a decent understanding of goals and testing methods, such as boundary value analysis and equivalence partitioning. Smaller models, however, often failed to meet even simple expectations, producing outputs that were incomplete or off-topic.

In automation, the gap between models became clearer. Most models could write code that was correct in structure, but the ability to make working tests varied a lot. Google Gemini often stood out with its careful approach, adding comments and suggesting fixes. But even the better models sometimes misunderstood requirements or made wrong guesses, showing that human review is still needed.

"Even the better models sometimes misunderstood requirements or made wrong guesses, showing that human review is still needed."

The third task—automating ready-made reference test cases—showed an interesting pattern. With clear data and instructions, larger models did well, turning test cases into working code with good accuracy. But their habit of changing instructions instead of following them exactly led to issues, especially in edge cases like the special behavior of "Aged Brie."

Some models showed creative thinking, offering edge cases or improvements not mentioned in the requirements. While this creativity is a good sign, it also shows why testers need to check AI results closely. Trusting these outputs without checking could miss problems or cause unexpected issues.

THE KEEPERS' JUDGEMENT

Several important lessons came from these experiments. First, the quality of AI results depends a lot on how clear and well-structured the input data is. Unclear requirements or instructions often caused mistakes, showing how important it is to write good prompts.

Second, while LLMs can speed up some parts of making and automating test cases, their results must be checked carefully. This is especially true with complex requirements, where misunderstandings can easily happen.

Also, the back-and-forth process of working with AI became clear. Writing good prompts often needed multiple tries, like having a conversation with the model. This process shows that AI works better as a partner to testers rather than a one-time fix.

Lastly, not all LLMs perform the same. Bigger models with more memory and better reasoning skills often did better than smaller ones. This difference suggests that investing in better models or hardware might be a good idea for companies wanting to use AI in testing.

AGILE
TESTING DAYS

Nov. 24 – 27, 2025
Potsdam, GER & Online

Early Bird

Registration open

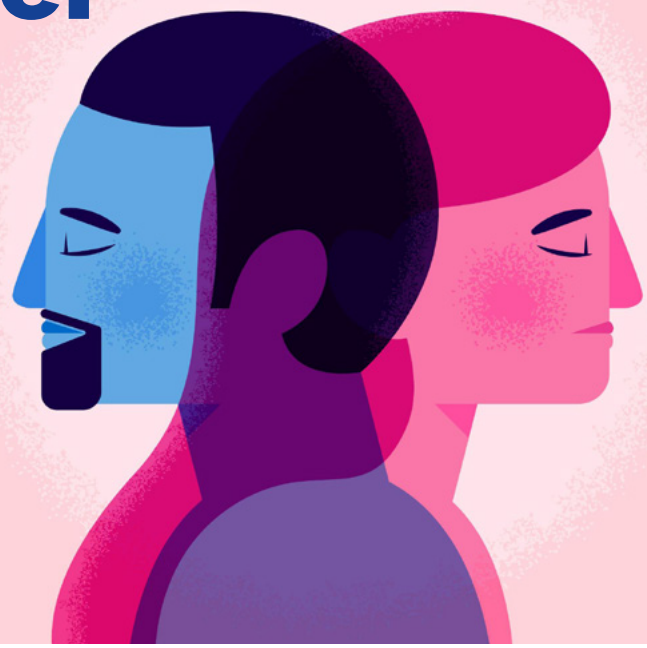
Onsite & Online Available

Only until September 21, 2025

agiletestingdays.com

Software accessibility includes gender diversity

Author: Tobias Geyer



Would you create a user account on a website which offers only one salutation - and it's not the one you use?

Or would you accept that you have to delete your user account on a website and create a new one, losing all your data because you made a mistake in your personal data and can't correct it?

Would you be okay with lying about yourself because limitations in software force you to do it?

While those situations may sound unrealistic or outright ridiculous they're a sad reality for a portion of the people using software which deals with personal data. But why - and what can we as software testers do to improve this?

IT'S NOT AN EDGE CASE

While there is data on **LGBTQIA*[1]** people and their distribution in the overall population, specific data on the distribution of trans and non-binary identities is scarce. Census data from Canada, the US, England and Wales shows that between 1:300 to 1:167 people don't identify with the gender they were assigned at birth. Not all of them are able to live as their real gender or change their name or pronouns and not all of them use the software you're working on. Then again most software deals with more than 300 registered users, so there's a real chance that some of them are **trans[2]** or gender diverse.

Which means that software which doesn't actively take gender diversity and the needs of LGBTQIA* people into account will not be usable to parts of the potential user base. Let's take a look at problems in existing software, improve our tester knowledge and learn about possible solutions.

A LACK OF GENDER OPTIONS / SALUTATIONS / PRONOUNS

When signing up for a user profile there is a lot of data to enter - first and last name, address, billing data and so on. Often either a gender or salutation must be chosen from a predefined list, sometimes pronouns must be given as well.

In case the salutation is limited to a predefined list of “Mr” / “Ms” or the gender is limited to “man” / “woman” setting up an account which matches their true self is impossible for people who don’t fall into those binary categories, e.g. **non-binary**[3] folks. The same happens if there’s a predefined list of pronouns. This is not a theoretical problem, a non-binary friend of mine can’t order from the online store of a large optician franchise in Germany because it’s not possible to use a gender neutral value for the mandatory salutation. There are two solutions to that: Don’t ask for gendered data at all, chances are you don’t really need it. If you absolutely have to ask for it, give users a free text option and make sure that the software deals well with the fact that some users may leave it empty. There’s even a legal component at play. In January 2025 the European Court of Justice ruled that the french railway company SNCF may not ask for a gender when booking train tickets since data collection under GDPR "must be adequate, relevant, and limited to what is necessary in the light of the purposes for which those data are processed." So if the first solution seemed impossible to you, that ruling might be a good opportunity to reconsider it before you’re forced to by a lawsuit.

UNCHANGEABLE PERSONAL DATA

Just like some people move and have to change the address in their user profile, some people change their name or their gender. However while changing the address themselves is mostly possible for users, changing their name or gender often isn’t. If they’re lucky, people can get those changes done by contacting a human support person, if they’re unlucky, they have to delete their user account and create a new one.

Having to get in touch with a support person may seem like a usability issue, a cumbersome workaround. It’s a problem which is well known to people who got married or divorced and changed their last name. For trans people it’s more than that because it requires them to mention their **deadname**[3] to another human - which is painful to them. It also puts them at risk of a support person starting to ask them questions about the change or be openly hostile towards them. The latter has been known to happen, especially when making changes which indicate a changed gender. In addition to that the sound of a voice is often associated with a gender, so when calling support on a phone the requested changes can be rejected because support staff think that the user has “the wrong voice”.

There’s an easy solution to that: Trust your users to change all of their own data. There is only a very small set of situations in which it’s required by law to use the exact legal name of a user, so requiring the user to supply their legal name should be a last resort. Using Germany as an example, those situations are when in front of a court, international travel and being the CEO of a DAX company. If the software covers one of those situations, have a process in place which allows users to provide the necessary documents and request the change electronically. Even then the software may run into other issues, see below.

> *"Setting up an account which matches their true self is impossible for people who don’t fall into those binary categories."*

USERNAMES OR EMAIL ADDRESSES AS USER IDENTIFIERS

From a software development perspective it may seem like a good idea to use an email address or a unique user name as a primary identifier for data storage. Which gets problematic quickly when this primary identifier can’t be changed and is visible to the end users. Sometimes users are forced to use that identifier during the login process. Many email addresses contain the name of the user, some user names contain gendered terms (like “guy”) and all of those can become outdated as the user discovers more about themselves and their gender identity. As

shown in the previous chapter, being able to change this identifier themselves is crucial for users.
Even if the identifier can only be seen in the backend by support staff it can cause problems to the users because it reveals sensitive information about them and their past without their consent.

The solution to this is simple: Make usernames and email addresses changeable by the user.

USING GENDER AS SUBSTITUTE DATA

A common trap software developers fall into is the assumption that knowing the gender of a person enables them to derive the preferred salutation of that person, their pronouns and even information about their body. At my last visit to the dentist I was asked to fill out a digital form with my personal information and as soon as I marked my gender as "male" I was no longer able to answer the "are you pregnant" question.

Trans men exist and can get pregnant – just as trans women exist who can't get pregnant. Some **cis**[6] women can't get pregnant either so connecting this question to the "gender" data is at best careless and at its worst endangering the wellbeing of the person being asked. As software testers we're used to asking clarifying questions about vague statements in requirements or acceptance criteria. We know that precise language matters and we need to approach gendered data the same way.

Which is already the solution to this problem: Don't make assumptions and don't derive gendered data. Ask for it precisely and openly. "Could you be pregnant" may be the correct question to ask in some contexts, "do you have a uterus" may be valid in others while "what's your salutation" is valid in even others. If you need all of these data points, don't hesitate to ask for each one individually. You might be surprised how often the answers from one person don't match your assumption.

INTRANSPARENCY ABOUT DATA USAGE

As a general rule it's best to trust users to enter the correct data, especially their name and gender marker. However that can cause problems depending on how the data is used and who gets access to it. Some people change their name socially but don't take the steps to change it legally. Others change their name only in certain social settings and use their assigned name in others.

> *"Be transparent about what the data is being used for, for example by adding the hint 'this name will be shown on your report cards'."*

If the software used by university students sends report cards to their parents but the student hasn't told their parents about their changed name, this can lead to them being involuntarily **outed**[7]. If the booking process for a rental car checks if there's a valid drivers license under the user's name against government systems using the name given by the user may result in the booking being declined.

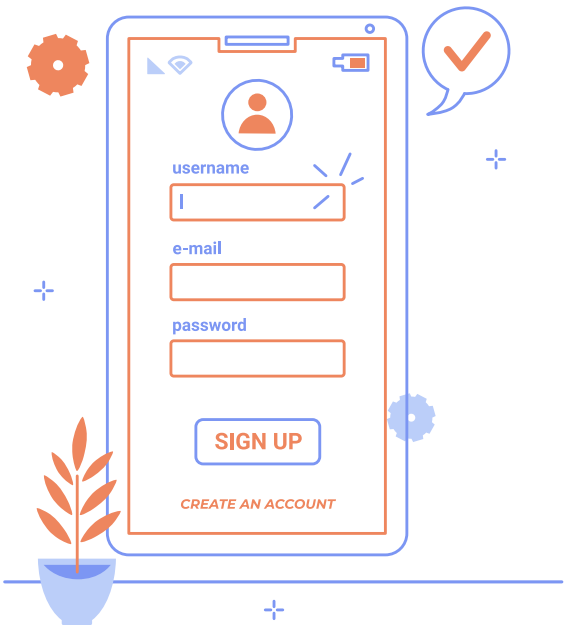
The solution to this consists of three parts: As before, use precise language like “name on your driver's license”. Be transparent about what the data is being used for, for example by adding the hint “this name will be shown on your report cards”. Finally allow the user to enter their name and their legal name separately if the latter is needed.

CONCLUSION

As testers it’s part of our job to know that things can be different. We don’t just test on one browser or screen size because we know that the environment where the software is used can be different. We keep an eye on accessibility support because we know that the needs of our users can be different. Hopefully you will also keep an eye open on how well your software supports gender diversity from now on.

Ignoring the gender diversity of your users can put hurdles in their way, similar to those put up when accessibility needs are ignored. Sometimes users will find workarounds to those hurdles, sometimes users will just not use the software and sometimes you may find yourself in a lawsuit because of it.
These risks can be mitigated and the problems and solutions shown in this article can serve as a starting point for that.

Finally, making sure that software is accessible for everyone isn't just a way to avoid lawsuits. It's the ethically right thing to do.



Glossary:

1. LGBTQIA* - An acronym for “Lesbian Gay Bisexual Trans Queer Intersex Asexual”, the “*” indicates that there are more identities which fall under this umbrella.

2. trans - An adjective describing people who don’t identify with the gender they were assigned at birth.

3. non-binary - An adjective describing a person who does not identify (exclusively) as a man or a woman. Non-binary people may identify as being both a man and a woman, somewhere in between, or as falling completely outside these categories.

4. deadname - The name that was assigned to a (trans) person after birth but which they’re no longer using. In case of trans people this name often doesn’t match their gender identity and being confronted with the deadname often causes them distress.

5. gender assigned at birth - The gender a person was assigned after birth based on physical characteristics of their body. This can be different from their gender identity.

6. cis - An adjective describing people who identify with the gender they were assigned at birth.

7. to out someone - Exposing someone’s lesbian, gay, bisexual, transgender or non-binary identity to others. Outing someone can have serious repercussions on their employment, economic stability, personal safety or religious or family situations. On top of that, outing someone without their consent is illegal in some jurisdictions.

Tobias Geyer
Tester, Conference speaker
@ VECTOR Informatik

Business-Driven Test Automation with Sahi Pro

How to win business departments for test automation

Author: Joerg Sievers

In a large corporation, things are different than in a small or medium-sized enterprise, or even in a startup. Departments are larger, and business and technical knowledge is more distributed. Having interdisciplinary teams, where Business Analysts (BAs), Developers (DEVs), and Testers (QA) form a unit, is already a very advanced structure. How can you get BAs and QA involved in testing or even test automation early on in sprints of, for example, three weeks, so that you don't end up dumping the entire workload from DEVs onto the other two roles?

For example, if the „three amigos“ can use the same tool from different perspectives. BAs define the test objectives, the expected parameters, and the keywords for the test steps in a spreadsheet, or even entire sentences as keywords, together with the other two roles. The focus is on business-critical use cases, not small units that DEVs could easily test themselves!

Initially, the test steps are red because they are not yet implemented in automated functions. DEVs could now program the functions, or the other two roles could record the individual steps, provided the SUT already has the functionality. Once recorded, a test step can be reused in all other areas of the project. We teach the teams to always apply AAA (arrange, act, assert or in German "Schienen-Ersatzverkehr", setup, execute, and verify) in each step and, if possible, to always subdivide it into these small parts to ensure reusability. The steps focus on "Where am I? Am I in the right place in the program?" (Arrange/Setup) to perform the action (Act/Execute)? Verify that the result of the action step has been fulfilled (Assert/Verify).

This way, the teams get the test cases running first. Once everything works, we try to make any steps that might fail and need a bit more attention more robust. Finally, any parameters (names, business data, etc.) are stored in so-called Data Drives (these

Joerg Sievers
Test Service Specialist
@ SIGNAL IDUNA Gruppe

can be CSV files, databases, etc.). This separates the test data from the test cases and allows us to use the same tests with different data constellations in different environments (test, integration, acceptance).

BAs and QAs mostly look at the spreadsheet and see little of the functions in the background, which are intelligently recorded in JavaScript by the tool. During recording, the verification steps (Assert/ Verify) are directly included, and selectors (called accessors) are stored in a central project file and given a synonym. So, if you need to change an accessor, you do it once, and all tests using the same accessor fix themselves. The accessor repository layer thus serves to make the tests adaptable with minimal effort.

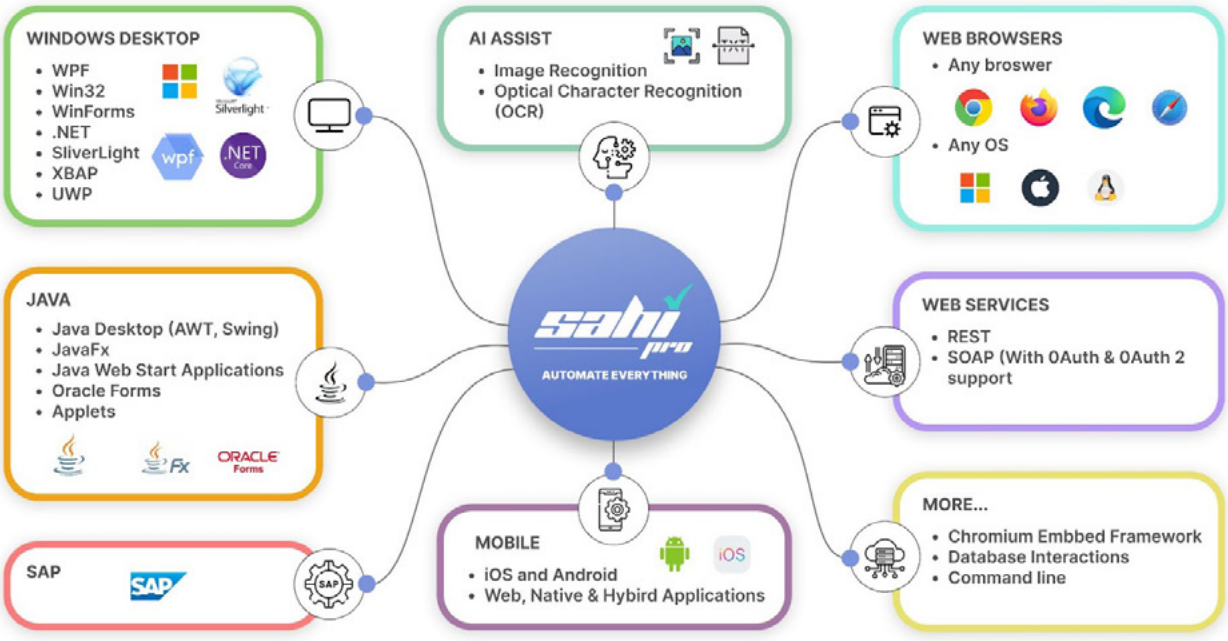
When DEVs read the above procedure, they inevitably think of behaviour-driven development (BDD). That's why this approach is called BDTA, business-driven Test Automation, because the basis of these acceptance tests lies in the business logic of mostly critical use cases and checks the behaviour of the system.

Using the approach briefly outlined here, we have succeeded in getting the "business department" or rather its representatives, the BAs, enthusiastic about helping with test automation in a dozen projects. They feel comfortable with the spreadsheet view and assisted recording of test steps, as do the DEVs who can do the whole thing in their IDE (Integrated Development Environment, e.g. IntelliJ).

By integrating the projects into ANT, Maven or Gradle, you can run the tests directly from the IDE on a remote "runner" and get the results back into the IDE. This enables DEVs to avoid regressions at the push of a button when they adjust business processes. This doesn't mean renaming a button but breaking the process. You don't need to tell the testing tool about anything else, because it automatically recognizes any cosmetic changes and can still execute the business transactions.

Tests that DEVs can run directly without considering any external systems or operating complex upload and download processes are not executed in this test stage, but, for example, with Cypress, Playwright & Co. However, the advantage of involving the business departments/BAs in the acceptance tests is obvious: you can finally focus on exploratory testing of new functionalities and no longer must verify that the business-critical processes still work.

Sahi Pro allows teams to use this approach to test different technologies and frameworks[1] without changing the tool or the way to work.



[1] Image Sahi Pro

Guarding the Gates:

Why Security Testing Cannot Replace a Strong Security Culture

Author: Yvonne Johnson

Repeat after me, dear reader: It does not matter how secure my code or configuration is if IT security processes and policies are not refined nor adhered to. This one simple truth has been the harbinger of many a success story for my team, where we were able to gain a foothold into the customer's network over the internet, escalate our privileges, move to further servers, and ultimately gain the keys to the kingdom.

Hi, my name is Yvonne Johnson, and I have seven years of experience in penetration testing and red teaming (see the table below for more information). I am currently working at Deutsche Telekom Security GmbH as a Senior Red Teamer. There, I conduct adversary simulations for companies both large and small, in order to improve their defenses and detections against real-world attacks. As a red teamer, I have the unique opportunity to observe a company's security posture from an organizational

Penetration Testing	Red Teaming
Finds technical vulnerabilities.	Finds technical and organizational vulnerabilities.
Tends to have a smaller scope of one system or application.	Covers the entire organization.
Goal is to uncover as many weak points and misconfigurations as possible.	Goal is set by customer (e.g. obtain domain admin privileges) and the red team focuses only on the vulnerabilities that help them achieve this specific goal.
Carried out in development or test environments.	Carried out in production environments.
Can be conducted by one person in one to two weeks.	Conducted by a team of people over a longer period (from weeks to months depending on which scenario the customer would like to have simulated).
Defenders are notified of the test beforehand. Incident response processes are not activated if an alarm is triggered.	Defenders are not aware of the test. The red team attempts to remain undetected. The red team may purposely trigger an alarm to test incident response processes.

Table 1: Differences Between Penetration Testing and Red Teaming

level. Over the years, it has become clear that testing and culture go hand in hand when it comes to protecting our assets.

I understand the argument I am making in this article sounds like common sense. However, the prevalence to which this problem has allowed my team to overtake entire organizations within days has convinced me that this point needs to be shared with the masses. So, please put on your metaphorical hacker hoodie, dear reader, and come with me on a little red team journey.

INITIAL ACCESS

We find ourselves in front of a computer screen. The browser is open to the login page of an application belonging to our customer. The software is up-to-date, and there are no known vulnerabilities. We do what any good hacker would do and type in "admin" for both the username and password. A second later, the page reloads and—bingo, we're in! With our newfound access, we click through the admin portal and discover that the application has built-in functionality that can be misused to execute code on the customer's server. Utilizing this function, we run our malware and begin to send it commands through our Command & Control (C2) infrastructure.

PRIVILEGE ESCALATION & LATERAL MOVEMENT

The C2 communication allows us to search the compromised machine for any data that can help us achieve our goal. In a file named "config.yaml", we discover credentials for an Active Directory service account. The service account is allowed to connect to SMB shares open to all user accounts in the network, one of which contains full server backups in the form of .VHDX files. We download a server backup file and extract the local administrator password hash, granting us the ability to log on to a new server as an administrative user.

DOMAIN DOMINANCE

Since we have administrative privileges, we can extract the password hashes on the new server, as well. It seems a domain admin account has logged into this server before, and their password hash in included in the results. Jackpot. We have gained domain admin privileges without exploiting a single technical vulnerability.

It does not matter in this scenario if the web application has undergone extensive testing to remove all software vulnerabilities or if there are no Active Directory misconfigurations for the attacker to exploit. Since a strong password policy

was not enforced on the web application, we were able to guess valid credentials and misuse a built-in function. Hardcoded credentials in a file stored on the server allowed us to escalate our privileges because a secrets management strategy was not in place. Sensitive files were available to every user in the domain as an authorization and authentication concept was not considered when exposing the SMB share. Finally, we were able to completely take over the domain as best practices like not logging into servers with a domain admin account were not followed.

"Be transparent about what the data is being used for, for example by adding the hint 'this name will be shown on your report cards'."

I could name many other examples such as finding valid SSH private keys to important servers in public Git repositories or useful application credentials in internal company wikis on public pages, but the point remains the same. When processes and policies are not implemented, organizational weaknesses that are just as harmful as technical vulnerabilities arise. Security testing will always play a vital role in protecting our applications and infrastructure from malicious actors, but it alone is not enough to prevent successful attacks. We need developers who think twice before choosing a weak password or exposing credentials in a public manner. We need administrators that understand that security hygiene is as equally important as secure coding. We need managers to motivate teams to appreciate and apply security policies. In other words, we must foster a culture where each employee knows their part in keeping a company safe.

Now, with that said, go forth and change those passwords.

Yvonne Johnson
IT Consultant

AGILE TESTING DAYS

Nov. 24 – 27, 2025
Potsdam, GER & Online

DEEP DIVE TRACKS

A *structured journey* through today's *key testing topics*:

GenAI

Test Automation

Security Testing

Up to 33 hours* of immersive learning *across 3 Days

agiletestingdays.com/program/deep-dive

The role of a Product Owner in Quality Engineering and Testing

Author: Huib Schoots & Niek van Malsen

This article highlights the critical role of Product Owners in Quality Engineering and Testing. To learn more about what Quality Engineering is and how it aligns with testing, see the blog post "How testing aligns with Quality Engineering[1]" written by Boyd Kronenberg and me.

The Product Owner (PO) is responsible for maximizing the value of the product and the work of the Development Team. Their main responsibilities are:

- Visionary Leader: defines and communicates the product vision and strategy to ensure alignment with business goals.
- Key decision-maker: Balances stakeholder needs and technical constraints
- Backlog Management: Prioritizes and refines the product backlog based on value, dependencies, and risk.
- Stakeholder Collaboration: Engages with stakeholders to gather requirements, provide updates, and ensure continuous feedback.
- Quality: The delivered product meets quality standards and customer expectations.
- Adaptability: Continuously adapts to changes and new insights to mitigate risks and improve the product.

Recently there was a survey on LinkedIn[2] on who is responsible for Quality. Although the team is responsible for doing the work, the PO is responsible for the product quality, since they decide about what work needs to be done and what the priorities are. POs play a vital role in building quality products. To be able to explain this well, let's first look at how software development works.

SOFTWARE DEVELOPMENT IN THE VUCA WORLD

Software development must deal with the VUCA world:

- Volatility: The dynamic rate of changes requires continuous adaptation and innovation
- Uncertainty: The unpredictability of events and issues in software development necessitates flexible planning and risk management
- Complexity: The complexity of interdependent systems and components require a holistic approach to problem-solving and a deep understanding of the IT landscape
- Ambiguity: The difficulty in accurately assessing reality in a complex and volatile landscape means that DevOps teams must be prepared to handle ambiguity and make informed decisions

VUCA highlights the need for (DevOps) teams to be agile, adaptable, and resilient in the face of rapid changes, unpredictability, complexity, ambiguity, confusion, new insights, and half answers. As a result, we face considerable risks. It underscores the importance of continuous learning, collaboration, and effective risk management to navigate the challenges and deliver quality software. We deal

1. https://labs.sogeti.com/how-testing-aligns-with-quality-engineering/
2. https://www.linkedin.com/feed/update/urn:li:activity:7292666987741130753

with users and clients who do not know exactly what they want or cannot explain it well. With IT teams that do not always understand the business well. How do we deal with these risks?

SOFTWARE DEVELOPMENT IS ABOUT VALUE AND RISKS

As a team, gaining insight into value and risks requires a detailed understanding of the workflows, the product, and the risks surrounding it. Teams collaborate to solve problems that threaten the value of the product or the timely, successful completion of the work.

Risk-based thinking involves organizing the software development process around suspected risks. This means conducting risk analyses continuously and creating a Quality Strategy for the entire software delivery process. Risk-based thinking is a powerful approach to mitigating risks and delivering quality. Besides providing insight into risks, it also deepens understanding of the product and promotes "common or shared understanding." This shared understanding helps reduce misunderstandings, improve effectiveness (and efficiency in the long run), and deliver quality outcomes. It is essential for effective collaboration and successful project delivery. According to the State of Teams research conducted by Atlassian in 2023[3], teams with shared understanding better meet stakeholders' expectations, use resources efficiently, develop new ideas, take pride and find a sense of purpose in their work, and experience increased motivation, energy, and enthusiasm.

THE ROLE OF A PO IN THE VUCA WORLD

POs are the ones responsible for backlog management. They must ensure the team achieves a shared understanding of the work items. Unfortunately, many refinements are too short, poorly prepared and have insufficient outcomes like vague requirements, missing acceptance criteria and team members not fully understanding what the epic/feature/story is about. Feeling pressured to deliver, teams do not take enough time to learn. Eventually this will slow them down by delivering lower quality. Our experience is that the team members often feel pressured to deliver and do not speak up when they do not fully understand what is expected from them. We also see POs focus on delivering features too fast. They push to do efficient and fast refinement, which often means going through stories as fast as possible. Instead of focusing on sustainable quality, which in the long run is cheaper, faster and leads to more business value and happier customers and stronger teams. Remember that deep learning takes time: so, start promoting practices like discussing risks, asking more questions, visualizing where possible and creating acceptance criteria.

SOME GOOD PRACTICES WHICH POS CAN ADVOCATE

- Reducing Work in Progress (WIP) and slicing stories as much as possible

Essential practices for maintaining a sustainable pace and delivering quality software. WIP limits help teams avoid overload, ensuring tasks are completed effectively by preventing bottlenecks, reducing context switching, and shortening lead times. Small stories enable teams to manage risk and receive quick feedback. According to the

3. https://www.atlassian.com/blog/leadership/shared-understanding
4. DORA | Accelerate State of DevOps Report 2024

State of DevOps report from 2024[4], this practice helps teams break down larger tasks into smaller, manageable pieces, allowing for more efficient and effective development. These practices help teams deliver incremental value, reduce complexity, and improve their ability to respond to changes.

- Risks analysis

Conducting risk analysis is crucial for teams to identify what needs action and to deepen their understanding of the product. Building quality software requires both solution-oriented and problem-oriented thinking. While most team members adopt an optimistic mindset, asking questions like "How should this work?", testers often take on a pessimistic mindset, asking "What if X happens?". Both mindsets are essential for developing quality software.

"Building quality software requires both solution-oriented and problem-oriented thinking."

Discussing risks within the team leads to better software by exposing the most significant pitfalls. Savvy POs engage in risk discussions with the team to enhance their understanding of the product and the necessary activities for delivery. This helps prioritize quality measures effectively. However, this approach may extend the refinement process which is worthwhile because learning takes time.

- Feedback loops wherever possible

Teams should implement as many feedback loops as possible. While this may initially seem inefficient, it will significantly speed up teams over time. It enables teams to continuously improve the product and/or themselves. Common scrum ceremonies are valuable feedback loops, but there are many more: refinements to ensure the team understands what to build, clean coding: code reviews, pull requests, static code analysis, sufficient unit tests in development pipelines, continuous integration to check if the code merges without issues in pipelines with automated regression checks at various levels, testing, and monitoring.

Unfortunately, Product Owners (POs) are often not involved in developers' workflows, which is a mistake. Technical debt results from choosing quicker delivery over clean code and thorough testing, leading to additional rework and maintenance challenges in the future. POs are often unaware of the time and costs involved in fixing the issues caused by shortcuts. An excellent PO will consider the risks and costs associated with repairing technical debt before taking shortcuts. They know costs are always higher than expected.

- POs are heavily involved in testing

In many teams, testers are responsible for deciding if the product can be shipped. Once the tests are completed, the story is marked as "done," and the product is ready for shipment. However, it should be the PO making this decision. POs should ask the team to report on three key topics: how the product is performing, what kind of testing was conducted to determine this, and what the remaining risks are in the product and the process. This is known as the Testing Story. With this information, the PO can decide whether the product can be shipped.

Discussing testing with the PO helps the team highlight risks and challenges in testing. The PO can then help improve testability for more effective and faster testing or accept the risks involved and proceed by going slower. The responsibility of determining "how much testing is enough" should be a collaborative discussion between the team and the PO. Testers often tend to over-test, so reporting on the risks and the testing done helps the team perform the best possible testing, which is just enough testing.

- Make learning part of the way of working

Better refinements are crucial for fostering common understanding, but the PO can do even more to cultivate a learning culture within the team. Here are some ideas to help the team:
Collaborate with the team to create visual overviews of the product from various perspectives, such as architectural models, process diagrams, business

model canvas, flow charts, and mind maps. This helps the team gain a comprehensive understanding of the product. Encourage the team to experiment where appropriate, fostering innovation and learning. Promote feedback loops, retrospectives, and reflection sessions to continuously improve processes. Help the team visualize the value stream, allowing them to identify and pick up improvement stories. Also practices like TDD and BDD help teams learn and create understanding across the team. Finally, proactively collect customer feedback to ensure the product meets users' needs and delivers the right value.

By implementing these strategies, the PO can significantly enhance the team's ability to learn, adapt, and deliver high-quality software.

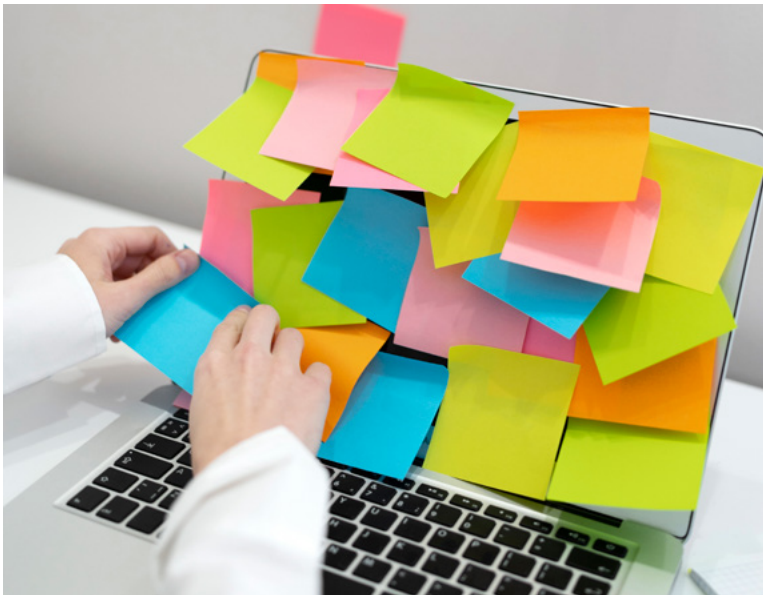
- Mandating quality

Mandating quality is essential for successful software development. Ensure your team actively uses the Definition of Ready to verify they are prepared to start the work and the Definition of Done to confirm that all anticipated tasks are truly completed. Foster a learning environment by making the team a safe space where mistakes are seen as opportunities for growth rather than reasons for blame. In VUCA environments, mistakes are inevitable. Encourage Root Cause Analysis and blameless postmortems when significant bugs are found to help the team learn and prevent future errors. Advocate for code quality and support refactoring, testing, and automation when necessary.

WRAPPING UP

Our clients demand faster, cheaper, and more predictable software delivery. The key to achieving this lies in adopting a "building quality in" mindset. This approach ultimately leads to faster, more cost-effective development, resulting in satisfied clients and happy teams.

POs play a crucial role in Quality Engineering by promoting quality products through effective collaboration. They should encourage risk analysis to help teams gain a deep understanding of the product and identify major pitfalls, leading to better software development and just enough testing.



Additionally, POs should ask the team to report on the product's status, the testing conducted, and the remaining risks. This information enables informed decisions about product shipment, potential process improvements, and enhanced testability.
By fostering a culture of quality and continuous improvement, we can exceed our clients' expectations and deliver exceptional software products.

Huib Schoots
Consultant & Quality Coach
@ Sogeti

Niek van Malsen
Subject Matter Expert in Transformation & Coaching @ Sogeti

Photo by Bernard Spragg. NZ

Little's Law: on ready-for... queues

Author: Antony Marcano & Andy Palmer

When your team introduces a 'ready for...' queue — beware. For example, as developers pull from the backlog faster than testing can be completed, a ready-for-testing queue builds. It feels faster but you're probably adding costs of delay and causing the team to deliver less overall. Little's Law and the Latency Effect, help us understand why...

LITTLE'S LAW

Little's Law is a way to understand the relationship between:

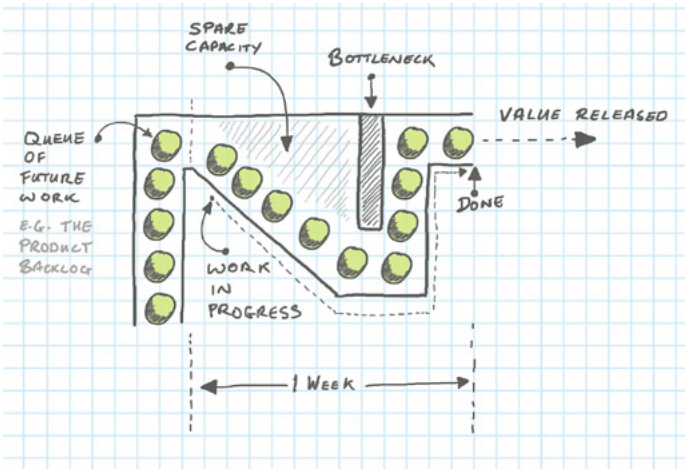
- **Throughput** — how many items of work we can complete each week (e.g. User Stories)
- **Work in progress** — all work that has been started but not finished, even if it's blocked waiting for someone or something.
- **Cycle time** — the time it takes to complete a specific item of work from start to finish (i.e. deployed to production).

Hopp and Spearman's book, Factory Physics, expresses Little's Law as:

$$\text{Cycle Time} = \frac{\text{Work in Progress}}{\text{Throughput}}$$

"Originally published on ideas.riverglide.com"

Consider a team that, at any given moment, has 10 items of work-in-progress. This team completes 10 items of work per week — the team's throughput.



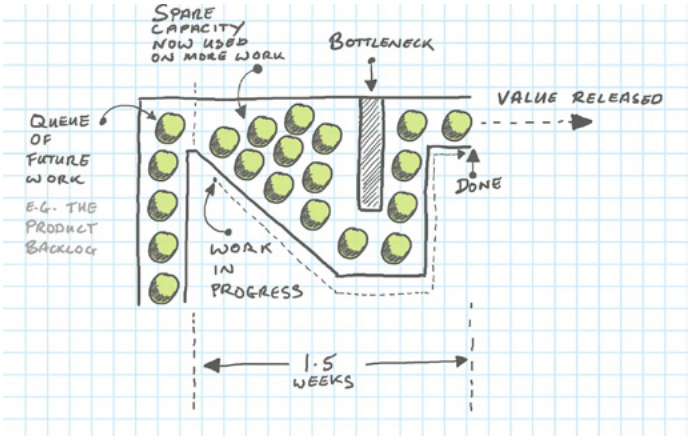
Little's Law predicts a cycle-time, per item of work, of 1 week:

$$10 \div 10 = 1 \text{ week}$$

When developers are able to get through the work faster than testing can be completed, this creates spare capacity among the developers. A developer may then use that available capacity to take the next item off the backlog. The more work the developers get through, the more features we'll have, right? Wrong — but why?

THROUGHPUT

Firstly, this team's throughput is constrained by a bottleneck in their process. Several testing activities are happening at the end of each item of work and can't go any faster, without some investment. Having more work in progress, even in the best-case scenario, won't result in any more being delivered due to this constraint.



WORK-IN-PROGRESS

Meanwhile, the developers are getting through 5 more items per week than can be completed by the testing team.

With 10 items of work in progress with the developers and 5 in progress with the testers, we now have 15 items of work in progress.

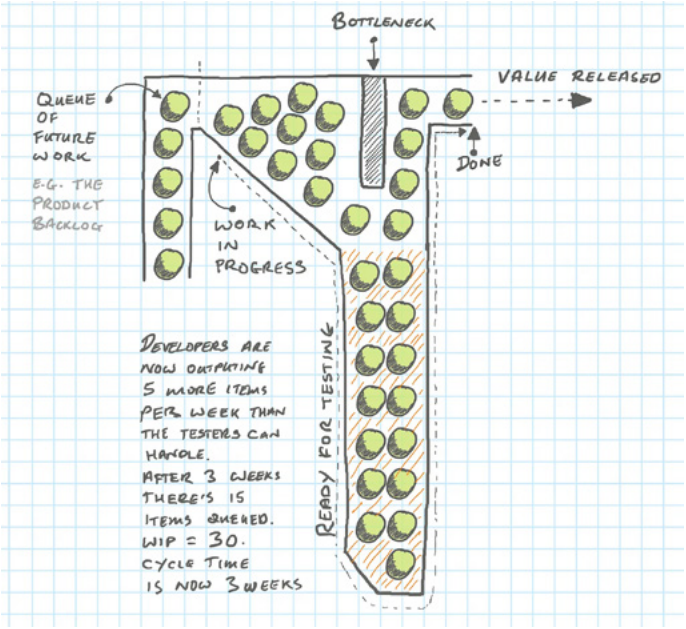
CYCLE TIME

With 15 items of work in progress, and the bottleneck constraining the throughput of the team to 10 items per week, the cycle-time increases — introducing costs of delay. Where each item was taking 1 week before, it's now taking 1.5 weeks from start to finish:

$$15 \div 10 = 1.5 \text{ week}$$

THE READY-FOR-TESTING QUEUE EMERGES

As developers get through more work, testing is still only being completed at the same rate as before. Many teams then introduce a 'ready for testing' queue to allow the developers to continue to work independently of the bottleneck. Why should the developers slow down because the testing can't be completed fast enough? Let's explore...



Each week, the developers add 5 more items to the queue. By the end of the third week, we end up with 30 items of work in progress: 15 queueing up for testing in addition to the other 15 items of work in progress.

This pushes the cycle-time out even more:

$$30 \div 10 = 3 \text{ weeks}$$

Now the cycle time is at 3 weeks per work item. And so the process continues, with ever increasing cycle time.
It is for similar reasons that, in some Scrum teams, we see coding happening in one sprint and testing a sprint behind. The result is that testing falls further and further behind.
To make things worse — throughput likely reduces, resulting in fewer features than before. This is largely due to the Latency Effect...

THE LATENCY EFFECT

As the cycle-time per item of work increases, the feedback loop lengthens. When the team was flowing, the developers were receiving feedback from the testers soon after they completed their work. Now, the latency between the testers providing feedback has increased. The developer has moved on by several backlog items.
The developer needs to reload the context from days, or even weeks, ago. The further in the past this work is, the longer it will take the developer to reload this context to then diagnose the cause of any defect and incorporate any feedback.

WHAT CAN YOU DO?

Instead of pulling more and more work in, what could the team have done to avoid the ready-for-testing queue? Simple, use the spare capacity to address why that queue was needed at all...

Are lots of defects being found, causing the testers to spend more time investigating and reproducing them? Can we get the testers involved earlier? Can any predictable tests be discussed, agreed and automated during development so developers can ensure the code passes those before any end-of-process testing happens?

Are the testers manually regression testing every user story or once per time-box (iteration/sprint)? Could the developers help by automating more of that? Are the testers having to perform repetitive tasks to get to the part of the user-journey they're actually testing? Can that be automated by the developers?

Addressing any of these issues is likely to speed up your testing activities, opening up the bottleneck. The result, is that this increases the throughput of the entire team. Now, taking that next item off the backlog will actually have the desired effect — more features than the team has delivered before.

Arguably it's the handoffs that are at the root of the problem. However, these handoffs are a reality for many teams. Little's Law and the Latency Effect aren't just restricted to teams with handoffs. Regardless of your process, increasing work in progress beyond the throughput that the bottlenecks in your process allow, is a false economy — you'll feel faster while, at best, delivering at the same rate. With the latency effect, you'll deliver less, rather than more.

Antony Marcano
Software Development Leadership Coach & Consultant, Co-Founder, RiverGlide

Andy Palmer
Coach, Author, Developer, Mentor, Advocate, Independent

trendig

your leading technology services provider

Product Vision
Design Sprints

Consulting, Coaching, Training & Expertise Integration for

Agile Testing Days
Community Meetups

Agile Quality Practices

Agile Transformation & Scaling Agile

Requirements & Backlog Refinement

Software Testing (Automation, Mobile, Performance, Security)

Artificial Intelligence (Applications, Data Services, Testing)

trendig.com

3 Lens Quality Coach Model

Author: Anne-Marie Charrett

The Quality Coach model is a product development approach that emphasises quality as a team activity, rather than a role. With the recognition that software testing is a skilled activity that many teams lack proficiency in, quality coaches help these teams improve their testing as well as their overall quality. A quality coach enables quality as opposed to owning quality.

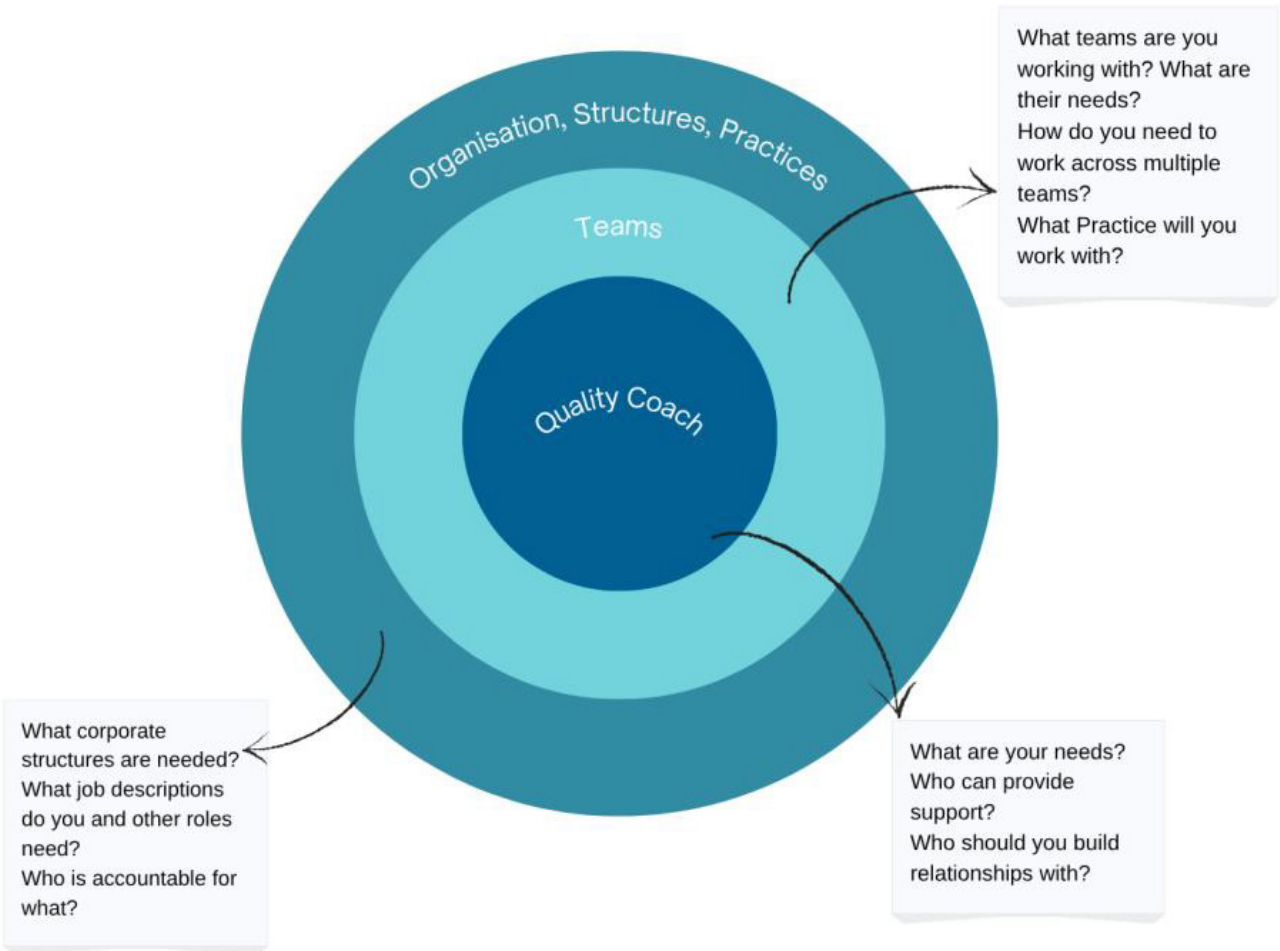


Figure 1 3 Lens Quality Coach Model

The Quality Coach model works well when an organisation views quality through three lenses[1]. They are:

1. **Organisational lens:** These are the structures and models required for the team to exist within an organisation's construct.
2. **Team lens:** At a team level, how a team prevents, detects and recovers. How teams build quality products.
3. **Quality coach lens:** how to coach teams and individuals, how to think about quality,

By framing how quality operates through these three lenses, you ensure you have considered the impact of any changes through the eyes of a quality coach, the teams, and the entire organisation.
There's a significant amount of context to consider. Here are some questions to ask yourself, your team and your organisation. You don't have to answer all of these, but it's worth thinking them through with engineering leadership so they understand the potential impacts this model will have, from hiring to support.

Organisation Structures
The essence of the quality coach model is to shift ownership of quality to everyone in product engineering, rather than allocating it to a single role. This requires more than a quality professional to adapt, but also teams and organisations.

Structure	Considerations
Current Company Structure	What are the Company Values? How big is the company? Maturity of Company (startup, scaleup, enterprise)
Current Company Structure	What is the current company structure? Which other practices will you align with (delivery, SRE, engineering)? What might need modifying to accommodate the quality coach model? Do Quality professionals exist in teams? How will they transition? Team Topologies: Are there stream-aligned teams? Enablement teams? Platform teams? What will you be? How will you structure product focus versus technical focus?
Enablement Team Structure	Will you be solely a technical enablement team? Will you work in isolation with other enablement teams or collaborate with them? How will you maintain frameworks and tooling? How will teams access your frameworks and example tests? What technical tooling will you be delivering?
Practice or Guild	Will you have a guild or a practice where quality coaches can gather and learn from one another? Will they report to the quality practice, or will their manager be within a squad? What is the aim of the practice? What is it accountable for?
Operating models	How will you interact with feature teams? Will you wait for a request? Or will you perform audits and track progress? What other practices will you collaborate with? Do you need an operating model for them? How will the operating models impact roles and responsibilities?

1: The three lenses are a concept from LifeLabs Learning, which provides training and resources for leadership and team development. The lenses help teams and organisations to view their work from different perspectives, ensuring a holistic approach to problem-solving and improvement: lifelabslearning.com

Roles & Responsibilities	Who is responsible for the elements of quality? What roles need modifying? Will software engineers need to have software testing included in their role? How will they transition those roles? What impact does this have on career paths? Career Growth?
Job Descriptions	What will the job descriptions look like? Will software engineering job descriptions include software testing? What other activities and tasks will they be taking on? What HR considerations do you need to make when changing anyone's role?

Process Area	Considerations
Hiring Process	What will the hiring process look like for new engineers and quality coaches?
Onboarding Process	What new information do quality coaches need during onboarding? What do engineers need to learn about owning software testing? What tools do they need to be taught? What self-service product knowledge can be made available to all new hires under the product?
Transition Process	What is the current state? What will be the end state? What does success look like, and how will you track it? How will you get from the current state to the end state? (What's your strategy and roadmap?) What change management tasks do you need to ensure you do? Will you utilise the ADKAR change management model, or does the company have an established method for handling change? Who is currently managing existing quality professionals? Who should manage them?

INTERACTING WITH TEAMS/PRACTICES

Interpersonal considerations are the people and groups with which the Quality Coach model interacts. For example, Quality Coaches, Software Engineers, Engineering Leadership, Product Managers, Designers, Practice Leads. Consider the following questions:

Area	Considerations
Informed	Who wants to know the state of the product's quality? How often should you catch up? How will you keep people informed? Through reports? Videos? How will you share success stories?

Informed	Who will the Quality Coaches report progress to? What information sessions will you need to hold? Who do you need to keep informed about the transition process? Who should be informed about quality improvements? Who needs to keep you informed? What do you want to be informed about?
Consulted	Who do you need to consult to develop structures and processes? Who will be impacted by the quality coach model? Who will want to have input into how quality operates? Who should you consult? Who has influence?
Product Teams	What product teams will you be working with? How many? Are they value stream teams or platform teams? What skills do software engineers have? Do they understand they are responsible for quality? What training will they need? Will you audit teams? Will you coach or direct? How will you decide how to interact with teams? Does the team have a whole-team approach to quality? If not, how do they manage quality? Is it working? What needs improving? Do the product teams know what success looks like? How will you begin coaching them on what good looks like? How will you split up quality ownership among teams? What are they responsible for? What are you responsible for? How will you go about working that out? How will you incorporate and involve product engineering in your decision-making?
Practices	What practices will you be collaborating with? Engineering? SRE? Delivery? Security? How do you plan to work together? How can they build quality into their ways of working? How can you help other practices? What can you collaborate on? What can you learn from them? What tooling are they using? What ways of working, operating models and structures do they use?

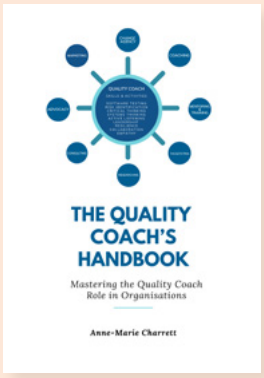
QUALITY PRACTICE STRUCTURE

Quality Coaches may or may not be part of teams. Regardless of where they sit, a centralised practice is required to develop a systematic approach. This will help Quality Coaches have a centralised library of material and tooling, as well as build consistency of approach across teams. Consider the following:

Quality Practice	Considerations
Career Path	How do you begin your career path? Where can it lead? What levels of Quality Coach will exist? What roles and levels can you compare a quality coach role to? Who within the organisation should you involve? Who will manage the quality coaches, engineering managers or the director of quality engineering? What are the reporting lines?
Training Plan	What training needs to be given to quality professionals to shift to the Quality Coach role? What training do the teams need? Who will do the training? Is it internal or external, or both? Do you need an additional budget for training? Who will provide that?
Job Descriptions	What skill sets are required? Will you train up or hire in? What does the job description look like for each level of Quality Coach? Will there be a difference between a Technical Quality Coach and a Product Quality Coach? Will there be a different job description? How can quality coaches switch between the roles?
Quality Coach Practice	What will the team structure look like? Who is responsible for what within the quality practice? What are you responsible for as opposed to what the team is responsible for? Test Environments, Test data, Frameworks. How many quality coaches will you need? Now, and in the future? What's your operating model? How will you interact with teams? How will you measure success?

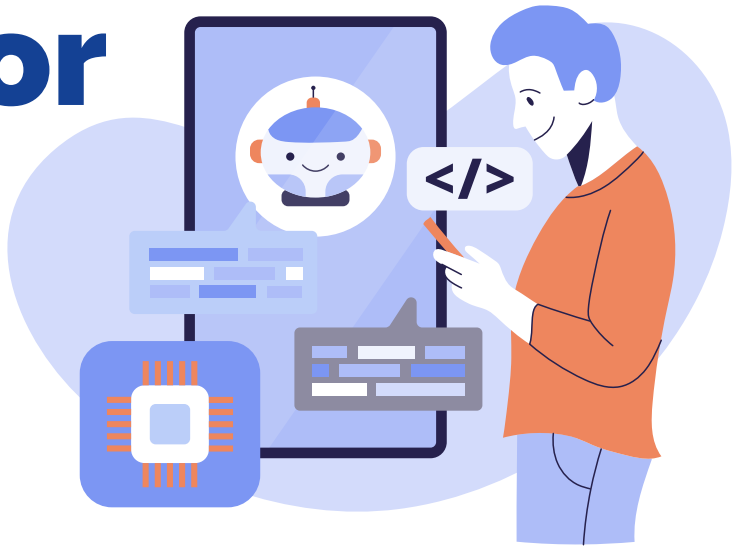
Sensible Defaults	Product Quality End State Quality Attributes for a product and/or a service Key risk areas for the company List of recommended tooling that quality practice supports Test data strategy, test environment strategy, test automation strategy Company-Wide Quality Strategy Quality Rituals & Practices you want teams to embrace (Risk Storming, Contract Testing etc.)
Principles	**Company Principles:** What are the company's principles? How do they impact quality? How will you incorporate them into your quality practice? How will you ensure teams understand them? **Quality Principles:** What are the quality principles? How will you create them? Who should be involved in creating them? How will you ensure they are adopted by teams? **Architecture & Engineering Principles:** What engineering principles will impact quality? How will quality coaches contribute to the conversation? **Product and Design Principles:** What principles in product and design exist? How can you incorporate quality into them?

Anne-Marie Charrett
Consultant
@Testing Times



How I build my own GenAI test data generator

Author: Stephan Dreher



Motivation

Are you familiar with the problem of generating and permuting test data to achieve a large and preferably comprehensive range of data for the software under test?

Testers with developer skills often build test data generators using languages such as Python, which makes this very simple and for which you can find many excellent libraries worldwide.

However, testers without programming skills usually must seek assistance from the development team. I have always been bothered by this in my career. That's the reason I have become a (test) data engineer myself over the years.

Test data generator

I use the *Replit* development platform[4] to create my own test data generator with the help of the AI Assistant. My open-source project is written in Python. You can find it on GitHub[5].

Replit is a tool that lets you develop things in the cloud. It also has a very powerful AI Assistant. Here's a prompt to get you started on

Stephan Dreher
Consultant & Quality Coach
@ ORBIT IT Solutions GmbH

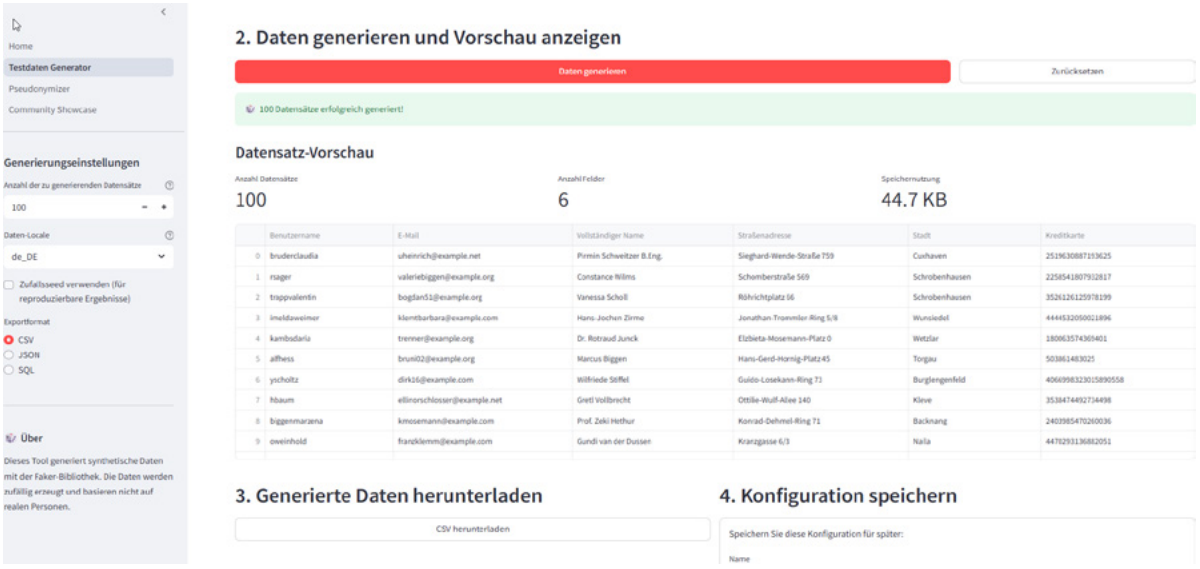
creating the app:

> *"Design a test data generator with a GUI frontend to prompt data specification that could be used for a web portal. Typical data specifications are username, password, street, city, zip code, country and more. User should be able to permutate inputs randomly"*

First results were quite buggy, and it takes a few hours debugging until GUI and features were in the right place.

Features

A test data generator developed in Python has a user interface that enables the specification and random permutation of web portal user data. The Test Data Generator is a tool for creating synthetic test data for web portals and applications. It offers the following main functions:



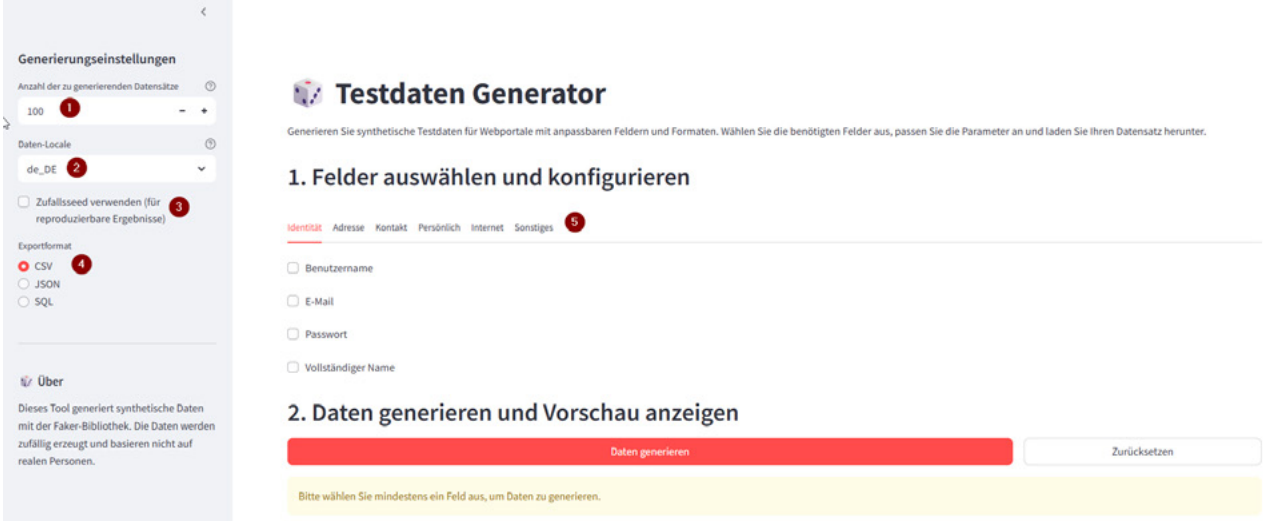
- Flexible data generation with customisable fields and formats
- Field configuration: parameters for each field (e.g. password length, etc.)
- Data export: export generated data as CSV or JSON
- Database connection: configurations for reuse
- Multilingual Support for different formats

The test data generator was developed with the following libraries:

- Streamlit: User interface and application framework
- Pandas: data processing and manipulation
- Faker: Generation of synthetic data
- SQLAlchemy: Database interaction
- PostgreSQL: Persistent storage of configurations

Start generating

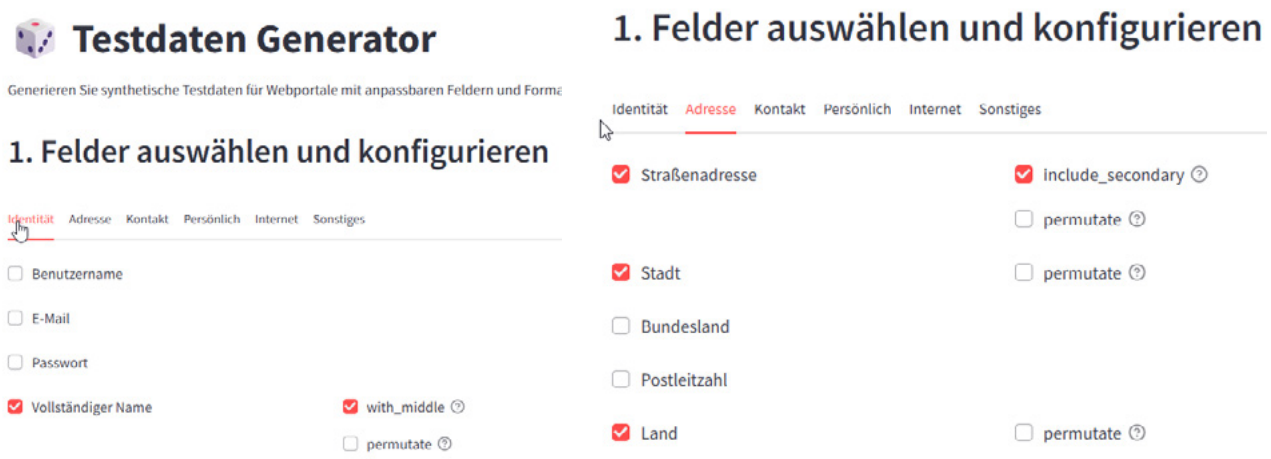
This first prototype is not designed for a fully developed user interface but is merely intended to show how easy it is to create such a tool.



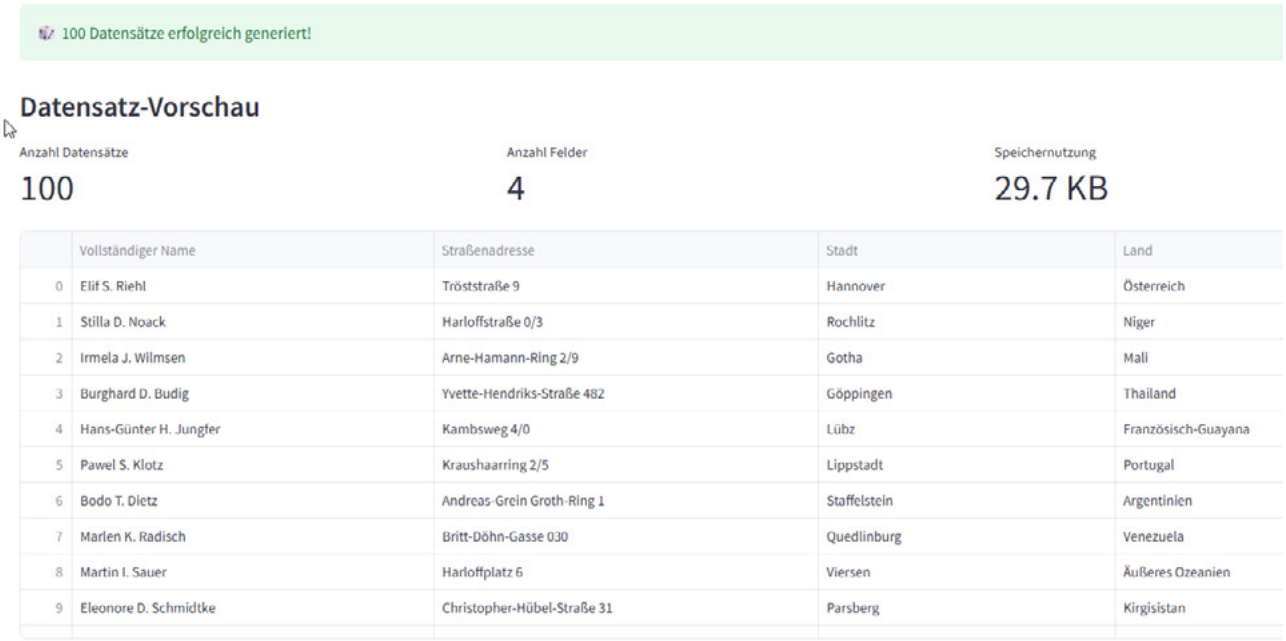
Features:

1. The number of test data records to be generated can be entered here. Currently the maximum is 10000, but this is due to the limitation of the computing power and the underlying DB. 10k is usually sufficient.
2. Different country templates can be used (DE, JP, US, CH, etc.).
3. The random seed helps with the permutation of the data (details at Faker[1]).
4. Export formats are CSV, JSON or SQL.
5. The fields can be selected here. The selection is purely arbitrary and can be customised very easily.
6. To keep it simple, I only generate test data with name and address (sorry GUI still mixed languages DE and EN).

The data is displayed in a preview and can then be downloaded in the selected format.



The selected configuration can also be saved for reuse.



	Vollständiger Name	Straßenadresse	Stadt	Land
0	Elif S. Riehl	Tröststraße 9	Hannover	Österreich
1	Stilla D. Noack	Harloffstraße 0/3	Rochlitz	Niger
2	Irmela J. Wilmsen	Arne-Hamann-Ring 2/9	Gotha	Mali
3	Burghard D. Budig	Yvette-Hendriks-Straße 482	Göppingen	Thailand
4	Hans-Günter H. Jungfer	Kambsweg 4/0	Lübz	Französisch-Guayana
5	Pawel S. Klotz	Kraushaarring 2/5	Lippstadt	Portugal
6	Bodo T. Dietz	Andreas-Grein Groth-Ring 1	Staffelstein	Argentinien
7	Marlen K. Radisch	Britt-Döhn-Gasse 030	Quedlinburg	Venezuela
8	Martin I. Sauer	Harloffplatz 6	Viersen	Äußeres Ozeanien
9	Eleonore D. Schmidtke	Christopher-Hübel-Straße 31	Parsberg	Kirgisistan

Conclusion

Whether you generate test data scripts using *ChatGPT* or other GPTs or even create a web app using *Replit* certainly depends on the situation (time, money, knowledge).

Nevertheless, the new AI assistants offer wide advantages for testers when generating test data. AI assistants, Python libraries such as *Faker*[1] or medical research tools such as *ARX*[7] also help with the anonymisation and design of existing test data sets.

TEST DATA PROTECTION

An important aspect is certainly data protection and compliance with the *GDPR*. The test data to be used should be classified and a strategy should then be considered to avoid GDPR-violations (red).

Here are some tips for making information anonymous or pseudonymous. It is said that the best way to do this is to use test data that has been created by ai (green).

However, anonymised data may no longer be consistent and therefore no longer usable. In this case, compromises in data protection must be compensated for by pseudonymisation measures (yellow).

Links

[1] Faker Library: faker.readthedocs.io/en/master

[2] Vibe coding: de.wikipedia.org/wiki/Vibe_Coding

[3] OpenAi ChatGPT: chatgpt.com

[4] Replit: replit.com

[5] DataGenPrototype: github.com/stdreher/DataGenPrototype

[6] Pytest: docs.pytest.org/en/stable

[7] Werkzeug ARX: arx.deidentifier.org

Option	GDPR Risk	Comment
Production (real) data	Very High	Violates purpose limitation; only lawful with explicit consent (rare in practice).
Pseudonymized real data	High	Still considered personal data; strong safeguards required.
Anonymized real data	Low	Acceptable only if true anonymization is ensured – which is hard to guarantee.
Synthetic test data (artificially generated)	Very Low	Ideal from a privacy standpoint; can mimic real data structure and behavior.
Manually created dummy data	Very Low	Safe, but may lack realism for complex test scenarios.
Masked real data (e.g., hash, placeholders)	Medium to High	Not anonymized; re-identification often still possible.
Public dataset (open data)	Medium	Depends on the source, content, and usage license – legal review recommended.
Data with user consent for testing	Medium	Only valid for specific, limited test purposes; legally burdensome.

Generated with ChatGPT

You often can't use US cloud IDEs like *Replit*[4] directly in projects in Germany because of known compliance reasons. However, apps created with them, which have simple, transparent and traceable code structures and which meet data protection requirements (self-hosted), can be used to generate neutral, anonymous test data.

OPEN SOURCE

I like open-source tools that use the *Python* programming language. *Python* is a powerful language for processing data and has many well-written packages. It's no surprise that Python is the go-to language for many AI projects. The community is also very friendly and active.

VIBE CODING

Vibe coding platforms in Germany are becoming more popular and are also necessary. There is a high demand for so-called lay developers (citizen developers), but this is also a controversial topic in the development community. QA will certainly have to make sure that the quality is good in the future.

AI CAN HELP US, BUT IT CANNOT REPLACE US.

AI assistance will not replace human testers soon, but it will change the profession a lot. All testers need more training.

hands-on training

AiU Certified

GenAI-Assisted Test Engineer

What you'll learn:

- AI-Assisted Testing Introduction
- Prompt Engineering
- Requirements Review
- Test Generation and Optimization
- Test Data Generation
- Bug Advocacy
- Future Possibilities

trendig.com