







# THE TESTING PLANET

**ISSUE 11** 

A MINISTRY OF TESTING PUBLICATION

SEPTEMBER 2024



# 404 HEADLINE NOT FOUND



In a data migration, you move all of the existing data into a new database.

## When Your Data Moves House

#### Practical Testing Tips for Clean Data Migrations

#### BY KONSTANTIN **SAKHCHINSKIY**

Today's software platforms rely on complex databases that contain data from many technical and business entities. When new features are added or legacy code is refactored, current data often is altered and changed, caused by the implementation of new tables and fields in the database and the deletion of old ones.

If your organization's data will be affected in this way, it's likely that a data migration from the old to the new database structure will be necessary. This type of migration can sometimes be more complex and time-consuming than the implementation and testing of the new feature itself.

And that's where you as the software tester come in. Let's discuss data migration testing. I've tested a few different kinds of data migrations and I'll share some of my experience and lessons learned with you,

giving you a basic framework from which you can develop your own approach.

#### WHAT IS DATA MIGRATION TESTING?

Data migration testing is part of a holistic effort that ensures a seamless transition from a legacy system to a new one with minimal disruption and no loss or corruption of data. It verifies that current data as well as new data will be handled correctly by the functional and nonfunctional aspects of your application.

In a data migration, you move all of the existing data (system and user data alike) into a new database. The end goal is for the application to handle both data with the new structure and the current data effectively. So you must ensure that:

Existing data gets to the new structure with no loss or corruption

Legacy and new applications are functioning properly with respect to the "new" database structure (assuming that legacy applications will be in use in production after the migration, which is likely).

#### WHY IS DATA MIGRATION TESTING NECESSARY?

Data is migrated to new systems for various reasons, such as system consolidation, technology obsolescence, new features, or optimization. Key considerations during migration testing are:

- Minimize disruption to customers. Your team should ensure that disruption is minimized in production. You want to avoid any inconvenience to the system's users, whether they be B2C customers or employees of organizations that use the system.
- Ensure feature continuity. Users should be able to access all software

CONTINUES ON PAGE 2

#### Better Software, Faster

A Tester's Report On The Quality Assistance Model

#### BY NATALIIA BURMEI

"Under the Quality Assistance Model, you're always on the lookout for the next challenge. As a tester and engineer, I have a lot of freedom, support, and bandwidth to experiment that I try to use wisely ... You must be vigilant and aware of all of your team's workstreams, and you should spot opportunities to improve and experiment, closing gaps whenever you see them. And finally you learn continuously so that you can adapt when needed."

CONTINUES ON PAGE 4

#### Also in the news...

Lessons Learned From 20 Years Of Software Testing



TURN TO PAGE 9

Quality Coaching: A Road Less Traveled



TURN TO PAGE 20

#### Co-creation undefined



**CONTINUED FROM PAGE 1** 

features with no issues after the migration is complete.

• Comply with applicable laws on data privacy and protection. Consult with your product manager or stakeholders and, if necessary, the organization's legal team to determine if the data migration involves any regulated user data and if additional steps have to be taken. This is crucial for compliance with legal standards and to avoid potential legal issues.

DATA MIGRATION TESTING STRATEGY IN A NUTSHELL

#### **Phases of Data Migration Testing**

Each of these phases should be executed in test before your team attempts a production migration.

- **Pre-migration** audit: Before the migration begins, examine the data in legacy systems and database tables to identify issues such as inconsistencies, duplicates, corruption, or incompleteness. This can prevent complications during the actual migration in production and it helps to prepare realistic test data.
- Backward compatibility testing: Ensure compatibility with existing data

and features.

- **Rollback testing:** Validate the ability to revert to the legacy database if needed.
- **Post-migration data validation:** Confirm that all data is migrated, is in the expected format, and functions as intended in the new environment.

#### Features and performance testing

- Verify that features and systems work with migrated data as expected.
- Test the interfaces between the migrated data in applications and other services they interact with.
- Test performance to ensure that it is on a par with (or faster than) that of the legacy system.

#### **Data Migration Challenges**

- · Handling large datasets
- Ensuring data consistency
- Addressing potential bugs that arise during system migration
- Data in more than one character set
- Migrating data and introducing new features at the same time
- Proper obfuscation of personallyidentifying user data
- "Movability" of a data set that contains obfuscated user data and user IDs

DATA MIGRATION TESTING IN ACTION

Let's consider our strategy in two real-

world cases to get a better understanding of data migration testing.

SCENARIO ONE: A MULTINATIONAL, MIRRORED WEB SERVICE

Imagine a complex web service that's integrated with several different microservices and third-party apps. Think of it as customer support service for a big company.

The system consists of instances across the world: Europe, Africa, Asia, and North, Central, and South America. Each of the instances mirrors the whole setup for a specific region.

Here's the deal: the company wants to save some cash. Running all these separate systems for different regions isn't cheap. We're using more servers, there's a lot of hassle with updates and management, and those third-party apps cost a lot. So, the plan is to bring everything into one instance: one system to rule the data for all the regions. The system has to be able to handle all these regions without any issues.

Clearly, this is a big challenge. We can't just mash all the data together. Data from each region needs to stay in its lane. It's crucial that the system knows where each piece of data came from and keeps it all sorted according to that region's specific rules and regulations. This way, even though it's all running on one system, each region's data is still playing by its own rules.

#### **How I Would Test This Scenario**

Use a mix of real-world data and generated test data. How do you know which type to use? Well, in my case, I used real-world data whenever possible, but I generated data if there was a particular test case that wouldn't have been covered otherwise.

We can't just mash all the data together. Data from each region needs to stay in its lane.

- Using real-world data also provided a lot of volume, meaning that performance would be assessed realistically.
- Finally, I avoided the extensive effort required to inspect real data just so that I could generate synthetic data, making the process more efficient.

Use several staging environments (test environments) to emulate different regions.

**Make a copy of your production data.** In my case, I was able to use an API to get





As a 25 year veteran of testing, I love new testers who challenge the status quo. It's so easy to get stuck in it's what it is.

#### Co-creating better testing



all the needed data from the production environment. The system was integrated with a third-party solution (Zendesk), so I used Zendesk REST APIs to get all the needed data without the need for a ton of custom SQL queries and unwieldy database dumps.

Obfuscate or remove all personally identifying data. This isn't an easy task for the most part. You may need to consult with the product owner or stakeholders to clarify which data identifies people. That data should be removed or obfuscated to avoid potential legal issues.

- You will have to determine whether to remove or obfuscate data. For example, if you have usernames and email addresses, you can't simply delete them and leave the fields empty. Instead, you would delete the original data and replace it with randomly generated names and emails based on these names.
- The good news: these tasks can be automated.

Generate synthetic test data where needed. In some cases, it might not be necessary, especially if the entities in your databases are simple and few. But you might need synthetic data to cover specific test cases. For example, a customer support service database that includes the names of company products might have a lot of data connected to Adobe product

names like "Photoshop" and "Illustrator". There is no need to spend time on getting this data from the production database, but you still need those names because many other entities will be connected to them.

Verify that all data sets are correctly represented. Implement data validation that compares source and target data sets for consistency. Depending on your data sets, you may need to do this manually or use scripts that will check selected characteristics of data.

**Add data to each staging environment.** If you have an API to do this, as I did (Zendesk), it makes this task easier.

- The challenge here is to match staging IDs with production IDs because, obviously, many instances can't be directly migrated. If you use, for example, a table with users' actions on the site, which uses user UUID as the primary key, then the user UUID must be matched with other tables and services where these UUIDs are also used.
- Without this matching, you'll get inconsistencies, and some system functionality may not work at all.
   And because of the large number of integrations with different services, it's impossible simply to get a dump and use it as is.

**Perform static and functional testing on each test environment.** Check that the data looks fine, everything works, and there are no crashes on each staging separately.

**Test the migration itself.** This is the test of the big production migration. Showtime!

- Run all of the region-specific migrations (data from different staging environments to a single new environment).
- Check that everything works with "old" (migrated) data and with new data, ensuring no conflicts, issues, and no crashes.
- Check that everything works with new fresh data (created using this new realm).
- Ensure that data from different regions (Europe, Asia) still belongs to its original region, no data was lost, and all data is accessible and can be changed, deleted, or updated according to the application logic.
- Test the rest of the features: full regression testing.
- If any step fails, then go back, make fixes, and repeat. When I ran into an invalid migration, I already had all the scripts for data generation, creation, and obfuscation, so it wasn't a problem to wipe the database of the new staging and retry migrations with fixes.

SCENARIO TWO: A CLOUD-BASED TOOL FOR BROWSER EXTENSION MANAGEMENT

In this scenario, we have an application that manages several browsers and their settings on your laptop or desktop, similar to an antidetect browser. My focus here will be on browser extensions.

Each browser has a unique set of extensions. Some of these extensions may store users' data, and some extensions may run in multiple browsers. Our application allows you to handle extensions across browsers no matter where you have it installed. These extensions are stored in the cloud along with their configuration information, for example, which extensions are installed on each browser profile.

The product owner wants to enhance this extension management tool, introducing new features and more options for installing extensions from various sources such as the Chrome Web Store, .crx or .zip files, or unpacked extensions. The end user should be able to install extensions without the need to open the browsers for each installation. The end user will also be able to delete extensions or use different (older) versions of the extensions.

These improvements will require significant changes to the database structure. As a result, existing data on the extensions and versions installed on each

**CONTINUES ON PAGE 4** 



Build credibility and confidence in

# **Test Automation**

Foundation Certificate in Test Automation Intermediate Certificate in Test Automation Advanced Certificate in Test Automation

For more information visit:

ministryoftesting.com/certifications

#### Testing the boundaries

#### **CONTINUED FROM PAGE 3**

browser instance will need to be migrated. How will we know the migration was successful? Clearly, we do not want to lose any user data. All extensions and their configuration information, including extensions that aren't available in the Chrome Web Store, should be available and working after the migration. And new features, such as the ability to delete extensions, should work on existing extensions.

#### **How I Would Test This Scenario**

The main approach is similar to the first example, but due to specific requirements, I had to change some steps in my approach.

Create a test environment that is identical to the planned production environment.

Get a copy of the production database and inspect the data. This approach is different from using obfuscated data in the first example, where I had access to some of that data through application functionality (API), and the data was matched with staging data and entities. When you use a full production data dump, obfuscate it, and perform migration tests, it's not about making the data functional within the application. It's about ensuring that the migration commands and logic work correctly. This is a technique that developers can use when they write migration scripts.

#### Generate some test data that covers most of the known cases.

- · Use real data as a source of statistics and particular details.
- · Add different extensions and their versions, including the popular ones, the complex ones, those with different permissions, and so on.
- Use different ways to add these extensions, such as Web Store, .crx files, and unpacked extensions.
- It is important to have a test environment that is identical to the production environment at this point.

Test backward compatibility. Update the server code and test that the legacy client application works properly. Make sure that all user data is available and that the application works with it as expected. In this phase, you will get three sets of similar data:

- Data from step one that wasn't used in
- Data from step one with which some tests were performed (updated, deleted, and so on)
- New data that was created in this step

Test the new client application and periodically check backward compatibility during the testing to make sure that the data created with the new client application was properly handled by the old client application. Also: test new features and conduct a smoke test and regression test of the existing features that were affected.

#### TO WRAP UP

Two examples demonstrate that the phases of data migration testing are in general similar no matter what your scenario, but each migration will be unique in certain wavs.

The key to successful data migrations is the quality of the test data. Whenever possible, use real data (obfuscated). Use real data, and statistics as a reference to ensure all test cases are covered and to generate robust test data sets.

Test backward compatibility, which may not always be necessary but often adds many additional tests and test data generation.

Test backward compatibility, which may not always be necessary but often adds many additional tests and test data generation.

The data migration should not cause any loss or corruption of users' data, or change it in a way that it cannot be used as it was before. Test to eliminate this risk.

You need some skill in working with databases and scripting.

- I used Python: I had some programming experience, but I used functions in my scripts, not the entire set of objectoriented programming structures.
- Among the tools I used: PyCharm for development along with the Zendesk REST API library. I used many Python libraries such as requests, uuid, names, random, json.
- I also worked with databases, wrote SQL queries, and used tools like PgAdmin and Postman for debugging request and response flows.

Make sure that the entire team understands and accepts your plan for testing the data migration.

Do you have some data migration testing coming up? Have you tested a migration recently? Tell us about it in the Ministry of Testing Club! ■



Under the Quality Assistance Model, you're always on the lookout for the next challenge

#### CONTINUED FROM PAGE 1

The pace of technology development speeds up every day. To cope, companies strive for excellence in every aspect, but most likely their main goal is speed and time to market. You can use the best technology available, hire the best people, and have an amazing barista serving vou morning coffee, but if you don't figure out how those things work together effectively, your business is unlikely to succeed.

When I joined my current company as a test engineer, our team followed an established quality assurance process. However, product delivery speed needed to keep pace with the company's growing revenues. As a result, the tried-and-true quality assurance process the company used was showing signs of wear. And we testers in particular needed to re-evaluate the way we worked.

Thankfully, during that evaluation, we came upon the Quality Assistance Model, which supports business to scale and keeps quality high without the need to hire more test engineers.

#### WHAT IS THE QUALITY ASSISTANCE MODEL?

As originally defined by Atlassian, the Quality Assistance Model promotes the ownership of quality and testing by every engineer on the team, regardless of their role. This allows each engineer to develop code, test it, and release it to production without gatekeeping by a quality team or quality engineer.

Quality Assistance Model, you use your and achieve reasonable goals, identify

testing expertise to guide and educate the team, empowering and influencing them to produce high quality software.

and Sharing quality testing responsibilities across the team spreads the testing workload more evenly and gives testers time to:

- Build testing infrastructure
- Fill gaps in test automation
- Explore new tools and techniques
- Monitor production performance
- Conduct exploratory testing, ensuring sapient investigation of your product that automated tests cannot provide
- Support developers on their team, or even on other teams

#### FOLLOWING A PHASED APPROACH TO QUALITY ASSISTANCE

As a company, we wanted to be able to release any important front-end or back-end work to production multiple times daily. The schedule for native app release, in contrast, was weekly. Everyone understood the goal, but they also needed to know what their role was and how to play it. Ultimately, our adoption of the Quality Assurance Model was not so much a transformation as it was a mindset shift, in which we committed to owning quality collectively while ensuring the quality bar remained high.

We carefully thought through the transition plan and divided it into three phases. We knew that we would not succeed if we transitioned to a completely new way of working overnight. We Then what do testers do? Under the needed a phased approach to define



We have to eliminate this idea of proving ourselves, particularly for women.



emergent problems, and develop solutions. And we developed tools and metrics to measure success at the end of each phase.

#### **Modelling A Troubled QA Workflow**

In the development lifecycle we sought to change, development and test were out of sync. The critical symptom: mountains of open tickets in the Test column in Jira. To add to the testing bottlenecks that plagued our team, testers were releasing code to production at the same time that developers were adding new features at the other end of the process. The result was an inefficient ping-pong game, and the whole team was frequently stressed out.

#### The Transition Begins: Implementing The First Phase

In Phase One of our implementation of the Quality Assistance Model, we sought to replace old, suboptimal practices with new ones. For example, we didn't have a specific place for testing in Jira, so when exactly testing should occur was anybody's guess. We needed to establish clear, easyto-follow processes to ensure that testing was happening in the right place and at the right time. We decided on these objectives for Phase One:

 Establish example mapping, feature kickoffs, and feature demos as new practices

- Ensure that teams were consistently practising new techniques
- Ensure that tickets were not getting stuck in the Jira Test column

Example mapping. In the Jira Test column, activity was divided into two parts. In the first part, teams were to run an example mapping session (similar to a Three Amigos session) before development began. These sessions would help ensure that everyone understood the requirements, the unknowns, and testing that needed to be done. The tester on each team planned the sessions, working with the product manager and designer to make sure all requirements were captured, and then running the session themselves

...testers were releasing code to production at the same time that developers were adding new features at the other end of the process.

or taking notes. At the end of the session, the tester led a discussion about testing, writing down all examples and suggested test scenarios. At the end of the session, the team would have a good deal of confidence about what to do next, what to build, and how to test it. Our aim was to reduce or eliminate ambiguity.

Feature kickoffs and feature demos. The right side of the Jira Test column represented feature kickoffs and feature demos. Feature kickoff is a short catchup between developer and tester before development begins to walk through final ticket requirements. This ensures everyone understands the requirement thoroughly. Feature demo occurs when the developer is done building the feature, to ensure the ticket is done as expected. Initially, the tester conducted both kickoffs and demos, but as Phase One continued, the tester trained developers on how to run the sessions with their peers so as not to create a silo. In this way, the process would scale up easier and teams did not depend on the tester being available to get important work done. This practice embodied the goal of ownership of quality by everyone, not just the testers.

Phase One was potentially the most challenging. The teams were still doing some of their old process tasks while adopting new ones. Testing was still mainly done by testers as the team slowly worked to shift ownership of testing to the whole team. The beginning of a shift in mindset can be the most difficult for many people.

Phase Two: Redistributing Testing Work And Changing The Role Of Testers

In Phase Two, we continued to establish new practices:

- Developers would write automated testing for new features and bug fixes, not just unit tests
- The whole team, not just testers, owned the process of releasing to production
- Testers helped keep focus on how testing is done; stories were rarely assigned to testers

Developers write automated tests for new features and bug fixes. In our previous development lifecycle, developers wrote unit tests, which are and continue to be a pivotal part of feature development. But unit tests by themselves are not enough to ensure that a feature is ready to go to production. And if we wait for testers to build out automation on their own, more bottlenecks emerge. So our new model sought to guarantee that developers wrote as many TYPES of tests as needed to ensure uneventful releases to production.

Developers take part in production releases. In our old way of doing things, testers released code to production while developers worked on new features. This ensured an ever-growing disconnect between developers and testers. In Phase Two, we moved to joint ownership of releases to production. To guarantee joint

CONTINUES ON PAGE 6



#### Beyond the Boundaries

#### **CONTINUED FROM PAGE 5**

ownership, we added to the definition of "done" for each feature a requirement that the code be moved to master AND released to production.

Testers educate the team on how testing should be done and support testing efforts. Meanwhile, testers worked with developers and the team on how testing should be done for individual features while also evaluating how testing was done overall. Testers did take on some of the work to bridge gaps in test automation and to help developers increase their test coverage. Happily, none of this required the addition of more testers to the team! Sharing testing responsibilities ensured that we moved fast and efficiently and that the entire team knew a lot about good testing practices.

How Phase Two went: Asking developers to write integration and UI tests was not a quick switch. We often debated over responsibilities and ownership. We all knew that delivery speed might be a bit slower due to the process of increasing automation coverage, but we all knew it would help the team in the future.

At the end of Phase Two, our team was far more aligned on process and goals than we were before. Development and testing were in sync, more bugs were caught before production releases, and test coverage was well on the rise, building our confidence that regression defects wouldn't show up in production. Communication and collaboration were greatly improved, and we had a broader base of common knowledge.

## Phase Three: Fortifying Our New Practices And Establishing A Culture Of Learning

In Phase Three we sought to build on our progress in Phase Two:

- · Developers owned test automation
- Developers assisted other developers with review, instead of relying on testers to do it
- Testers took on the task of monitoring production

Developers own test automation. In Phase Two, developers began to write tests for features and bug fixes. In Phase Three, they began to take on the ownership of test automation and its maintenance. Owning test automation means that you are accountable for all levels of testing for features you develop. That includes both your own code and potential impacts to other code in the project. Developers were called on to broaden not only their technical skills, but also the awareness of business logic and full accountability for the team's deliverables, not just their own.

Developers assist other developers with review. While testers were often available to support the review of tests the developers wrote, developers were to assume that role as often as possible so as not to create bottlenecks.

Testers tackle production monitoring. You might be asking: where did QA fit in after the developers took on so much work? Apart from working with developers on daily challenges, the big task for testers was to build production monitoring. At that time, we didn't have a cohesive approach on how to monitor the impact of released features, nor did we know enough how to build systems that could uncover defects due to behaviours that were not easy to spot during testing.

Since then we have created a good level of monitoring and observability. However, I think we still have to figure out a way to make it consistent for each team, and find an optimal way to get quick feedback on not only critical features, but also for other features we can learn more about.

#### HOW DID WE DO?

At the end of each phase, the teams measured their successes and failures. Each phase lasted at least three months to ensure we were comfortable with the new processes and that they were working as intended.

Testers' role in the transformation was huge. Not only did they drive the changes in each phase, but they also coached developers on approaching testing earlier in the cycle. Clearly defined objectives for each phase helped testers to know where they were heading.

Acting as influencers was challenging. Imagine having to shape the thinking and practices of others after years of following an established process, all the while making sure that delivery speed and quality remained intact. Happily, we testers had a lot of support and encouragement from our team members. Many people volunteered to facilitate meetings and take on new responsibilities so that testers could trim their overflowing task lists.

These were challenging times, but I can say that I grew as a tester, as a team member, and as a person.

#### HOW TESTERS AT MY COMPANY WORK NOW

Under the Quality Assistance Model, you're always on the lookout for the next challenge. As a tester and engineer, I have a lot of freedom, support, and bandwidth to experiment that I try to use wisely. I use my observational, analytical, and technical skills to discover where the next solution for a problem can bring impact and value not only in my squad, test discipline or technology, but also across the business. It's not an easy task. It means to face a lot of ambiguity, experimentation, failures and sometimes giving up on ideas that don't work.

My job has three main focuses: daily

support for my team; leading the test engineering discipline; and mentoring junior engineers. I have to look ahead to the future while keeping a close eye on the work of the day ahead of me. On any given day, I act as an expert tester, an analyst, a product manager, or DevOps. You must be vigilant and aware of all of your team's workstreams, and you should spot opportunities to improve and experiment, closing gaps whenever you see them. And finally you learn continuously so that you can adapt when needed.

Not surprisingly, we rebranded! We testers used to be referred to as "Quality Assurance," but we are now known as "Test Engineering." And we're always thinking of how the role should change and grow.

#### TO WRAP UP...

Any initiative that requires people to change the way we work is difficult. You are asking people to change their behaviour and habits and to alter their expectations. Not everyone likes to change. Even fewer people like to change quickly.

The transition took us a long time, possibly longer than we were expecting. But the model serves us well today, since we move fast, fix forward, and reiterate reliably. It also gives us room to fail safely, because we have tools and support to identify failures and fix them quickly. As a reflection of our success in establishing efficient, reliable processes, we have maintained a ratio of eight developers to one test engineer in the team.

The Quality Assistance Model won't work for everyone. It requires not only technical skills to be able to navigate different stacks, but also the agility and open-mindedness to reinvent your role. You have to be proactive and disciplined. It won't work for engineers who want a lot of guidance and who don't like ambiguity. But a team whose members are willing to grow and change can benefit greatly by adopting the Quality Assistance Model.

# I put the in state.



#### Dear Agony Ant...

I keep letting bugs go live on purpose. What should I do?

Scan the QR code for answers



THE TESTING PLANET, ISSUE 11, SEPTEMBER 2024

Let's go places!





It's not just about refining bugs, it's about working as one team to create quality software, and to have fun doing so!

# A Guide to Bug Refinement in Software Testing

#### Streamlining Your Workflow

#### BY MEG MACKAY

"The most heartwarming moment for me was when one of our regular attendees said that the bug refinement session is their favourite meeting of the week. It's not just about refining bugs, it's about working as one team to create quality software, and to have fun doing so!"

When I joined my current company, there were more bugs in the backlog than anyone could keep track of. Everyone knew there was a bug problem, but they didn't know where to start. To top it off, many folks joined and left the development team over a short period of time. We saw changes in many key positions, such as our VP Product, Principal Developer, Principal QA Engineer and Tech Lead. To add to the confusion, the core development team shrank greatly in size, meaning some of us had to wear many new hats.

As a new team member and a tester, my

goal was to support the team in improving the quality of the product. But I could do this only if we figured out how to work as the team we had now become.

We'd had regular bug triage meetings

The QA representative
helps the group
understand the context
of each bug, expanding
on the reproduction steps
and test data setup and
highlighting the impact
of the bug and associated
risks to the customer
and business.

in the past, but nobody seemed eager to resurrect these sessions, for various reasons. However, I noticed that the team felt that feature refinement sessions helped us deliver higher quality work on the code for user stories. So we thought of applying that approach to bug triage meetings. And this was the way that the bug refinement session was born!

#### WHAT IS A BUG REFINEMENT SESSION?

During a bug refinement session, we aim to create prioritised, workable bug tickets. At a minimum, newly filed bugs are reviewed, replicated, refined, and then prioritised. After each bug goes through the refinement process, it is assigned to a specific developer to fix or it is dropped into the backlog for a team member to pick up at a later date. Using this approach, bug tickets are treated with the same love as a feature task, with clear acceptance criteria defined for the fix.

We usually hold these sessions at least

once a week, sometimes more frequently depending on project need. The structure of the meetings may vary; sometimes we use the session to review the validity of the bugs in the backlog. For releases that were buggier than normal, we'll use the session primarily for firefighting. The goal is always the same though: curate a high quality bug backlog containing workable tasks with measurable outputs any one of which can be picked up and easily worked on by the development team.

#### WHO SHOULD ATTEND BUG REFINEMENT SESSIONS?

Typically, the bug refinement session includes people in the following roles:

- Project manager
- · QA representative
- Developer representative

The project manager advocates for customer needs and ensures that the proposed fix to any bug will satisfy these needs. They provide important information on prioritisation, acting as a voice for the business, and they assist with finding the most suitable person to work on the bug fix.

The QA representative helps the group understand the context of each bug, expanding on the reproduction steps and test data setup and highlighting the impact of the bug and associated risks to the customer and business. They also contribute to discussions about the technical approach taken to fix the bug and how it will be tested.

The developer representative brings their technical perspective to the table, assessing how much effort it may take to fix the bug and highlighting any dependencies which might affect the work. They participate in root cause analysis by doing a deep dive into the code and investigating what led to the defect.

Some companies include a representative from their Customer Success team in the meeting as well. They bring a fresh, customer-focused perspective, as well as a deep working knowledge of the system, which helps keep the meetings focused on the real goal: building a better quality product for the customer.

Our company has seen a huge difference in the quality of the fixes we provide because of the diverse mix of people who attend the meeting. The most striking difference is that the developers now understand the impact to the customer of the choices they make during the development process.

#### HOW DOES A TYPICAL BUG REFINEMENT SESSION GO?

For bug refinement sessions to be productive, the bug backlog has to be well

CONTINUES ON PAGE 8



Collective success is the thing that is valuable and the thing we reward.

**DREW PONTIKIS** 

77



#### CONTINUED FROM PAGE 7

organised. If your team is just starting out with refinement sessions, the focus should be on clearing the bug backlog, closing old bugs which are no longer relevant due to system changes, and rejecting tasks that lack appropriate details such as full reproduction steps.

After the existing bug tickets meet the new quality standards, a typical meeting flow might be:

Triage of urgent new bugs. First off, attendees should be asked if they have brought any urgent bugs with them to the meeting. These will typically be bugs with a high enough priority and severity to warrant being pulled into the next release. The context around the bug will be confirmed, the group will agree (or not) that it is an urgent bug, then a full reproduction will be performed and the success criteria for a fix will be agreed upon.

Bug verification and root cause **analysis.** In this phase, the team determines whether a bug is a legitimate bug, or whether it is due to user error, bad data, or some other sort of misunderstanding. This quick assessment should be led by product management and QA, who have more experience in this area. Once the team confirms that the bug is legitimate, one or more of the code-literate attendees could try to track down the root cause in the product code. While they do that, someone else may have a browser open with developer tools loaded, to inspect network requests for potential data errors, or use tools to investigate frontend components. They may delve into the system logs to find any useful error messaging. The aim is to quickly identify the root cause of the issue.

Selection of technical approach. If root cause analysis is successful, the team then discusses the appropriate technical approach to correct the error. Sometimes it might just be a one-line fix in a stored procedure, sometimes it requires a full overhaul of the way a whole feature will work. All team members participate in this discussion whether or not they are fulltime coders. If the team cannot decide on a solution, all data from the investigation is recorded in the task, as well as any additional speculation on root cause, to aid the person

Bug refinement sessions can help build relationships among people no matter what their role at the organisation.

who eventually picks up the work. Should a technical approach be agreed upon, that too must be recorded where the developer can find it easily (Jira story, for example).

**Prioritisation.** Your company might use a different methodology for prioritisation than ours does, but the common elements are that there is a clear agreement from the attendees on how soon the fix should be undertaken based on risk, the impact to the business and customer of the bug's existence, and the importance of the fix related to the current work in progress. This will be noted down on the task somewhere using an agreed notation, such as high / medium / low or Po-P<sub>5</sub>.

of the company and their development processes, the bug may get added to a backlog for a single team or a cross-team effort. Or, it may get assigned out directly to the developer who has the most appropriate skillset and / or domain experience.

Cover all unrefined tickets. Unrefined tickets should be ordered from newest to oldest, and the above process should be repeated to cover any new bugs raised since the last session. This provides a high quality, workable bug backlog from which developers can choose tasks as their schedules and workload permit.

#### WHAT ARE THE BENEFITS OF BUG REFINEMENT?

Wherever a company is in their development lifecycle journey, running bug refinement sessions can bring a number of benefits. For less organised teams, it can be a great way to kickstart the collaboration necessary for further improvements. For more established departments, it can add another layer of efficiency to existing processes. It can even just be a great way to get to know various members of the team better, learning how they think and who they are as people.

#### **EFFICIENCY AND SPEED**

Bug refinement sessions align well with the popular Shift Left approach, since they bring the debugging, scoping, and design forward in the process. This approach can prevent the rework entailed by waterfall development processes, where work gets passed back and forth from developer to QA, often with communication limited to what's in the bug tracker. And, just as a refinement session roots out missing requirements in a user story, a bug refinement session highlights areas of impact that might otherwise go unnoted.

Developers can get frustrated because they spend time investigating a bug task they've been assigned, only to find out that it wasn't a bug at all and instead was a data or configuration issue in a test environment. A good bug refinement session results in tasks like this being rejected before they can take the developer's time. It's a great way to ensure that only relevant and workable tasks are on the backlog.

#### **RELATIONSHIP BUILDING**

Bug refinement sessions can help build relationships among people no matter what their role at the organisation. By working closely with colleagues from different disciplines, you get to develop a real sense of respect and understanding which might not have been there before, and this helps you work together more effectively. These strong relationships lead to benefits outside of the meetings, **Assignment.** Depending on the size customer success agents collaborating on to have fun doing so! ■

their work and sometimes even becoming

#### SHARED UNDERSTANDING

Details can often get lost when work passes from a person in one role to a person in another. Working together on tasks from the beginning can cement a shared understanding of the domain's language and concepts that may not have been there before. This means that all involved parties can easily communicate about the work, ensuring the understanding of the work is retained all the way from the customer raising an issue to the fix being deployed. This can lead to fewer instances where a fix is pushed out, only to find that the customer is still having issues.

#### CHALLENGES

It's not all smooth sailing when a group tries to introduce new processes in an organisation. As testers, we're all probably familiar with the common pushbacks of "We just don't have time for that" or "We're already sick of meetings, how will yet another one help?'

One challenge we had at our company was that the benefits of the bug refinement sessions weren't immediately obvious to the leadership team. They ended up frequently questioning their value, leading to reduced attendance from members of their departments. Clear communication of the meetings' output via clear and reliable reporting can be helpful in this situation, as can being curious about what is fuelling any resistance to the meetings. Sometimes, those who object are having trouble understanding something... but since you don't know what you don't know, they can't articulate what that is.

Another challenge can be that repeated attendance at any type of meeting, bug refinement sessions included, can lead to people falling into a rut, getting bored, or just going through the motions. At this point, it's suggested to rotate your line-up to give folks a break, meaning you'll get some fresh points of view and more enthusiastic participation.

#### TO WRAP UP

Creating high-quality, workable bug tickets brings tangible benefits to engineering departments, ranging from better relationships between colleagues and departments to faster fixes being pushed out to customers. It also creates a rare space for folks from different disciplines to explore new perspectives and learn from

The most heartwarming moment for me was when one of our regular attendees said that the bug refinement session was their favourite meeting of the week. So it's not just about refining bugs, it's about working where you may see developers, testers, and as one team to create quality software, and

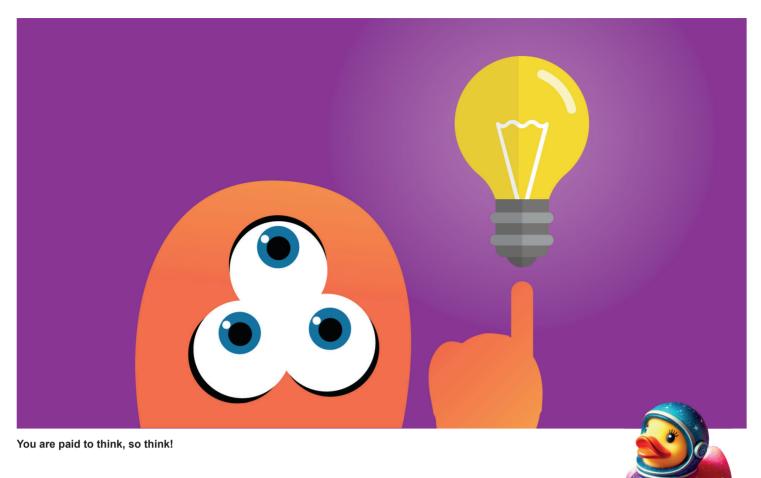


Users don't care about test cases, or plans, or bugs found, they care about the app providing a useful and desirable experience. **KULAS ANGELES** 

THE TESTING PLANET, ISSUE 11, SEPTEMBER 2024

#### Works on our machine





# Lessons Learned From **20** Years Of Software Testing

Explore the personal journey and collective wisdom of a seasoned software tester

#### BY ADY **STOKES**

"Assume the best of your colleagues. If you come across something that's less than perfect, remember that they, like you, are human and don't have all possible information available to them at all times. We can strive for quality and be kind at the same time."

Istarted out in software testing as a 37-year-old with experience ranging from farm work to driving wagons. Along the way, I audited British Standards in Quality, Health and Safety and Environment. Little did I know that some twenty years later I would have found a career in testing, rather than just another job. I'm not saying I didn't enjoy those other jobs, but I'd never really thought of myself as having a career until I met the wonderful testing community. Ministry of Testing in particular, opened up to me a

world of brilliant humans and knowledge I might otherwise never have found.

So here, in no particular order, is a collection of lessons I have learned through those twenty (mostly) wonderful years. I invite you to consider, ignore, argue against, or agree with them. If you have your own lessons, please share them on the Ministry of Testing Club. I'd love to hear them.

#### AND NOW, THE LESSONS...

#### It Depends: Context, As Always, Is Key

"Well it depends on...", is the answer to every testing question ever asked. I say that because context is the key to understanding, and every testing scenario needs a context to help it make sense.

Consider the list of questions below and note that every answer you think of will contain assumptions, since they are asked without context. So how do I test a log-in page, not how do I test a specific log-in page? How do I test a log-in page?

- It depends on whether you are on a desktop or mobile device.
- It depends on whether you use a mouse, a keyboard, a touch screen, or a screen reader.
- It depends on which operating system you're running.
- It depends on the fields present on the log-in page and, if any are present, what you can or cannot enter into them.
- It depends on the information the URL gives you.
- It depends on what happens if you click submit without filling anything in.

I could go on but hopefully you get the point.

If you keep the principle "it depends" in mind, you'll ask further questions. You'll find yourself digging deeper and questioning your initial reactions, biases,

and assumptions.

So remind yourself that whatever the question, scenario or situation, understanding the context in which it arises is of the utmost importance.

#### **Continuous Learning Is Vital**

You can learn anything from anyone and from everywhere you go, and everything you see or hear. No matter your experience in testing, you always have something to learn. Sometimes it will be a new technique, and sometimes it will be more in-depth knowledge. Every encounter, every new person and situation you encounter, old or new, can teach you something. At every conference I've been to, I've learned as much from the conversations I've had with people I might otherwise never have interacted with as I did from the conference talks and activities themselves.

Just like we can't test all the things, we can't know all the things either. This is where continuous learning comes in and helps you gather more knowledge as you go, helping your testing, your career, and your future. Through continuous learning I not only have deep knowledge of testing, Agile methodologies and digital accessibility, but I also gained some knowledge in other areas, from performance to security. That breadth of learning has helped me in so many ways over the years, particularly in creating a learning path for apprentice testers.

#### You Are Paid To Think, So Think!

It took me most of my first decade of testing to understand what a thought worker really was. We all take thinking for granted 99 percent of the time. It's like breathing.

But when you are paid to think, you

ou can learn anything from anyone and from everywhere you go, and everything you see or hear.

need to learn all about thinking and then actually apply those thought techniques to your work. There are so many techniques we can use. Just to name a few: heuristics and mnemonics, critical and systems thinking, or methods like the six thinking hats. Thinking Fast and Slow by Daniel Kahneman is a great book that explains how to think deliberately and effectively.

I like to say that 'testing is a task without end.' The reason I like to say that is because

CONTINUES ON PAGE 10



#### Dear Agony Ant...

How do I tell if my testing mentor is any good?

Scan the QR code for answers



#### Breaking boundaries

# MINISTRY OF TESTING

# CONTINUOUS CALL FOR CONTRIBUTIONS

Ministry of Testing is built with people like you.

Contributions from the community is what keeps our testing planet spinning. Contribute your way, find out more.

ministryoftesting.com/contribute

**CONTINUED FROM PAGE 9** 

there's always more to think about, areas to consider. Look at Ministry of Testing's Test Heuristics Cheat Sheet to see how many different things we need to think about. Whether we decide to act on those thoughts or decide (or have it decided for us) that it is time to move on, or that the risk is sufficiently low, we always need the skill and mindset to be able to think deeply about testing.

#### Find A Thriving Community Of **Testing Practice**

There are some brilliant test communities out there. I have a special affinity for Ministry of Testing (MoT) since that's the one I found first and where I found the most support. A community can not only help you learn and support your journey but also open up or make you aware of opportunities.

A while back, thanks to a sudden hiring freeze on account of the COVID-19 lockdown, I had a job offer withdrawn. That rendered me technically unemployed, and at such an uncertain time for the world it was quite a frightening and daunting prospect. Fortunately the great MoT community gave me support and found me opportunities. I did my first webinar and first online talk, gave training remotely, and was pointed to several job opportunities. I was fortunate to receive a couple of job offers and accepted one just five weeks later. Without the power of the community, I'm not sure how I would have handled that situation.

In 2020 and again in 2021, MoT stepped up to keep the community going during lockdown with TestBash Home. That was such a cathartic event for me personally. I hadn't realised just how much I'd missed connecting with the wider community directly and it was quite an emotional day

So find and get involved in testing

communities. Like me, you might just make lifelong friends and have your career

#### Learn How To Give Helpful Critique

People talk about the testing mindset, but it is very rarely explained. In my opinion there is a fine line between being critical in a shallow way such as, 'it doesn't work, doesn't do this or that' and critiquing an idea or product in a helpful way by suggesting ways to improve the system. That fine line contains some of the most important parts of humanity like communication, empathy, and intent.

How you explain your thoughts can be damaging in a number of ways. Whatever your intent, undue harshness of expression when giving feedback or writing a bug report can be damaging to your relationship with the receiver as well as your personal reputation. Imagine if you had written the code yourself. How would you like to hear approaching products and systems.

to someone who's just starting out in their profession?

How you think about the person or group you explain to can change your tone, body language, and language choice. This is where empathy comes into play. Norm Kerth, author of Project Retrospectives: A Handbook for Team Review created a 'Prime Directive' for retrospectives, but I think it can apply to all aspects of our professional interactions.

"Regardless of what we discover, we understand and truly believe that everyone did the best job they could, given what they knew at the time, their skills and abilities, the resources available, and the situation at hand."

#### Negative Testing Doesn't Require A **Negative Mindset**

As testers we have lots to consider when feedback? How would you give feedback Generally my first response is curiosity. At



AI will not kill testers but they will help testers start from a different place.

**BEN DOWEN** 



some point we will consider risk and the 'sad path,' but that doesn't mean it always has to be negative activity.

There are many aspects of negative testing that can actually make you happy or smile. Here are a couple of examples.

- Graceful degradation done well generally makes me happy.
- Precisely worded, helpful, and appropriately placed error messages increase accessibility, which is always a good thing.
- Good handling of interrupted connections gives the opportunity to have an offline option, accompanied by a helpful message. When a connection is again available, the application should pick up where it left off with no loss of data. Sounds like good quality and user experience to me.

I like to think like this Classic Fisher Price Advert baby when encountering new things. I use my curiosity, consideration of risk, and a number of other influences to find the positive in negative testing.

#### Promote Mobile-First And Universal / Inclusive Design

Mobile-first design forces you to consider all different sizes of devices and how to ensure your responsive design adapts to them. Considering mobile device usage also means you have to focus on the most essential features and content and put them front and centre. This focus benefits all users, including those with cognitive impairments, by potentially reducing cognitive load and enhancing the overall user experience.

A definition of universal design from the National Disability Authority says, "Universal Design is the design and composition of an environment so that it can be accessed, understood and used to the greatest extent possible by all people regardless of their age, size, ability or disability." Considering all the different ways people may use your software means it can be more inclusive without the need for add-ons or adaptations. As technology advances, a universal or inclusive design is likely to continue to be more usable with minimal updates.

Combining those two approaches means software will be accessible and adaptable no matter how, or on what kind of device, a person interacts with it.

#### Being Comfortable With Public Speaking And Storytelling Can Help Your Career

Often testers take the lead in discussions, demos, and conversations about product features and quality. We tell stories to explain what we have found, what we feel needs addressing, and why that's important. Telling stories can help to explain complicated concepts and ideas and adds context to our findings. That's one of the reasons we like heuristics,

mnemonics, and examples to work with when testing.

Being the lead in demos and other team meetings naturally lends itself to storytelling, which in turn gives you some of the skills for public speaking. I've been very fortunate to speak at many meetups and conferences, both in the UK and abroad. However, I'm not a natural public speaker. I was very shy as a child, and a teacher once pushed me into a public speaking competition. It was a horrible experience and one I was keen not to repeat.

However, the storytelling I did on the job as a tester gave me confidence that, when I wanted to talk to people about accessibility, I'd be able to carry it off. I've been doing public speaking for over a decade, and it's gone well, but it's still a challenge. I still get nervous and need breathing exercises to get up on stage, even for 99-second talks!

So even if public speaking seems unattainable for you right now, I guarantee that if you have been a tester for a while, you already have some of the skills required to become a public speaker.

#### Capture And Celebrate Your Successes

Find a way to record your successes no matter how big or small. Doing this helps in many ways: sessions with your line manager of course, not to mention your own self esteem and mental health. It is also excellent for providing evidence for

raises and promotions at end-of-year reviews.

If, like me, you sometimes focus on what went wrong or was less than perfect much more than what went right, then recording your successes as you go can be really helpful. When I first started reviewing my successes, I did it only once a month. I found that I missed a lot that way. Instead, try to capture your wins as they happen, when they are fresh in your mind. That way you can do yourself justice.

The logical next step is to celebrate those successes. Whether that is in a review at work, on social media, or as an addition to your CV, that is up to you. If you do something awesome but you don't share it, you are doing yourself and your career a disservice.

#### **Develop A Habit Of Empathy**

As I mentioned above, testers have to be critics of the product or system they are reviewing. That does not mean we have to be critical of the person who made the decisions or wrote the code.

A long time ago, while I was training to be an auditor, a wise trainer said, 'The best way to create a rapport with people is to tell them that you want to catch them in, not out.' The point of that expression is to be honest and open about what you want

CONTINUES ON PAGE 12



#### Born to explore

#### **CONTINUED FROM PAGE 11**

and need so you can build trust. As for our users, we need to consider all possible uses and users, as neither you nor I can possibly speak for the needs of all of humanity. Not everyone will do the same thing or even use the same small set of variations.

There's a reason I say that testing is a task without end. That's because there are so many variations, as described in the 'it depends' section above. It costs nothing to be kind. It doesn't make you less professional, far from it. It can make you more successful in building trust across your team.

Assume the best of your colleagues. If you come across something that's less than perfect, remember that they, like you, are human and don't have all possible information available to them at all times. We can strive for quality and be kind at the same time.

#### **Don't Dismiss Your Life Experience**

To me this is especially important for those wanting to learn to be testers or those new who have come from other roles. I think all testers bring life experience with them to the work they do. That's one of the reasons why having a diverse set of testers is much better than ones from similar backgrounds.

Many of my prior experiences influenced me as a tester. One standout is being trained as an auditor. Being able to look at a process and understand all the touchpoints and steps, then working through that documented process and assessing if reality is aligned to it, was a great background to build on when I became a tester.

Another part of my background that came in handy in testing was capturing knowledge to create guides and training material when I was a team lead in customer support and processing. That has helped in so many ways: creating how-to guides, designing a testing apprenticeship, and giving workshops and talks. I'm so grateful for all the different paths I travelled before coming to testing.

#### **Helping Others Helps You**

Along the way in my testing career, I've been a mentor, a trainer, a public speaker, and I've written articles too. I've also created a software testing apprenticeship curriculum and content for it. I assumed these roles not just to share my thoughts and ideas but also to help others learn and grow. I've had so many benefits from doing those things: meeting people I might otherwise never have met, financial reward, reputation building, and mental and emotional well-being (confidence and self esteem).

You may feel that you have nothing 'new' had people say that to me directly. But to get things done.

there's something you need to remember. You are unique through your personal experiences and no one else has had exactly the same journey or same experience as you. You have stories to tell and knowledge to share from your approach to testing, a fun bug found, or something unique to you. Sharing makes us all stronger and supports our continuous learning journey.

So help yourself by helping others. The Ministry of Testing Club is a great place to start contributing by answering the questions asked by the community.

Sharing through social media isn't the only way you can aid people in becoming stronger. There's mentoring, or simply supporting those around you. I saw a great thought on LinkedIn: "Be the person you needed 10 years ago. The old you of 10 years ago needed someone. Someone with a set of skills, someone with experience, someone with understanding, someone with values." Alyzande Renard MSc MBCS.

I think it is great to think that way, since we have all had people help us on our journeys. It could be in a formal setting, or just a conversation with an explanation. Everyone needs help, so please try to make others stronger where you can.

As Scott Kenyon said in review comments while helping me with this article.

"Don't be amazing on your own, but be amazing with everyone around you."

#### Testing Experience Doesn't Mean You Know More Than Others

I've encountered testers of all levels, with zero to 40 years experience in software testing. One thing has become very clear to me. The number of years someone has been doing something does not necessarily mean that their knowledge and experience are what you might expect.

There are many facts and many more opinions on how to develop and test software. Doing it for a long time doesn't automatically equate to having good knowledge of testing. Often a tester with one year of experience can have as much or more insight as someone who's been in the field for ten years. People brand new to testing have shared insights with me that taught me something. So don't underestimate a new tester or overestimate an established one.

#### TAKE CARE OF YOURSELF

It is easy to focus on work and your career to the exclusion of everything else, especially in the early stages. But please don't sell yourself short by forgoing a good work-life balance.

I, like many, am highly motivated, working long hours either for work, learning, or for extras (side hustles). And I have sometimes lost focus, overcommitted, to share with the community, and I've even or simply put too much pressure on myself

Be kind to yourself, talk to yourself with empathy and love. Allow yourself to say no, do it tomorrow, or pause if you need to. I have had some hard periods in my life and career with my own mental health issues. I'm still learning about myself and trying to learn what works for me. After working for 25 years outside of testing and then for 20 years as a tester, I still don't have all the answers, so please do try to take care and look after yourself.

And finally, an adage that took me far too long to learn...

#### KNOW THE VALUE OF YOUR CONTRIBUTION AND GET PAID FOR IT

I know that sounds mercenary and selfish, but you only get one career and one life. Make the most of it. It is important that you know your value and are prepared to walk away if your employer doesn't value your contributions by paying you adequately.

I've always been one to collect evidence as I go. And I have used that evidence to ask if there is any appetite for a pay increase at the halfway point of the year. I've never demanded a pay rise. I also use salary market data to back up my proposals. Your salary affects not only your current situation but also your future level of comfort in retirement. So the better your salary, the more you save for later.

Sometimes this means walking away. After over a decade at one company, I tendered my resignation even though I didn't have a new job yet. But I gave three months' notice so that I would have plenty of time to find a new role. It was what I had to do in that situation.

#### **QUICK TIPS**

• Work the TESTING problem, not the

situation, pressure or time constraints. Don't let outside influences stop you from doing a great job.

- Everyone learns differently, so share information in multiple accommodate visual, auditory, kinesthetic, and reading / writing learners.
- It is OK not to know something. There are no 'stupid' questions, or, as I prefer to say, there are no questions too simple or obvious. Questions lead to clarity of understanding for all.
- It is your job if it helps. Saying 'it's not my job' might be technically correct, but it could keep you from suggesting positive change, just for starters. Yes, there are role definitions. But sometimes you just need to do something, even if it's not usually yours to do or it is a little outside your comfort zone.
- Reading code can be a huge advantage. Testers absolutely do not need to know how to code, but it is so useful to be able to read and understand how code works.
- What goes on behind the screen is fascinating. Every part of software hiding behind the F12 button (browser developer tools) is interesting and gives you insights into bugs you might otherwise not have observed. The same applies to APIs, databases... in fact, the whole system.

#### TO WRAP UP

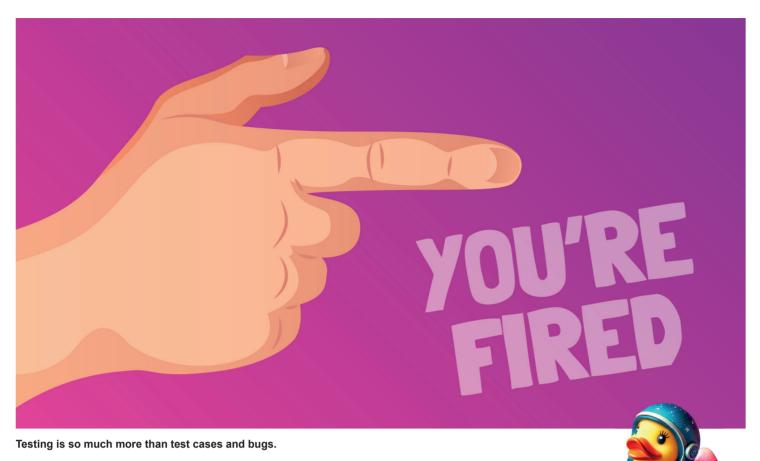
Whether you agree or get any benefit from one, all, or none of these lessons, they are what I will take forward with me through the rest of my career. And you can rest assured, this list is not exhaustive, and I will no doubt add to it. Continuous learning isn't an option; it most definitely is a must! ■





Rethink code reviews. They are a chance to automate, learn, and share knowledge.

**BETH CLARKE** 



# **Laying Off Testers? Think Twice Before You Act!**

#### Improve your job security by changing other people's perceptions of testing

#### BY LIDIA BARKANOVA

We are experiencing layoffs in tech left and right. Software testers are usually some of the first people to be let go: they don't have visible deliverables, they don't write code, the bugs they find "only slow down development," and so on. So why keep them in the organization?

The thing is, testers do a lot of invisible, important work. They are the glue between engineering and product development, and they actually help speed up the delivery of a quality product to customers.

#### HOW TESTERS ARE USUALLY SEEN

Testers are often perceived ONLY as the people who receive the code when a feature is developed and execute test cases on it. Or, if we speak about test automation, a tester is somebody who writes automated tests and maintains them, reports discovered issues, and that's it.

When I started my career as a quality engineer seven years ago, my understanding was similarly narrow. My working routine was creating test cases, based on the requirements written by the product owner and when development was done, executing tests, documenting found bugs, and verifying defect fixes.

I didn't realize that testing is so much more than that. But then I changed jobs, joined various testing communities across the globe, and talked with people from all over the industry. This opened my eyes: testers play much broader roles across a software development group than what I've described above. And the new information I learned shaped my vision and ways of working.

#### CHANGING PERCEPTIONS OF TESTERS... **AND TESTING**

In modern companies, testers are active

of the development cycle. In the definition phase, we help with risk analysis. We look at the requirements, highlight gaps and uncertainties, identify risks that could threaten the product, and foster discussion with product development and engineering. Testers learn about a product from both technical and business standpoints, which allows us to bring new perspectives to the table. We often ask outof-the-box questions which might not be asked by anybody else. Sometimes, such questions can invoke a discussion whose results bring benefits to the company.

Testers help during requirements gathering and design. I remember one day when a product owner came up with an idea for new functionality. He initiated a kick-off meeting with the team. Developers liked the idea and started to jump to a solution. Fortunately, the testers on the team advocated looking before leaping. The testers, who worked with different members of the team from the very start teams across the company and had a good

overview of what had already been done, pointed out that similar functionality already existed in one of our products and that it might be useful to consult with the responsible team and reuse the implementation. And that is, in fact, what happened.

This way, we reduced costs significantly. The new feature required no new design or coding, and we managed to keep consistency in user experience across our SaaS solutions.

#### HOW TESTING CAN EXPEDITE DEPLOYMENT

People might think that testing slows down development. Testing means more work to be done during implementation, and it must be very annoying when developers say, "We are done; let's deploy to production!" and instead hear that there are issues to be fixed before release takes place. Testers execute their scripts and do their checks, find more and more issues, and advocate for them to be fixed. And then, developers need to invest time and effort to resolve the issues found instead of "just releasing."

Testers are the glue between engineering and product development, and they actually help speed up the delivery of a quality product to customers.

Contrary to popular belief, testing drives the process forward. As they do different activities - from exploratory testing to running automated scripts - testers identify the problems and communicate them promptly to stakeholders. Testers not only raise the awareness of the state of product development, but they also advocate for fixing the issues found and often accelerate the development process.

Even though the release of new features can feel more worthwhile than anything else, in reality, if a bug is released, and customers complain about errors in the new feature, the mental cost of context switching for developers (moving back and forth between defects found in released features and work on new features) actually slows down progress on new projects.

#### PRODUCT RELEASE: AN END AND A **BEGINNING**

Finally, there is the release. Development and test execution are completed, and the

**CONTINUES ON PAGE 14** 



#### Dear Agony Ant...

How do I deal with finding critical bugs that were missed during testing? Scan the QR code for answers



Exploratory Learning Centre

**CONTINUED FROM PAGE 13** 

feature becomes available to clients. Yay! People would say that the tester's job is done here, and it's time to focus on the new feature. However, this is not the case.

*If we know that clients* are suffering or stuck in a certain place, we will voice it aloud so that our engineering team can address it.

After the release of a new feature, we analyze metrics around it: how fast and reliable it is and how the new functionality affects overall product performance and stability. This is important for identifying critical parts of the software and prioritizing work in the future. We also look at the adoption of the new feature: how clients are using it, which questions they raise, and where their main struggles lie.

Last vear at a conference, I shared how knowledge of customer usage data can help us shape testing strategy and explained that we testers need this knowledge to do our job efficiently. If we know that clients are suffering or stuck in a certain place, we will voice it aloud so that our engineering team can address it and will advocate solving those issues with high priority. And in another common case, if a feature is not widely used, there is probably not much sense in investing time and effort in fixing small defects in it which were found internally.

#### A TESTER'S WORK IS NEVER DONE

Our work never ends. We are the glue holding the team together, we accumulate knowledge from developers, managers, designers, customer support engineers and other stakeholders, and we see the big picture. All of these points of view on a product are important in the tester's work, and you can hardly find any other role in the company that has this knowledge.

This unique mixture of skills and knowledge allows us to be the go-to people in case of any questions about "how it works now." We know the current state and can assist in bridging the gap between vision and reality, significantly easing development.

Now that you know what your testers REALLY do, think twice about letting them go. ■

### THE 10 Ps OF TESTABILITY

By Rob Meaney and Ash Winter

#### Testability is a measure of how easy something is to test.

When there is a lack of testability, it slows down feedback, reduces the quality and depth of our testing efforts, increases cost, destroys motivation and morale and ultimately results in poor quality.

Conversely, focusing on Testability can unleash your development team to do their best work, allowing them to focus on the work that matters and facilitates:

- 1. Faster Development.
- 2. Earlier testing
- 3. Broader and deeper testing
- 4. Better, more robust automation

This focus allows teams to get changes into production safely and quickly in a sustainable way. The 10 P's of Testability model help teams identify ten context factors that influence the team's ability to test:



#### 01 PEOPLE

The people in our team possess the mindset, skillset and knowledge set to do great testing and are aligned in their pursuit of quality.



#### **02 PHILOSOPHY**

The philosophy of our team encourages whole team responsibility for quality and collaboration across team roles, the business and with the customer.



#### **03 PRODUCT**

The product is designed to facilitate great exploratory testing and automation at every level of the product.



#### **04 PROCESS**

The process helps the team decompose work into small testable chunks and discourages the accumulation of testing debt.



#### **05 PROBLEM**

The team has a deep understanding of the problem the product solves for their customer and actively identifies and mitigates risk.



#### **06 PROJECT**

The team is provided the time, resources, space and autonomy to focus and do great testing.



#### **07 PIPELINE**

The team's pipeline provides fast, reliable, accessible and comprehensive feedback on every change as it moves towards production.



#### **08 PRODUCTIVITY**

The team considers and applies the appropriate blend of testing to facilitate continuous feedback and unearth important problems as quickly as possible.



#### 09 PRODUCTION ISSUES

The team has very few customer-impacting production issues, but when they do occur, the team can very quickly detect, debug and remediate the issue.



#### 10 PROACTIVITY

The team proactively seeks to continuously improve their test approach, learn from their mistakes and experiment with new tools and techniques.



Personas help us think of characteristics instead of ourselves.

**KATJA OBRING** 



# Five Things I Learnt Speaking At A Testing Conference For The First Time



BY DEBORAH REID

I was due to speak at an event but it was delayed 2 months. Although this gave me more time to prepare, it also gave my nerves more time to build up!

I have previously spoken at a small meetup (10 people) and run a workshop for about 20 people, all on the same topic, however that was a while ago and I was nervous for this event - it could be up to 100 people, and no familiar faces in the crowd (some people prefer not to know anyone, however I imagined that a few strangers were cheering me on)

As testers, we are always looking for potential risks with a system or process, how to mitigate these risks and be proactive about resolving them.

I knew I had a great topic for a talk, but wasn't sure it would be accepted, so the nerves didn't really kick in until it was confirmed. Whilst battling the nerves I tried to identify what was concerning me - what specifically was I nervous about - what risks was I foreseeing, and how could

I attempt to mitigate them, setting myself up for success when delivering the talk.

This is the same approach I took after I had a conference talk accepted at a relatively large conference. I was nervous, but I decided to try and understand why I was nervous and what the risks were. Once I'd done this I could identify a plan to (hopefully!) mitigate and overcome each one.

RISK 1: I WILL FORGET WHAT I WANT TO TALK ABOUT AND JUST STAND THERE SILENT

Context: Whether it's in a 1-2-1 at work, a presentation or social event, I think we've all experienced our mind going blank - and you've possibly seen this happen to someone else too. It's natural to be anxious/nervous (it's actually good because it shows you care!), sometimes we need reminders of what we want to say, so I recalled a useful tool from a big meeting at work a few years ago.

**Mitigation:** Note cards

**Notecards are ok-** I see many speakers these days confidently speaking without any notecards at all (including but not limited to Jenny Bramble, Vernon Richards, Simon Prior, Jenna Charlton) - either they have rehearsed lots, or their slides are enough of a trigger to remember things, or

I was nervous, but I decided to try and understand why I was nervous and what the risks were.

maybe they aren't dyslexic like me? I tried running through my presentation without note cards but I couldn't remember some things that weren't on the slides (I had lots I wanted to talk about but I just don't have the memory to recall it all!) With my talk

being around accessibility, I didn't want to have too much stuff on my slides, I wanted the slides to contain enough information without being overwhelming. Also I've heard about speakers just reading their slides and it's not very engaging for the audience at all.

I tried to have key things on my slides, and then when there was something else I was keen to speak about, I wrote it down so there were some triggers (not a script) on some note cards - together with the slides covering the really important things I wanted to speak about. It's important that the notecards just contain triggers, and not a script, the last thing I wanted was to be reading from the notecards. It was important to number the cards as well, just in case I dropped them, then I knew what order they had to be in!

The aim was I'd still remember some of the extra bits not on the notecards but if nerves took over on the day, they weren't

**CONTINUES ON PAGE 16** 



# WE'RE CREATING A TESTING PLANET

For you. And with you. Join us.

ministryoftesting.com

16 THE TESTING PLANET, ISSUE 11, SEPTEMBER 2024

Test we can

#### **CONTINUED FROM PAGE 15**

as critical as what I had written down. I also wrote down 'slow' and 'smile' as when we're nervous we can easily forget these! Remember, the audience doesn't know what you wanted to say, so if you forget something, they won't know!

I truly admire people who don't need note cards, but I also accept that my brain works differently, and I don't think anyone in the audience gave 2 hoots that I had notecards.

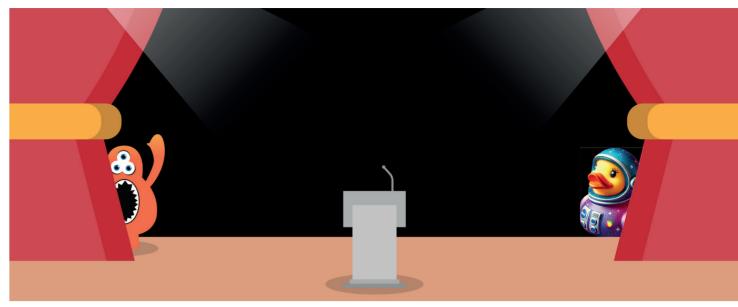
RISK 2: THE AUDIENCE WILL BE DISENGAGED, NOT PAY ATTENTION, OR EVEN WORSE, THEY MAY FALL ASLEEP

Context: I knew this topic was interesting to me and some other people, but I couldn't be sure it would be to the whole audience. Some conferences may have multiple tracks so people are actively selecting to attend your talk. At the conference I was speaking at, I knew there was just one track, so whether they wanted to or not, they were stuck listening to me! In the end there was another track for virtual speakers in the next room, but I planned something to try and reduce the likelihood of disengagement.

**Mitigation:** Audience engagement - get them involved

Interaction with the audience is good: There are a few ways you can do this - try one, or even, try all of them!

- You could start by checking that everyone can hear you as sometimes the microphone isn't working properly or you're too quiet which can mean some people can't hear you and therefore lose interest right from the start.
- 2. If they seem sleepy, ask them to stand up, stretch and reset. This might be relevant just before or after lunch, or for the first or final talks of the day! Or if you're feeling really adventurous, get people to dance, like Vernon and Stu did at Test Bash!
- 3. I had planned a couple of activities which people could do alone or chat to the person next to them. I warned them these were coming up in my intro, and the activities gave me a chance to catch my breath, take a sip of water and take in the experience (and check the time if you're concerned). It wasn't a vague "what do you think?" interaction but a couple of small activities where they had to assess if something was accessible or not. I made it clear that it was about learning so not to get too bogged down in getting it right. Giving the option to chat to the person next to them or to think alone gave them options depending on what people preferred (think about neurodiverse or shy folk in



Rehearse, rehearse, rehearse. It's key to getting up on stage.

the audience), and I set a 1 minute timer as that is the time I'd allowed in my practice sessions. I also asked a question about how many people they thought were affected by something in the world. Some shout out, some whisper

It might sound obvious, but practising is important, and variety in practising helps too.

and some just think and don't interact (all ok with me) but if you give them some information to base their answer on, you'll always get someone joining in! You could do a hands up exercise if they've experienced something or gauge understanding of your topic before starting.

Whatever engagement activity you choose, try and keep it short, and relevant to your talk if possible. Something memorable is even better!

RISK 3: MY NERVES WILL TAKE OVER AND I WON'T BE VERY GOOD ON THE DAY

Context: I know from some work and social events in the past that I do not feel confident with lots of eyes on me, quite the opposite! Historically it's made me feel uncomfortable, self conscious and nervous. Planning what we will say or do is a great trick to feel more confident as it's not unknown content - it may sound obvious - I know from experience that familiarity and lack of unknown are linked to reducing anxiety and nerves.

**Mitigation:** Rehearse so the content feels

more natural and free flowing.

**Practising is key:** it might sound obvious, but practising is important, and variety in practising helps too. Each run through I did was slightly different but helped me feel more confident with my content and alleviate some nerves. Here are some things I tried:

- Rehearse it alone, with my partner, with some colleagues: (Thanks to Saida, Cheryl and Alison at Bloom & Wild!) This is also useful for feedback, if there was a spelling mistake or something was not clear, they were on hand to provide that feedback. Sometimes I would look at my slides more than the people, sometimes I would stare at my slides a balance is ideal!
- Rehearse it sitting, standing, pacing: I wasn't sure how I'd feel on the day so I tried all of them so I could decide nearer the time. Traditionally people stand still or pace, but with my nerves, I had a chair on stand by incase I needed to sit down! In your rehearsals, see how you feel most comfortable do you feel like pacing a little will ease your nerves? Will standing still with something to lean on make you feel more confident? You can ask if there will be a lecturn or space to walk around and then practise knowing the limitations/options on the day.
- Rehearse it speaking out loud, running through it in my head, whispering it: Running it through in my head was great sometimes, but of course you can't do that on the day! Whispering was a great halfway measure when I didn't feel like speaking loudly! Running it through in my head when I just wanted to reinforce the content in my brain and not practise my delivery, though delivery is key, hence speaking it out loud in practice is important too.
- Rehearse it with my note cards and without them: This was important,

- especially as I wanted to see what I could remember, and create triggers for other important things on my cards. I had one note card per slide if I needed to recall extra information. You could attach them together (or number them) to prevent potential panic of dropping or muddling them up on the day!
- Rehearse it with a timer and without a timer: I had been asked how long I wanted for my talk and I'd estimated 30 minutes. I thought this would allow for 25 minutes of me speaking and 5 minutes for questions. When I timed my talk, it was pretty consistently around 19 minutes, so I knew with me allowing for a couple of minutes for the audience participation and questions, and hopefully slowing down (thanks to the notecards!) as well (when nervous, many people including me speak quickly!), this was probably ok.
- Pausing during a run through to edit slides or notes, running through without stopping: We've all been there, running through our slides and we notice something, sometimes you need to stop and either make a note of something to edit, or actually make the change to your slides or notes, but also you need to practise full run throughs without stopping like you would on the day. Remember the audience doesn't know what you don't tell them, and no-one is perfect so a small spelling mistake or transition issue is not the end of the world.
- Rehearse it with a clicker and without a clicker: Most of my practices were using the spacebar to trigger my transitions but in my last few practice run throughs, I wanted to check how the clicker worked and how it felt and how to hold that along with my note cards. I borrowed a clicker from my dad, but in the end the venue had their own (it was different shape and size to mine though!)



Do it until it feels comfortable.

**KULAS ANGELES** 

99



RISK 4: ANOTHER SPEAKER WILL COVER ALL OF MY CONTENT LEAVING ME NOTHING TO SPEAK ABOUT

Context: Whether it's reading an article which covers the same topic as another, a repeated storyline on TV or your friend telling you the same piece of gossip again, we know that repetition isn't always exciting. New, fresh, interesting information is more likely to capture our attention. So I was keen to approach my talk content this way too.

**Mitigation:** Trust the organisers, listen to the other talks and have confidence in your content. Offer a unique insight, talk about your experiences, something no one else can do.

Try not to be concerned about overlapping content with another **speaker:** When the schedule for the conference was announced, I noticed another speaker on a similar topic. On the original schedule, I was due to speak after this person and was worried I'd spend the break between their talk and mine worrying about how to edit my slides to reduce overlap. When the conference date changed, so did the schedule order and in the end I was first. During the other speakers' talk, they referenced stuff I'd mentioned a couple of times (which was nice), skimmed over a couple of areas that may have overlapped and focused on

other areas. Remember, the organisers had set the schedule (and in our case seen our slides), so they knew we weren't going to be 100% overlapping! Alan Giles spoke after me and did an awesome job. Some tips if you are second, to show you were paying attention and give a nice shout out are:

- Reference or reiterate points from the other talk at relevant points
- Skim over sections that might have been mentioned previously
- Add anecdotes/personal experiences to your talk if you do have to skim over things

There's no doubt that going second can be challenging, but trust the organisers, and add your own unique spin.

RISK 5: THERE ARE MANY THINGS WE CANNOT CONTROL WHICH CAN AFFECT OUR DELIVERY AND CONFIDENCE

Such as anxiety on the day, but some we can control, like influencing how the audience might feel whilst listening. I can't control how well the talk will do and that makes me scared, and I fear I won't feel confident

**Context:** If you're lucky enough to have been able to work from home even 1 day in your career (in these times, who hasn't?), you've likely either been tempted to, or

even worked, in your pyjamas. However, I've always felt they make me feel tired and want to rest, whereas I want to feel inspired, awake and invigorated in order to do my best when working. I may work in a tracksuit, jeggings and a big cosy jumper, but these items are a mix of comfort but professional 'enough' for me to get my work done. This will be different for everyone! Therefore it's important to be comfortable and confident in your own style.

**Mitigation:** Focus on what you can control - this can include a drink or snack just before your talk, somewhere quiet to focus and remain calm just beforehand, also pay attention to what you wear as this can be a tool for success when presenting.

Choose an outfit that makes you feel confident: I found out 24 hours before the conference that they were recording it to share later. I panicked as not only did I want to look and feel confident but worried what would look good on camera/video! You don't want to be adjusting your outfit all the time, or wear an itchy material. You could even rehearse in a few outfits to check you're comfortable. Think about the potential for the room to be a bit cold or a bit warm and perhaps have an option to cover both. I had some walking in my commute to get there so I wore trainers but then changed into boots for the rest of the day as I felt more confident in them. I ironed my outfit the night before too so I was prepared. It was also raining the day of my talk so it might have been a good idea to pack a spare something in case you get drenched! Luckily the rain wasn't too bad until after the conference.

For some people, a T-shirt and jeans, like they wear everyday, will be what makes them feel comfortable and confident. For others, maybe a favourite colour, a blazer, a more formal dress or shirt perhaps feels more empowering. Unless there is a dress code, you can wear whatever you like! And like I said, you can change into it specifically for your talk if you want to (Lee Marshall surprised us by changing into a Pirate shirt at TestBash Bucks!).

Speaking in front of people can be scary. I don't know if it gets less scary the more times you do it (maybe some more experienced speakers can share their experiences), but if you have an important message to share with people, submit your talk and try these tips and who knows, you might quieten your imposter syndrome for a while!

I had some lovely comments directly after, during the lunch break and at the end of the day so I feel like my talk went quite well! Things can always go better or worse but I told myself, if just one person was interested or inspired after my talk, I would feel like I had done well and all the prep was worth it!



product development testing operations

swith risks

sticky notes with risks

quality

sticky note

risk storming

product | development

## risk storming

product development

sticky note

aspects

s with risks

sticky notes with risks

product | development | testing | operations

t testing operations



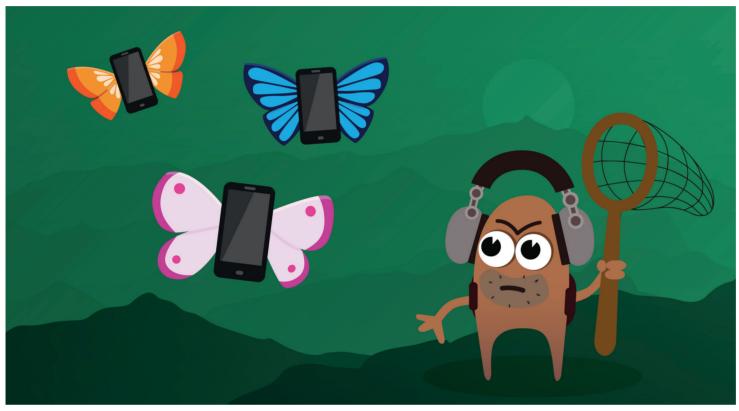
Let's go p1ac3s

THE TESTING PLANET, ISSUE 11, SEPTEMBER 2024

# Where Am I and Where Is My Test Data?



Enhancing Testability Of Location Services



The Controllability, Observability, Decomposability, and Simplicity (CODS) model gives us a great framework to testability.

#### BY ASH WINTER

"Mobile applications can be among the most difficult to test because of their reliance on background processes like location services. Enhancing testability is vital to the testing effort. Thanks to the Controllability, Observability, Decomposability, and Simplicity (CODS) model, we have a great framework to guide us on the journey to optimal testability.'

R ecently, I've been working on a mobile application that has provided a big testing challenge. The application in question works in the background, using location tracking services. The end user frequently keeps the device in their pocket while they go about their business. Our application generates insights for them based on locations they visit.

Location tracking-based applications can be time-consuming to test, as you need to move around just as the end user will do. You can't rely entirely on mocks and location spoofing, since you need to

test in the physical world too. Plus there are many different configurations to take into account - many devices have WiFiassisted location tracking, for example.

In short, to test the application well, we needed better testability.

IMPROVING TESTABILITY: THE CONTROLLABILITY, OBSERVABILITY, DECOMPOSABILITY AND SIMPLICITY (CODS) MODEL

When it comes to testability it's important to have a model in mind. So my friend Rob Meaney created the Controllability, Observability, Decomposability, Simplicity (CODS) model.

- Controllability is the ability to control the system so you can reproduce each important state.
- **Observability** is the ability to observe everything important in the system.
- **Decomposability** is the ability to analyze the system as a series of independently testable components.
- Simplicity is how easy the system is to Thorough research. Popularity isn't understand.

Suggest to your team that you start using this model before a single line of product code is written. It is far easier to build in testability at that stage.

I'll break down some of the improvements we made to our application as we built it. In our story, we started with decomposability, because being able to test each component independently means getting feedback as early as possible.

DECOMPOSABILITY IN PRACTICE: CHOOSING THIRD-PARTY LIBRARIES AND BUILDING **CUSTOM INTERFACES** 

Since we were building a mobile app, we needed to decide how to interact with device location tracking. So, to avoid reinventing the wheel, we evaluated several third-party libraries that provide tracking services. Using third-party libraries can save the team a lot of time and effort. However, you need to choose carefully. Based on my team's experience, I suggest:

everything, but with third-party libraries

it can be telling. To begin with, you can look for large numbers of users and star ratings. What's more, the library should be actively maintained, which is often reflected in recent issue fixes and robust unit tests. It's worth joining the online forum for the library if one exists. And remember, especially if it's open source, be kind in your interactions.

**Build your own Interfaces.** This is where decomposability comes in. With custom interfaces, you can add highquality logging. And a well-instrumented interface will help you uncover bugs in the third-party library's communication with your business logic.

If you as a tester can get involved in decisions on third-party library usage and interface design, your testing journey will be far easier.

#### CONTROLLABILITY IN PRACTICE: SETTING **APPLICATION STATES**

You can imagine how much variety there is when using a mobile application that depends on location tracking. Driving instead of walking, not moving for a while and losing the positional signal or going through a tunnel are just a few examples. There are lots of `that's weird` moments.

To test code thoroughly you want to be able to set the application to its most important states. This requires controllability.

It's a boon for your testing to start to gather feedback from your device. Yet, this only goes so far, especially when diagnosing bugs, because the infinite variety of possible locations makes it much harder to reproduce what you find. Reporting bugs that are hard to reproduce doesn't help your relationship with your developers.

To test code thoroughly you want to be able to set the application to its most important states. This requires controllability. You need tools to assist you, and in the realm of location services, GPS Exchange Format (GPX) files containing stops and routes to define a journey are the name of the game. We needed to:

- Draw routes on a map so we could simulate a device in motion.
- Simulate stops of varying durations at different locations.
- Export those routes and stops to GPX files to include in bug reports.



#### Dear Agony Ant...

I feel pressured to skip some test cases to meet the schedule. Is it ever acceptable to cut corners?

Scan the QR code for answers



• Import and explore GPX files generated on the device during travel.

We used three classes of tools to augment our testing:

- · Virtual location tools
- An editor for GPX files
- GPX tools installed on our test devices

With this toolset, we could easily share information about bugs and exploratory testing. Data generated by physical devices was key, and we could then recreate specific scenarios using targeted GPX files.

**OBSERVABILITY IN PRACTICE: GATHERING** INFORMATION ABOUT LOCATION SERVICES

Controllability and observability are two sides of the same coin. You need to be able to see if you have set the application into its most important states. Also, you need to be able to see what happens when you are moving between states.

For our team, reviewing this log data was also a great group exercise for deciding where we truly needed logging.

These days, observability is a hot topic. You'll also find a lot of tools of varying cost and complexity. I urge you to do some research before you invest in any one tool.

We implemented the following patterns: Classify your important events. Mobile devices generate many log events. And location tracking libraries generate a ton of information in their debug modes. It is eminently possible to lose important information because of the sheer volume.

To counter this, enumerate all your important events, giving them unique IDs and human-readable names:

```
NotSet = 0
NotInitialised = 10000,
Initialised = 10001
NotAuthenticated = 20001.
Authenticated = 20002
```

For our team, reviewing this log data was also a great group exercise for deciding where we truly needed logging.

Choose a centralized logging tool. Since you will be on the move often, you device. All devices should log to the same how it went! ■

tool. Finding patterns within logs from many devices helps to contextualize problems you find. If a problem appears to be common across devices, it probably merits further investigation. We used Bugfender, but there are many others available.

Add hidden development menu. As well as exporting logs to a centralized location, being able to see what is happening on the device itself is important. For this purpose, we added a special submenu, well hidden in the app's "About" screens. It enabled us to trigger insights from the app while on the move. To start with, we added the ability to list the locations that we had been to and how long we had spent at each stop. As the testing evolved, we added more to this menu to help us.

SIMPLICITY IN PRACTICE: OPTIMIZING TEST **AUTOMATION FOR LOCATION SERVICES** 

Mobile app test automation is difficult, and even more so if you are running it within a pipeline. The apps need to be running in the background, too. And if you're using emulators for any other testing, don't rely on the location data they provide. Keep it simple; focus on unit and component tests, as they are much more reliable. Add end to end tests sparingly. Focus on three areas:

- Initialize your location tracking library correctly. Usually, you need authentication, device permissions, location accuracy settings and current location. Getting this right will make all later testing much smoother.
- Monitor changes to the contract between your interface and the **location tracking library.** Detecting changes in that contract is expensive later in your testing.
- Verify that the interface handles business logic correctly.

Keeping these aspects in mind as you build out test automation will minimize false starts. Considering the lengthy build times for mobile apps, saving testing time is of the essence.

#### TO WRAP UP...

We found countless interesting problems during testing. One of my favourite silentbut-deadly bugs had to do with devices automatically protecting battery use by limiting location tracking.

Mobile applications can be among the most difficult to test because of their reliance on background processes like location services. Enhancing testability is vital to the testing effort. Thanks to the CODS model, we have a great framework to guide us on the journey to optimal testability. Give the CODS model a try will need to save log events from your yourself and let us know in the comments



**Interested?** Find your next

opportunity at Capital One





Once is happenstance, twice is coincidence, three times is a pattern.

**KATE PAULK** 

#### Expect the unexpected



## **Quality Coaching:** A Road Less Traveled

Explore the unique and impactful role of a quality coach in revolutionising software testing practices

#### BY MIRZA **SISIC**

ne of the most significant advantages of a software testing career is the vast amount of options open to you to grow in your career. As we get more experience, we may get tired of doing the same old thing every day, and feel the need to look for changes and challenges.

And challenges abound: you can become an automation specialist, security tester, niche domain specialist tester, generalist tester, Agile tester, SDET, QA team lead, test architect, test manager, user acceptance tester, head of quality, mobile tester, usability tester, performance tester, just to name a few. And one of the newer options for you: quality coach!

The role of the quality coach is relatively new, and since it is not encountered as often as other QA / testing roles, it remains virtually unknown to many testers and IT professionals. I'll outline aspects of the role in this article.

#### WHAT IS A QUALITY COACH?

For a certain kind of tester, the role of a quality coach is ideal. If you...

- Are empathetic
- · Analyze risks with comfort and skill
- Lead people and teams effectively
- Have broad and deep technical knowledge
- Know how to set up processes from the ground up

You might be a great quality coach! Sprinkle in relevant domain knowledge, the ability to ask the right questions, and keen listening and observation skills, and you have a winner. Not all quality coaches come from a traditional QA background; some coaches I've spoken to were developers, scrum masters, or customer support managers before they moved into the quality coach role.

To me, this seems like a dream job.

Just imagine helping teams set up and maintain a culture of quality and to advocate a change in mindset, with the

The role of the quality coach is relatively new, and since it is not encountered as often as other QA / testing roles, it remains virtually unknown to many testers and IT professionals.

end goal of delivering the best possible product for customers! Some of the things done by a quality coach in a typical working day:

- Attend sprint demos
- · Work with stakeholders to understand upcoming features thoroughly
- the documentation contributing to it
- · Lead practical workshops for testers and non-testers
- Teach testing techniques to developers and other people in non-testing roles
- Participate in bug analysis
- Take part in triage sessions, such as going over production defects
- Organize mob-testing sessions throughout the organization
- Promote pairing
- Propose and implement quality improvements
- Use monitoring to obtain data about use of the product by real-world customers
- Analyze existing metrics and propose improvements
- · Teach by doing: join the teams in their regular testing activities
- · Take part in regular meetings, such as scrum events
- Formulate a quality road map for the whole company
- Practice individual one-to-one coaching with team members
- Work closely with domain experts
- · Establish new and improve existing processes

WHAT IS THE DIFFERENCE BETWEEN A QUALITY COACH AND A QA LEAD OR TEST MANAGER?

Quality coaching focuses on:

- Changing team and organizational culture to promote shared ownership of quality
- Helping the team improve soft AND technical skills
- Facilitating learning and adaptability
- Building toward long-term continuous improvement

Also, a quality coach fosters individual growth among the team members. A good coach will work themselves out of the job, so to speak, by enabling and encouraging the team towards independence.

In contrast, QA leads and managers are focused on team activities, testing processes, day-to-day testing activities, reporting to stakeholders, evaluating the risks of release. Their priority is usually short-term projects and improving technical knowledge to get the job done. Additionally, typical leads and managers generally have some executive authority, while a coach is more of an advisor.

A common quality among all three roles is excellent communication and organizational skills. While managers will primarily focus on team-level communication, the coach will put more emphasis on cross-team and organizational communication. Even in organizations where there is no formal quality coach role defined, the whole organization can



Tools and technology will come and go but the real testers are here to stay.

THE TESTING PLANET, ISSUE 11, SEPTEMBER 2024

#### NaN expectations



benefit from leads or managers adopting a coaching approach. This way they can inspire individual contributors towards more autonomy, resulting in increased performance and productivity.

#### ISN'T "QUALITY COACH" JUST ANOTHER NAME FOR SCRUM MASTER OR AGILE COACH?

Generally, a scrum master or an Agile coach guides the team through the initial Agile transformation, providing support during scrums and stand-ups, for example. And they usually take the lead when it comes to grooming the backlog.

In contrast, the quality coach is more of a generalist. The coach promotes quality through the entire process with an end goal of shipping products of higher value to the customer. But the quality coach shares with the scrum master and Agile coach the duty to lead by example and to foster changes in mindset.

HOW DO YOU SUCCEED AS A QUALITY

#### **Get Managerial Buy-In**

You might have great leadership skills as well as broad and deep technical knowledge. But to be truly successful in this role, it is crucial that you have support from management at your organization.

Sometimes, suggestions by a quality coach might seem disruptive to the way people are used to doing things. Here's where having support and general consensus goes a long way toward achieving a culture of quality. You will still need to put in a lot of hard work and make long-term commitments, with having the flexibility to adapt to changes when needed. Remember, someone in authority will need to give your recommendations their blessing before you start implementing them. Finally, but perhaps most important, you need to get your teams on board with your vision of quality!

Without backing from management, any significant change will simply not be possible. Introducing process changes will often require additional budget. The management will approve the additional cost of a measure only if they are convinced that the measure will help retain existing customers and attract new ones. A quality coach needs to speak to everyone in their own language, and it's wise to remember that management's first language is generally that of money.

#### FIT IN ON THE TEAM AND AT THE ORGANIZATION

Depending on the direction you are heading, and whether or not the company has dedicated testers, will determine how you implement quality improvements. Having a dedicated QA department in the company means that the testers will be able to assist in teaching team members from other roles how to get more involved in testing. If there are no dedicated testers

in the company, a quality coach will need to distribute testing activities among the existing team members. For example, developers could perform technical testing, unit, integration, and end-to-end testing, while product owners or other business people could do the functional testing.

And organizational structure beyond the immediate team matters, too. For example, some organizations have a head of quality, to whom the quality coach reports, while the technical team leads might report to the quality coach.

#### PROMOTE SHARED OWNERSHIP OF QUALITY

When a quality coach is placed in a team, they should be an advocate and a champion of quality. A quality coach can help implement a whole-team approach to quality, where ownership over features is shared by the entire team. And of course, if needed, the quality coach can be a member of multiple teams.

Team adoption of shared ownership of quality can succeed only after you have gained the trust of the team, as this is not a simple short-term process. You need to set clear expectations. Measures that embody shared ownership of quality might include moving towards earlier testing, more effective detection of defects, and shifting left or right in terms of when testing activities occur. Shifting left usually entails testing "earlier," evaluating the product requirements

before there is any code. Shifting right can include testing in production and monitoring use of the product by customers in real time.

You will need to practice what you preach and lead by example. A quality coach is not there just to lecture, but

But to be truly successful in this role, it is crucial that you have support from management at your organization.

to teach by doing as well. Fortunately, technical people are very responsive to logical arguments and are quick to try out new approaches, out of curiosity. Show that you care; do not hide that you are passionate about promoting and embedding quality. Collaborate frequently to drive change and motivate team members. No significant change happens overnight and you need to be patient and persistent. Small incremental changes will build up over time and results will speak for themselves.

**CONTINUES ON PAGE 24** 



#### Giving you some Gravi-tea

#### **CONTINUED FROM PAGE 23**

#### PREACH TESTING TO DEVELOPERS

Sometimes developers are so used to having a dedicated tester that they don't even consider the alternatives. Apportioning a few testing tasks to every team member no matter what their role removes and reduces bottlenecks. Quality should be the responsibility of every team

If you are a bit old-fashioned, like me, you might think of it as being honorbound to deliver value to the customer! Concepts like the theory of constraints can help teams deliver faster; with everyone testing, you eliminate the "bottleneck" of testing at the end of the cycle. Pair with the developers often, hold workshops to teach the team useful testing techniques, and try to involve them in end-to-end automation as well.

Here's a real-world example for you. On my current project, the developers are the ones doing most of the end to end automation testing. It's a way to do modern testing through BDD, the testers write test scenarios before the code, and those scenarios are reviewed by other testers, product owners, and developers. The three amigos session is a very effective way to distill your requirements and to get a deeper look at how to test your features better.

#### CONVINCE QA YOU'RE ON THEIR SIDE

When you first come on board, QA team members might look at you with some suspicion, thinking you'll advocate for a way to eliminate testing as a role altogether.

So you will need to proceed with compassion and caution. You might explain your role as advocating for a cross-disciplinary approach to quality. Ideally, people with different perspectives should be able to share their opinions and

Another way to build trust: Find out what motivates your testers. If it's exploratory testing, have them pair with the developers to teach them how to make the most out of their exploratory sessions. Also, try to identify what people from different roles do not like about current practices and suggest concrete improvements.

#### MANAGE CONFLICTS

In any line of work, occasional conflict is unavoidable. But it does not always have to be destructive. Conflict can illuminate different points of view, leading to constructive changes in thought and process.

It's best to start by listening to the parties involved and getting to know each side of the story to understand all viewpoints. This can be a simple face to face meeting. augmented if you want by using a "retrospective" note-taking tool.

Be gentle when introducing new ideas. If people go along with it, proceed, otherwise adjust your approach. Show that you care and the team will trust in you, and trust goes a long way towards helping to resolve conflicts.

#### TO WRAP UP...

To reiterate some of the points made above: The quality coach will need to work closely with the team and even be a member of teams to lead by example and earn the trust of team members. For a quality coach, it's also important to be able to handle and resolve conflict situations in a manner that is respectful and to be able to turn conflict into something productive.

Promoting quality and advocating for shared ownership of quality is a continuous, ongoing task. There will always be room for improvement. However, it's also important to give praise for successes along the way. This will motivate the team to stay the course of a cultural transformation. Pair with the developers, convince them to help out with the automation effort, and teach them testing techniques and the value of exploratory testing of their own features.

Getting traditional testers on your side might be tricky, but it's not impossible. This might require a unique and creative approach to convince the testers to share their responsibilities with other roles. Also,

make it clear to the testers that they can do more for quality than simply planning and executing thousands of test cases! It's likely that their jobs will become more interactive and interesting in a whole-team approach.

A quality coach will also need full management support and buy-in, from top to bottom. As a coach, you might not have the authority to tell someone to change the way things are done: you'll need a manager to tell them to do it.

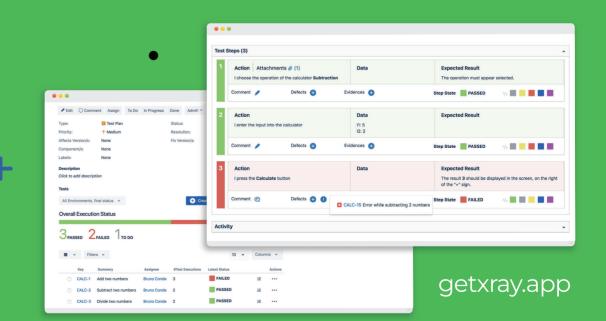
And remember, the most important thing is to convince people, not to boss anyone around. We want to improve quality and integrate it into our very culture, and quality is not simply believing in clean code and well-tested software. An essential precondition for quality is trust among team members and within your organization. ■





With test management woven into every stage of development, quality now comes naturally.

Xray uses **Jira native** issue types so your **Development** and QA work using the same terminology, under the **same toolset**.



#### NO BUG LEFT BEHIND

Catch problems faster, and release quality software with confidence. With development requirements naturally linked to testing, you'll never overlook another test.

#### QUALITY AT SCALE

Pull off large-scale testing projects without missing a beat. Our powerful REST API integrates with leading CI/CD and automation frameworks.

#### GOODBYE SILOS

Give development and QA teams up-to-the-moment insight into test coverage and status, all from inside user stories and agile boards. THE TESTING PLANET, ISSUE 11, SEPTEMBER 2024

100 Continue









When you compare your tools to others. Do it in a healthy fashion.



# Pragmatism, prioritisation and patience

Skills for success in start-ups as a QA Engineer!



#### BY AMY STUART

This article explores the specific skills that are beneficial to consciously practise and improve if you are joining a start-up. Additionally, it provides a clear illustration of what working in a start-up is like, and how it compares to the experience of working at established companies. We will discuss what can be the same, what is different, and what skills and knowledge are transferable and useful. Alongside defining the skills for success, and providing insights, there are descriptions of the challenges from my general experiences of working in start-up

and similar fast growing environments. Finally, I will provide insight into the positive impact refining those skills had on the way I worked.

#### TURNING CHALLENGES INTO OPPORTUNITIES AT START-UPS!

Start-ups are companies in the early stages of operating as a business often going through exciting stages of growth. Working for a startup can be an exciting opportunity to get involved in the beginning of a business's story. However like any project, getting involved in the early stages comes with unique challenges related to figuring out what works, and what doesn't. As

someone working in a quality role, it's important to know how this affects you specifically.

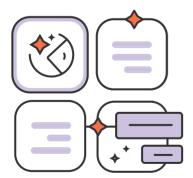
Start-ups face challenges that established companies do not, or may experience similar challenges more frequently. By their nature they are new businesses, and this comes with an increased element of risk. Many start-ups fail, or experience turbulence in their first few years of operating. Action is necessary to keep up with changes in the market in order to succeed. Company structure may change to compensate. There could be changes to teams, including people joining, transferring, or leaving. There may be product roadmap pivots to keep up with

Getting involved in the early stages comes with unique challenges related to figuring out what works, and what doesn't.

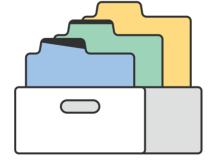
market changes or competitors.

To be a start-up, is to be in a state of operation to prove a business idea works and is viable, and big changes are sometimes necessary to prove this. Change can be exciting, but it can also feel

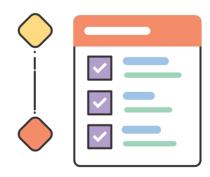
# Move fast and (don't) break things with Postman



Use Postbot, Postman's Al companion, to write and update API tests in seconds.



Create reusable scripts and share them with others via the Package Library.



Run tests in your CI/CD pipeline with the Postman CLI—and view and analyze test reports in Postman.



Postman.com

disruptive to those impacted. As a result, many who decide to work in start-ups tend to be ambitious and driven people with a tolerance for this increased risk.

As well as challenges affecting and causing change across the whole company, individual contributors face challenges in their roles and responsibilities. Contributors may wear several hats in their day to day role, develop new skills on-the-go to fill knowledge gaps within a team, or have a wider scope in terms of responsibility than would be usual for the same role at an established company.

So, how do companies in their early stages differ in reality from companies who have operated for many years?

- Processes might not exist in some areas of business, or may be informal, i.e. not documented.
- The pace of work may be faster, you may work on new projects frequently, whole teams or individuals might be expected to context switch more.
- Individual contributors may be expected to be more independent and self-starting.

When applying for a job within a startup you might wonder how it will differ from the experience of working at an established company if you have not worked for one

> Processes might not exist in some areas of business, or may be informal, i.e. not documented.

before. On job adverts you might frequently see the words 'flexible', 'self-sufficient' and 'challenging' - but what realities do these words actually represent? How is your day to day role going to be different compared to when you worked at a company that has been around forever?

Being an effective QA Engineer within a startup requires carefully applying previously learned behaviours and expectations. If you're coming from an established company, there may have been specific quality standards the company expected the product to adhere to in the form of processes and documentation. You may have your own personal standards that you believe a feature should meet. You might have beliefs about what is and what isn't acceptable when it comes to quality. Let's explore how this experience may affect your transition into the start-up

#### LET PRAGMATISM BE YOUR GUIDE

When you move into a start-up, your

external context changes, which means you may need to adjust your approach to protect quality within a team. The quality expectations which are applicable in the context of a company with plenty of resources and time, may come across inflexible and difficult in the context of a startup.

For example, it's important to Quality Assurance Engineers that we prevent as many bugs as reasonably possible from reaching production. Consider a situation where close to the end of a project you find a bug you believe should be fixed so it costs the business less overall. In the context of an established company it might make sense to fix this bug before it goes to production. But in the context of a startup, the immediate need of the business to release the feature may overrule the desire for a 'perfect' product. The reality may be that the feature is released and the bug fixed later on.

This is not an ideal situation, but startups operate in constantly changing environments that require flexibility. It's important to consider many factors in making this kind of decision such as: the severity of the bug, who it affects, and the impact.

You do not need to change your personal opinion about which bugs should be fixed and when this should happen, but you should consider challenging your existing expectations and reconsider if they are realistic and useful in this new environment. To challenge your existing pre-conceptions is a strength! Your personal opinion can stay the same, it's a strength to hold your beliefs in favour of quality even when others do not think the same, but you also need to know when to compromise when it's the right decision for the product and users. Take every situation as it comes and be balanced.

#### **EXAMPLES OF PRACTISING PRAGMATISM**

One example of an opportunity to use pragmatism is when discussing requirements for a feature with a product owner. You may disagree on the importance of catering to a specific, less likely user journey. You may be of the opinion that the journey should be covered, but the product owner informs you that due to time constraints and low resources (which are more likely to be a challenge in start-ups) it is not realistic to cover it now. It's important to continuously consider the wider context in these situations and approach discussions in a pragmatic and balanced way.

For a second example, imagine a situation where you find a bug which occurs only in a very specific situation. Let's say it occurs on a certain page, when a device is turned landscape, and only when that device has a specific screen height! After investigating product, you find devices with this height 
If you lead with always keeping the user's

TestBash **Testing Trends** This Week in Testing The Testing Planet (TWIT) **Community Newsletter Articles** by Simon Tomes Professional Masterclasse Membership Meetups **Bananas** Awesome members Community OnDemand Courses Certifications **Ministry of Testing** 

are popular, the user journey involving this page also involves turning the device landscape, and so this actually represents a significant portion of users. You may need to challenge the misconception that just because a bug is niche or an 'edge case', does not mean it's irrelevant and doesn't affect users. You may need to

Knowing when to be firm versus when to allow leniency will become very important.

persuade others to prioritise bugs like this and reviewing the device statistics for your by backing up your assertions with data. best interests in mind, you can't go wrong.

#### PRACTICALLY IMPLEMENTING PRAGMATISM

Knowing when to be firm versus when to allow leniency will become very important. Consider having a candid conversation with the three amigos, usually the QA engineer, developer, and product owner. It could illuminate whether a bug should or shouldn't be fixed. Changes like focusing more effort earlier in the development lifecycle to reduce the chance of bugs occurring at all may be beneficial. Use data to inform your decisions.

The phrase 'choose your battles wisely' is useful to keep in mind. We have touched on the time-sensitive and fast-paced nature of start-ups. In this context, it can be difficult to persuade others to your idea or solution if they aren't convinced of the benefits. Your views around bugs, project

**CONTINUES ON PAGE 28** 



Testers know the system better than anyone else. They know where the bodies are buried.

**LINZI CARLIN** 

#### 416 Request Not Satisfiable





**CONTINUED FROM PAGE 27** 

ideas, and feature improvements may be harder to implement considering these factors. In these instances, speaking to individuals one-on-one to understand their viewpoint and any concerns, gives you the opportunity to refine your idea (or revisit it entirely) and alleviate any concerns. By speaking to people individually and hearing out their concerns, and directly alleviating them, individuals are more likely to agree with your ideas. If you persuade individuals, then you persuade the team. It may be more effort to take this approach, however it may assist you in being successful at protecting the quality of the product in the end.

We should take care not to become overwhelmed by this context change and forget our main role as quality advocates. We need to educate on the cost of fixing bugs at each stage, correct misconceptions, and continually advocate for quality. Taking the time to think of new ways to be an effective QA in your unique context, to find the areas where you can practically apply yourself, will help you adapt to the challenges of your new environment.

HOW TO USE PRIORITISATION STRATEGIES TO MANAGE A HIGH WORKLOAD

When you join a startup, your prioritisation skills will be put to the test. There is a virtually endless supply of work, but limited resources to go alongside. Startups tend to be formed of passionate and hard-working, but small teams. You might be asked to help out with ad-hoc requests or other projects if necessary. With so much work to go around with a smaller number of people, it's easy to become very

When you join a startup, your prioritisation skills will be put to the test.

busy, very quickly!

Personally, my to-do list increased in size more often than I checked items off it. Start-up sprint teams may have a high ratio of developers to QA Engineers. And, if there are few QA Engineers across the company, it increases your likelihood of being called upon for meetings, advice, and general ad-hoc requests for help.

If the QA function was introduced to the

Engineering team after development had begun on the software then there may be a significant number of features which were released with only the basic functionality and user journeys having been tested. Now you, a QA professional, have joined the team with expertise in testing, it's likely they'll want you to take a look at and find any bugs in these legacy areas. This is in addition to the ongoing sprint tasks completed by developers, which also needs testing!

This results in an always busy QA engineer, constantly occupied, and then some.

THE PRIORITISATION SUPERPOWER OF SAYING

Saying 'no' improves your prioritisation skills, and helps the team more in the long run. Saying 'no' is a super power you can use to avoid your to-do list growing too large, to avoid slowly gathering action items that aren't really that important, and could be completed by another engineer on the team, improving efficiency all-round!

If you feel bad about saying no to a task you think you won't get around to, consider how by protecting your own time, you are making yourself a more efficient asset for your team. You are ensuring your expertise and skills are used on the tasks they are most needed for. Simultaneously, allowing that task to be picked up by someone with more time is a positive thing, as it increases the throughput of important tasks. It also allows others in the team to learn new skills and develop their knowledge of areas they may not otherwise.

Transparency builds trust between yourself and the others in your team. Others will respect your honesty if you are candid when talking about your workload. By saying 'no' and seeing the tasks you do agree to through to completion, it shows you are accountable and reliable. This is positive for your reputation within your team and your relationship with other team members.

On a personal note, it was difficult for me to accept that I couldn't get around to everything I saw as my personal responsibility, or as part of my role. I needed to assure myself that I was working on the most important tasks with my limited time. Clear communication about your workload within your team during standups, managing expectations in sprint planning, and receiving input from stakeholders on priorities on an ongoing basis, will help you remain assured that your limited time is being spent on the correct tasks.

PATIENCE: A VIRTUE AND A SKILL



#### Dear Agony Ant...

How do I assert my importance in my team without stepping on toes?

Scan the QR code for answers





At startups, processes, tools and documentation are not givens, as opposed to many established companies. After joining a start-up, you will probably be expected to help create new processes and select tools to help the team succeed! In my experience joining a QA team at a growing company was a rewarding experience, it allowed me to develop my skills and leadership qualities by developing solutions for the unique needs of a startup. I was able to use my prior knowledge from my previous experience and make an informed decision on subjects such as: what tools to use (such as, how I wanted to handle test cases); the formation of QA processes (such as triage and regression testing); and more technical decisions such as the approach to automation. Joining a team where there are not yet set QA processes or tools means in many cases you can shape them how you want to, and this is very empowering.

To revisit the wider context, you face the challenge of assuring quality in the early stages of a development team. This means there could be more challenges you face as a QA Engineer such as:

- · Issues you've resolved or otherwise avoided in a previous team might now occur more frequently and require finding a solution in this new context.
- · Early or existing processes might be imperfect and require close attention to attain improvement.
- · Engineers may never have worked with QA before, it may be up to you

to introduce them to the purpose of a Quality Assurance Engineer in a team. There may be gaps in knowledge around QA. Any misconceptions and misunderstandings may require correction.

It's true that these things can occur anywhere, but in my experience are more frequent at companies that are newer or growing quickly.

You need patience and the ability to deal with these hurdles in a proactive way. Finding solutions that work for your team will involve trying new ideas and occasionally these may fail. Which is okay!

You may come across many of these situations, but on the positive side, this means you can be a true asset in helping to resolve them and make things better. It is a good opportunity to get to know your team better as a whole and as individuals, understand their perspectives, and work together to form ways of working that fit the team.

#### PATIENCE HELPS REDUCE FRUSTRATION

Patience will allow you to face the ongoing challenges without becoming flustered or frustrated. Remaining calm and positive, by reminding yourself that it's a fantastic opportunity to be a part of helping a team grow, will be motivating for the rest of your

It is important to remain determined, and be an active participant in making dangers, and use your prior knowledge of dealing with problems and forming solutions, to help your team get there as smoothly and as quickly as you can.

Furthermore, a start-up being a new business in the early stages of operating with limited funds, may not have had a QA team for long (or at all!) prior to your arrival. This means the individuals within the team may not have worked with QA Engineers recently, or potentially ever!

Engineers may never have worked with *QA before, it may be up* to you to introduce them to the purpose of a Quality Assurance Engineer in a team.

Patience will be required in this situation too. Team members may not be familiar with what a QA Engineer needs in terms of information in order to work effectively, or they may not be used to receiving such in depth feedback about the quality of the features they deliver. If you are a knowledgeable and experienced OA Engineer, it's important to impart this knowledge to the team in a supportive way.

Use the tools at your disposal to fix the problems you see, understand the touchpoints you can use to make a difference:

- Raise points in retrospectives
- Feedback in channels
- · Talk with and educate team members in 1 on 1s

Ensure you do this in a kind, understanding way for your points to be received well. Starting difficult conversations is not easy, but it is the first step in resolving problems and in making the team the best it can be.

#### TO WRAP UP

This article is written from my own personal experience of previously working in established companies and moving into start-up and fast-growing business environments. My account may differ from your or others' experiences, but I hope the insights explored are useful. If you would like to get in touch and share your views, feel free, I'm always interested to hear about others' experiences!

Finally, to summarise the three skills:

- Pragmatism, being practical in the varied, possibly turbulent environment of a startup.
- Prioritisation, being red hot on ensuring your to-do list only contains vital tasks, protecting your time, and communicating openly.
- Patience, being mindful of yourself and your team whilst you work together to reach company goals. ■

improvements. Be on the lookout for

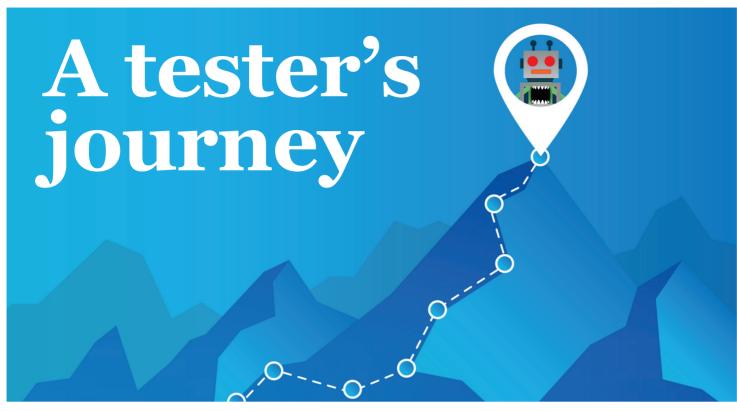
M T ATHENS Celebrating it's 6th year anniversary! ministryoftesting.com/meetups



We are 100% manual and every automation project failed. What is the secret?

THE TESTING PLANET, ISSUE 11, SEPTEMBER 2024

#### 508 Loop Detected



No one said our testing journey was going to be easy!

#### Uncover the inspiring journey of transitioning from a curious novice to a seasoned software tester, mastering automation and teamwork along the way

#### BY PETROS PLAKOGIANNIS

My passion for software testing started back in the early '90s when I was a child, with my first computer, the legendary Amstrad 6128 CPC. I remember I wanted to rip open the computer monitor and see what's inside. This was my first-time exploratory testing. When I was out of university, I had no idea what testing really was

I still remember my first interview in one of the most famous companies in the world (with a 3-letter logo). I rejected a position because I thought, "Software Testing? What is it? No, thanks." The next month I got my first job as a software tester in a different company.

Since then, I have worked as a software tester. To be more precise, I have been working as a professional software tester for 15 years. I have tried during these years to learn something new every day. However, I have to mention that the journey from being a junior tester, primarily focused on exploratory testing, to someone who can set up a test automation framework from scratch was long.

TEAMWORK AND COLLABORATION ARE

In Greece, back in the late 2000s, software  $\,$ 

testing was not so popular. It was hard to find information sources if you wanted to learn a new testing technique or an automation tool. This was an issue because, as testers, we all should improve our work continuously. A lesson that I learned is that teamwork and collaboration are essential.

I recall the day when a software developer told me about cross-site scripting (XSS) attacks.

I injected a script into the login form of a web-page, and magically, the page crashed! It was enlightening to see it. When I went back home, I stayed up all night trying to crash all the pages with XSS attacks. I found so many bugs that the next day I received recognition from my manager for doing good work.

#### BECOMING AN AUTOMATION TESTER

My first attempt to use automation tools was with the Selenium plug-in with Firefox. Many people ask why Firefox and not Chrome? The answer is that Chrome did not exist when the Selenium plugin was created. So my first tests were recorded tests using record and playback automation, without coding at all. I remember creating the scripts and then editing them to handle more scenarios. This was my first encounter with coding, primarily involving simple tasks such as tweaking keywords and names to align with

my requirements. I started automating my daily testing tasks to make my work easier. Seeing how automation skills could help me sparked my interest, and I wanted to learn more about it.

Subsequently, I had the opportunity to work with Watir (Web Application Testing in Ruby) at a small branch of a UK company in Greece. I was in a team with front-end developers who taught me a lot about JavaScript and the DOM of the

...you should not
worry about failures,
and believe me in the
beginning you probably
would have a lot.

websites. They gave me useful pieces of advice. It was there that I created my first scripts. This team was truly exceptional. Now, many of its members have risen to become managers, leaders, and principal engineers across different companies. I'm deeply grateful to each one of them for the invaluable lessons I've learned.

However, in my current company, I am enhancing my automation skills. I started

by using QTP (Quick Test Professional) with VBScript and then moved on to Selenium with Java. I've been tasked with setting up automation frameworks for various projects. I've traveled to places like Luxembourg for Eurostat, Brussels for the Commission, and Finland for the European Chemical Agency, setting up Selenium frameworks with Java each time. With each project, the framework became more robust and efficient. Now that I am a Cypress Ambassador, I have developed a versatile skill set that allows me to transition between Typescript / Javascript frameworks and Java-based frameworks easily. In addition to mastering automation skills, I've also learned about performance

At first, it was tough to understand how to test and optimize system performance under different loads. But with practice and help from colleagues, I got better. I remember a project where we faced performance issues. We worked hard to fix them by balancing loads and optimizing databases. It was tough, but it taught me a lot.

Now, I feel confident in my ability to handle performance testing for important projects. This experience showed me the importance of teamwork and determination in overcoming challenges. So my first piece of advice is as a software tester, you have to dare to ask questions, try to find a mentor that can inspire you, ask for feedback and help.

#### IF AT FIRST YOU DON'T SUCCEED, BE A CONTINUOUS LEARNER!

My second piece of advice is that you should not worry about failures, and believe me in the beginning you probably would have a lot. The first time that I set up an automation framework, my code was just ugly, what you would call spaghetti code. One memorable failure from my early career involves an attempt to practice with JMeter, a tool I hadn't used before. Naively, I added Google's site and set it to receive 100,000 hits. Little did I realize, being a junior software tester, that this would actually send 100,000 requests to Google. The consequence? Google promptly banned my company's site. The CEO's email asking, 'Who is this Petros?' swiftly followed. While it might sound amusing in hindsight, it was a serious lesson at the time

During an interview 13 years ago with a major UK broadcasting company, I faced a setback. The interviewer, by chance, a Greek Testing Leader, asked me two questions about Selenium, which I didn't know much about back then. She ended the interview immediately. But that rejection fueled my determination. I said to myself, "Prove them wrong", and I did it. Try to learn from your failures and mistakes, and gain experience so you can be better in the future.



Testing is amplifying fail signals.

MAARET PYHÄJÄRVI

#### 451 Unavailable for Legal Reasons



#### COLLABORATION AND CONTINUOUS LEARNING ARE KEY

I've refined my automation skills through diverse missions and projects for European organizations. With each project, my framework has evolved and grown more robust. A key aspect of my approach is actively engaging in discussions with developers, carefully scrutinizing their code and delving into the nuances of Object-Oriented Programming. This process demands countless hours of dedicated effort to ensure the quality of the code. Additionally, I actively seek opportunities to expand my knowledge by studying projects on GitHub and participating in online webinars to enhance my automation skills further.

The experience gained from asking questions to these experts is invaluable.

Also, I've had discussions about technical issues with renowned developers who created many of the frameworks we commonly use, such as Jenkins and Selenium. The experience gained from asking questions to these experts is invaluable. I vividly remember occasions like having dinner in Ohio, US with Angie Jones, or meeting Simon Stewart and Kohsuke Kawaguchi in Athens, Greece. It's an experience beyond words. Therefore, my advice is simple: dive in, write code, seek guidance, and explore the best solutions available.

#### FIND YOUR COMMUNITY

In the first years of my career as a junior tester, I always had in my mind how I can improve my skills. I kept asking why there is not a community or a support network of people in Greece that can help you and encourage you regarding software testing?

The years have passed, and one day I said to myself that I couldn't sit back. I had to do something about it. I found out about the Ministry of Testing at the Selenium Conference in London in 2016. When I arrived at the venue, I saw the banner of the Ministry of Testing. After some short talks with Heather Reid, I found myself longing to make more and more people in Greece aware of this community.

The decision to run the local meetup was taken. Now there is a Greek community where we all share knowledge and experience to teach and learn about software testing. If I have a question, I go to the Slack channel and ask.

My third piece of advice is the following.



Try to find your testing community. Go to meetups. Community is all about people. Meetups are about members of the community rooting for each other, learning together, searching for opportunities,

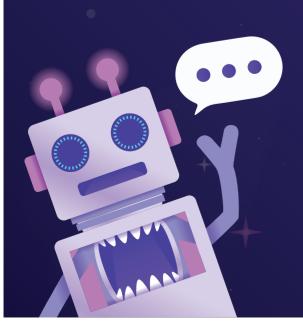
and helping each other find them. This is the biggest advice that anybody can give you. 85 to 90% of life is just showing up. Show up. If you show up, there's a chance something could happen. ■





Have you got your

## PROFESSIONAL MEMBERSHIP



Professional Membership, your one stop testing shop!

Quality engineering and software testing teams rely on our Professional Membership to stay relevant.

ministryoftesting.com/pro

# **Taming The Beast of** Irreproducible Bugs

#### Finding Opportunities in Chaos

#### BY RAHUL PARWAL

"Irreproducible bugs are a formidable challenge for software professionals. However, with the right mindset, techniques, and tools, you can navigate this complex aspect of your work and bring these elusive creatures into the spotlight."

Trreproducible bugs are bugs that are difficult or practically challenging to reproduce. Usually, we consider a bug as "irreproducible" if it cannot be reproduced consistently, even after multiple attempts. Irreproducible bugs are a nightmare for software testers and developers. They emerge and survive best in chaos and confusion, making it notoriously challenging to track down and fix.

The difference between a professional and an amateur development team is the ability to tame such scenarios.

In this article, I'll discuss various ways to break through the barriers and common excuses preventing the resolution of these unreproducible bugs. We'll delve into the art and science of debugging these elusive issues with tools, techniques, and a healthy dose of determination.

#### WHY DO THESE IRREPRODUCIBLE BUGS

Irreproducible bugs haunt testers and developers alike. They are like mysterious creatures that no one wants to waste time investigating, debugging, or reproducing.

Most developers resist fixing them for two primary reasons:

- 1. They can't replicate the failure ("it does not exist on my machine")
- 2. It requires too much analysis ("we are already short on time")
- 3. There is no helpful heuristic (a set of reliable principles or guideposts, like a checklist) to aid us in root-causing the problem. Keep reading for some suggestions!

#### BUT DO THESE BUGS MATTER?

Consider them the tip of the iceberg. Irreproducible bugs are warning signs of issues hiding beneath the surface. Addressing them can prevent more significant problems down the line. Issues that may underlie irreproducible bugs are:

- Hidden problems: Problems that are not in the conscious frame of the tester debugging the irreproducible bug
- Operating system dependencies: These include OS updates, patches, registry settings, system defaults, and the like, which are often ignored while debugging such issues
- Third-party library mismatches: Applications often rely on a collection of various third-party libraries. A version mismatch could trigger mysterious behaviors, leading to unreproducible
- User configurations: These are settings specific to each user of the application. Variations in configuration settings can result in unexpected behavior
- Unknown problems or unexpected **consequences:** Problems arising outside the direct focus area of the application

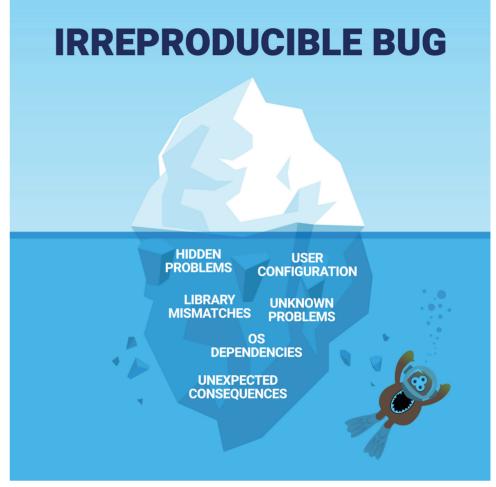
#### **DEBUGGING IRREPRODUCIBLE BUGS: CREATING A RELIABLE HEURISTIC**

When faced with an irreproducible bug, don't we all feel puzzled and confused? How do we even begin to think about tackling the problem?

However, it's essential to keep cool and follow a structured approach to get to the root cause. After a few tries, you and your team might even be able to come up with a set of principles, such as a checklist, for investigating bugs that can't easily be reproduced. These sets of principles are also known as heuristics.

#### THE BASICS OF ELUSIVE BUG SLEUTHING

- 1. Report irreproducible failures, but do it with extra care: Even if you can't replicate a bug, report it. Careful reporting is the first step towards finding a solution. Here's how you report any unreproducible bug:
- Mark the issue as irreproducible: Clearly label the bug as irreproducible to avoid
- Describe the failure precisely: Provide a detailed account of what happened. Include screen captures, screen 2. Look for follow-up errors: It's tempting



Observe what is hidden beneath the irreproducible bug iceberg

recordings, console logs, API calls, basically anything that might be helpful

- Pro Tip: Always record your actions when doing exploratory or scripted test execution. This will enable you to add relevant evidence when you report such bugs.
- Identify variables that matter: Note any specific conditions that could be relevant. This includes factors such as platform, browser, OS version, thirdparty library version used, hardware devices and peripherals (if applicable). information like messages, error codes, screen changes, other messages, and any relevant factors
- Describe your reproduction attempts: Explain how you tried to recreate the failure. This will acquaint the developers with the various attempts you have already made and allow them to plan their work accordingly
- Preserve application state and the steps taken to get there: Document the steps you've taken, as they might be crucial to the issue
- Look for configuration dependence: Consider whether different system configurations might be contributing to the problem. An unexpected version of an operating system, Java runtime, or browser can sometimes make a huge difference.

When faced with an irreproducible bug, don't we all feel puzzled and confused? How do we even begin to think about tackling the problem?

to report the issue the moment you find it. However, do some followup work first. Experiment with different data, behaviors, options, settings, software, and hardware environments. Sometimes, unreproducible bug can reveal itself when you vary something outside the immediate application you're testing.

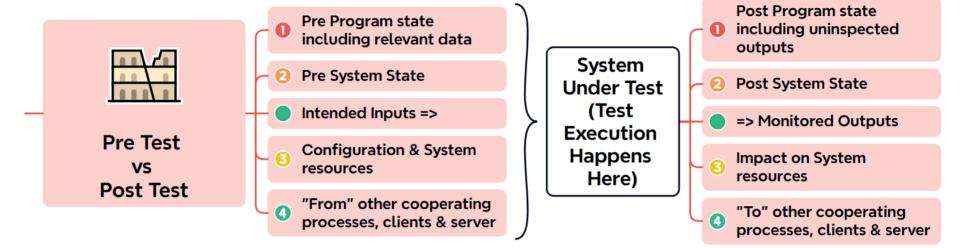
- 3. Seek help: Nothing will help if you cannot seek help. Don't hesitate to reach out for assistance from your team and beyond. Collaborate with programmers and inspect the code. Consult with experts in technical marketing, support, documentation and training, network administration, or power users such as the sales team. Ask questions like:
  - How many people might encounter



Yes indeed about the cost! The cost of automation is high, but what about the cost of NOT having test automation?!

**ELISE BROOKS** 





such a scenario

- · How serious is such a failure?
- Has the customer reported similar issues in the past?
- Have we noticed similar problems in any other similar product of our organization?

#### TIPS FROM PERSONAL EXPERIENCE

Having devoted almost half a decade of my career to this area, I would like to share these tips that have helped me become good at reporting and advocating for irreproducible bugs. Here are a few things from my experience to help you excel at this craft:

- Write down everything: Document every detail about the issue. Don't assume that anything is unnecessary
- Note system state before testing: Record the state of the system before conducting tests, including the defaults and preconditions
- Check your existing bug-tracking system for similar issues: Look for patterns or similar failures from the past. You never know if someone else has already seen such a thing before. This will help you strengthen your case
- Preserve the test machine's state: Save the configuration of your test environment. In case you are using a virtual machine or container, create a

duplicate image immediately

- Vary the timing for interactions: Experiment with timing to uncover hidden bugs. Timing is often the most ignored factor while testing. Vary your pace of steps through the application
- Assign resolution codes in the bug tracker: Even if you have to close down some unreproducible bugs, assign them a specific resolution code or tag for easy reference in case you find them in the future

#### PUT TESTABILITY AND OBSERVABILITY TO WORK

There is a lot that happens in the background even when a simple test or test step gets executed. Understanding the underlying conditions is crucial when dealing with irreproducible bugs.

Here is a depiction of various factors in a system before and after the test.

Leverage the testability and observability of your system to focus on the following:

- 1. Preconditions and postconditions: Document what happened before and after the occurrence
- 2. Program state and relevant data: Save the system's state, including data that are critical to the issue

**CONTINUES ON PAGE 34** 





#### **CONTINUED FROM PAGE 33**

- 3. System state: Record the overall system state, as it might be connected to the problem
- 4. Intended inputs: Document what your inputs were and your expected outcomes
- 5. Configuration and system resources: Analyze the hardware and software environment
- 6. Associated processes, clients, and servers: Check for interactions from external sources, resources, and processes

#### SOME HELPFUL TOOLS

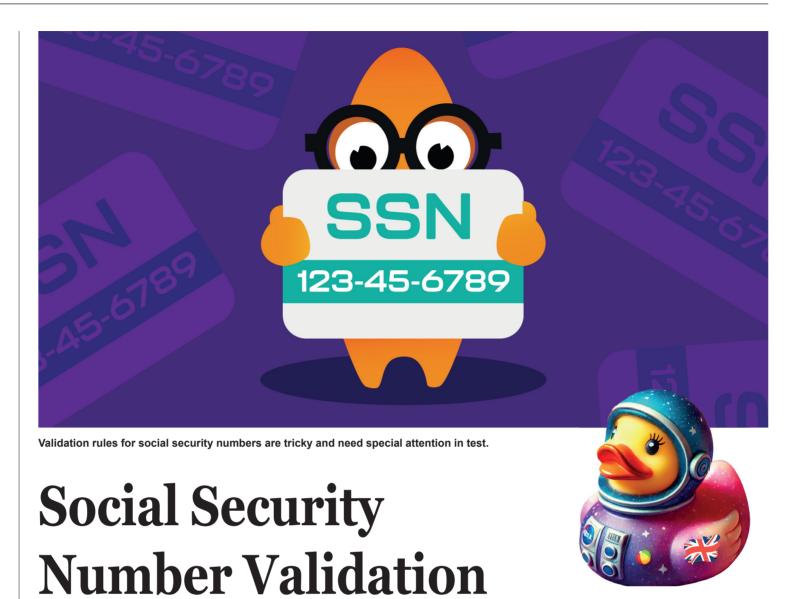
To aid in your bug-hunting expedition, consider using these common tools:

- Bird Eat Bugs: Capture and report bugs in real time
- **Details:**  Support Collect system information for better troubleshooting
- Wink: Create screen captures to document unexpected behavior
- Yattie: Exploratory testing assistant. Learn more about it in my article for Ministry of Testing
- Rapid Reporter: A tool tailored for testers to report issues efficiently
- Greenshot: Capture screenshots
- Win + R > dxdiag: A handy command line tool to get detailed information about your Windows system
- Free tail BareTail  $\mathbf{for}$ **Windows:** Real-time monitoring tool
- Wireshark Go Deep: Tool to track your network traffic.
- Licecap: Record images to capture subtle changes.
- Sysinternals Sysinternals Microsoft Learn: Collection of powerful micro-utilities.
- Dev Tools (F11 in the browser): Explore developer tools for web applications.
- Task Manager: Monitor system resource usage.

#### TO WRAP UP

Irreproducible bugs are a formidable challenge for software professionals. However, with the right mindset, techniques, and tools, you can navigate this complex aspect of your work and bring these elusive creatures into the spotlight.

Do encourage your fellow testers to share and discuss their own bug investigation stories, and create collective wisdom for bug investigators within your organization and project. ■



## A Tester's Guide to Uncovering Hidden Defects

#### BY NATE BOSSCHER

Tt's amazing how many ways you can **⊥**break software validation of social security numbers (SSNs), especially in the United States! It's almost like the government wanted to give testers more surface area. Jokes aside, SSN validation is really important to get right. Let's take a look at what's going on.

#### **INTRODUCTION TO SSNS**

Ok, so tell me more about SSNs (USA):

**SSN Rule 1:** SSNs follow the following format: ###-####. The first part is called the Area Number. The second part is called the Group Number. The last part is the Serial Number. These parts will be important later. For now, let's just focus on the format.

- Let's see what the validation rules are:
- · Try non-numeric characters like letters, emoji, whitespace, or extra dashes
- Try putting dashes in the wrong spots

- · Goldilocks: Too long, too short, big numbers, little numbers
- Try some number -parsing characters; for example. use exponent 1e10, -102,

The ###-##-### format is identical to the way it appears on users' government cards. This is important. Users have a hard time remembering all the digits at one time. The breaks in the format allow them to look up and down and start again. From a usability standpoint, your app would do well to mirror this behaviour.

Some applications will automatically inject the dashes for you. That's a whole other area for finding defects. Try backspacing, or try jumping back in the string and editing a number.

**SSN Rule 2:** The Area Number cannot be 000, 666, or 900-999. Let's test that

- Try numbers that supposedly are not allowed. They should fail validation.
- ranges such as 665, 667...

**SSN Rule 3:** The Group Number may not be oo.

SSN Rule 4: The Serial Number may not be oooo.

Rules 3 and 4 seem pretty straightforward. Let's test how our application handles them:

- Try 00, 0000 respectively. They should
- Try both oo and oooo at the same time (123-00-000), and check the error message. Sometimes if two errors occur at once, error messages can get garbled.

#### HOW JUNE 25, 2011 CHANGED EVERYTHING

Well, not quite everything, but the U.S. government relaxed some of the validation rules a little bit for SSNs issued on or after June 25, 2011.

Software development teams still have to keep their eye out for social security Try numbers near those reserved numbers issued BEFORE June 25, 2011 and implement the more forgiving

Find out what users need, not want.

**RACHEL WINTER** 

 $\mathsf{m}\mathscr{D}^{\mathsf{T}}$ 

standard for SSNs issued later as well.

**SSN Rule 5:** SSNs issued before June 25, 2011 may not have an Area Number of 734-749 or a value greater than or equal to 773. Area numbers were used to differentiate districts (for example, 001-003 designated New Hampshire residents) and those ranges were reserved as a result. Here are a few area numbers you can test with:

- 001-003: New Hampshire
- 691-699: Virginia
- 766-772: Florida

**SSN Rule 6:** SSNs issued before June 25, 2011 used a sequential Group Number. You can use this group number to validate the year the number was issued. For example, SSNs starting with 417-57 were issued in 2003 in Alabama. Here are a few examples you can test with:

- 1997 Alaska: 574-25-...
- 1994 Delaware: 221-86-...
- 2001 Kansas: 509-21...

Ok, so rules 5 and 6 might not be worth implementing at your organisation, but they could be useful for fraud detection. Before applying any of these rules you'll need to verify that the number was issued before June 25, 2011.

#### **PUBLICLY ADVERTISED SSNS**

SSN Rule 7: SSNs that are issued

specifically to be publicly advertised are not valid. In 1938, a wallet manufacturer thought it would be fun to insert a paper (pretend) SSN card in every wallet they sold. They used SSN 078-05-1120 in stores across America. This SSN was actually valid and belonged to the manufacturer's secretary! Since then, over 40,000 people have claimed to have that SSN. You can read the full story here. Let's test how our software validates the rule:

• The following SSNs are considered public and invalid. 078-05-1120, 721-07-4426, 219-09-9999

#### SSNS IN OTHER COUNTRIES: THE BSN

In the Netherlands they also have an SSN system called the BSN (short for Burgerservicenummer). Spoiler alert, there's not nearly as many rules for this one.

**BSN Rule 1:** BSN uses the following 9 digit format - #######. Let's test how our software handles the BSN format:

- Try non-numeric characters (alpha, emoji, negatives, whitespace)
- Goldilocks: biggest possible number (99999999), smallest possible (00000000), less than 9 characters, more than 9 character

**BSN Rule 2:** The nine digits must comply with the Eleven Test, which is essentially a checksum similar to credit cards. Let me explain with an example:

- BSN 323416391:
  - 3\*9 + 2\*8 + 3\*7 + 4\*6 + 1\*5 + 6\*4 + 3\*3 + 9\*2 + 1\*-1 = 143
    - We're doing #\*9 + #8.... #\*2 +
  - 143 mod 11 = 0 (mod 11 must return zero for the checksum to be valid)

If that is a bit too complicated for you, don't worry. You can use a generator to do all the fancy maths for you. Let's test it:

- Try a few valid BSN, try a few invalid BSNs
- Try BSNs with all zeros
- Try BSNs with big numbers (mostly 9s)
- Try BSNs with leading zeros

#### TOOLS TO GENERATE SSNS AND BSNS

It's tough to keep all these rules straight and actually get on with the work of testing real values in your app. Here are a few resources that can help you get started quickly:

• SSN Generator for June 25, 2011: This webpage has a great database of pre-2011 SSN rules you can test with. Choose your regions and years and it'll generate a nearly endless supply of SSNs.

It's almost like the government wanted to give testers more surface area.

- BSN Generator: This webpage generates lists of valid and invalid BSNs you can test with. It's pretty straightforward. The site generates mock bank account numbers by default, but if you flip the selector you can get the citizenship numbers.
- Testing Taxi Edge Cases: This Chrome extension incorporates all the "Let's test it" rules above and generates valid and invalid SSNs and BSNs. It also autofills the values right into your text

#### TO WRAP UP

Validation rules for social security numbers are tricky and need special attention in test. But they aren't impossible to navigate, especially with number generation tools and understanding of some gotchas. As a tester, if you're mindful of the potential pitfalls in this area and you implement manual and automated tests for them, you'll be making a great contribution to the quality of the product you test. ■









#### Dear Agony Ant...

How do I mediate the brewing war between automation and manual testers?

Scan the QR code for answers

