Talk to Article: Exploring Pipelines

Many things in life happen through serendipity - great bugs, inspiration to write, getting to know people. This chapter is a result of a serendipituous chain of events.

First, Jenny Bramble was looking for speakers for TSQA meetups. Her way of asking as if she was asking for help made me step up on that day and volunteer. I usually need inviting, with the particular brand of socially awkward I carry with me. I need to feel welcome, but helping someone does the trick.

Second, on that particular day I was tracking down a complicated build pipeline and realizing how much in common my particular style of exploratory testing and the work I was doing on connecting code with code had in common. In the moment I labeled a new talk idea "Exploratory Pipelines" and threw it at Jenny with a few others.

Third, the conversational nature of TSQA meetup. Some sessions give you more than they take, and this was one of those. I'm using the energy from people I recognized in the audience, with special mentions to Jenny Bramble, Tristan Lombard, Christian Feldt and Darrel Farris.

So, writing the slides out to a particular form of an article with the flow of the story like the flow of the talk was intended. Since this talk, like so many talks of mine, is likely to never see a second chance, this is a way of accessing the content in an article format.

Slides for Exploring Pipelines

Exploring Pipelines is about putting together two things that clicked for me on one day at work. As I was digging my way through a particularly annoying and unnecessarily complicated pipeline on Jenkins jobs with various rules of when to run and where to start, I started recognizing that the efforts I was going through had a strong resemblance to exploratory testing. Exploratory testing being my favorite activity, I came to the idea that perhaps this connection could be useful to others as well.

In order to make the connection, we need to talk about both things: exploratory testing and pipelines. Finally, we'll talk about a few different organizations and my experiences there to show that the theory of today may be a practice of tomorrow.

Slides for Exploring Pipelines

Exploratory testing has been around since 1984, when Cem Kaner first coined the term. Even if we like to think same words mean the same thing for different people, it is particularly not the case with this word.

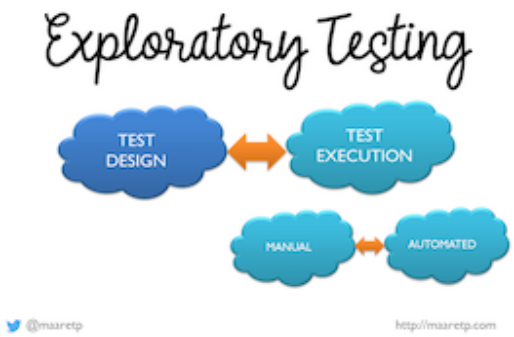
What it means for me is that we intertwine test design and test execution by avoiding splitting those two into two different people's heads so that whoever is doing the testing gets to learn from executing a test and design differently for the next test. It's about that silent moment inside your head where you ask: what did I just learn and how it makes my next action different?

The pace in how quickly we move between the two activities is controlled by whoever is doing the testing. We can take as much time between actions as we need. Or we can intertwine them so that in some moments we can't tell one from the other, while other times we are clearly spending all of our energy just on one.

To emphasize learning, the in-head connection between the activities makes a difference. Whoever does exploratory testing needs agency, the possibility to make decisions allowing what they learn to influence their choices, instead of following their own plan, let alone someone else's plan.

We do this to optimize results and investment. For the time we spend exploratory testing, we want our results to become as impactful as they can.

Remember, the core to this is agency that enables learning. And for that, we can't split this work for two different people, we would seek different dimensions in splitting of exploratory testing when we need scale.



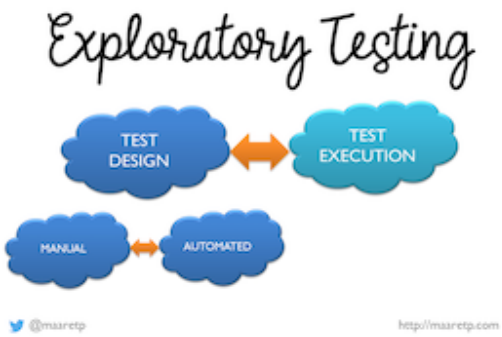
Slides for Exploring Pipelines

Now that we established that for exploratory testing to be exploratory, we can't split design and execution, we need to talk a bit about test automation in exploratory testing. It comes to my attention at work, with various organizations, that people somehow think that even if we can't split the design and execution, we could do that for the dimension of manual vs. automated in execution.

Let me just talk you through an example from office in just the last weeks. There was a feature around registering devices into cloud with a certificate that my team was building. While it was frustratingly open-ended not being able to see task close a few times a day, I'm very happy with the way the feature was tested. Usually every night the tester would leave their tests running, to come to a day of analysis of the results the next day. There was a part of execution that was automated, generating files on disk. There was a part of execution that was manual, finding trends from those files. And there was definitely the exploratory testing loop to designing different variables for the next night for deeper understanding of ways this feature could fail. Over a few weeks on that one testing task, the tester run thousands of registrations, covering many variables that no one could list at start of the task.

So you see, it makes very little sense to try and define exploratory testing as manual testing in the part of execution. Even if we think of our executed automated tests as regression tests, whenever they fail, their execution has a manual element of going in figuring out what needs to be different.

If you are looking at exploratory testing, with design and execution strongly linked, you can't separate automation in the execution. Even the creation of automation is a manual task. Or like I learned last week, humanual. I really like the ring to the term from Erika Chestnut.



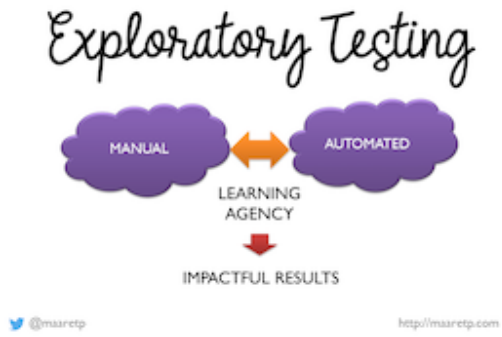
Slides for Exploring Pipelines

Manual vs. automated however isn't part of only the execution part. We are also working to figure out, and have been for decades in the academia, automating parts of test design. If we split design consideration into coming up with an idea of what to do, and how to tell if the result is right, we can easily already give examples of activities that are both manual and automated within the design.

Think about approval testing. Approval testing is this idea where you design your actions and most likely automate them, but you decide on your results based on humanual element of recognizing acceptable, and approving it for future baseline.

Think about model based testing. We design our tests with images, boxes and arrows that we map into pieces of code. Our model defines where we should end up. Our code moves us there, and runs checks on where we are. We can argue that we are automating one aspect of designing of the tests, but not all of it.

You may be seeing a pattern here: if you can't split design and execution in exploratory testing, and you can't split manual and automation in either of those two, perhaps what we mean by exploratory testing is that there's the same kind of relationship with manual vs. automated than with design vs. execution?



Slides for Exploring Pipelines

There indeed is. If you are separating the manual and automated, you are not living true to what exploratory testing frames to be. You are removing agency from the person making those design and

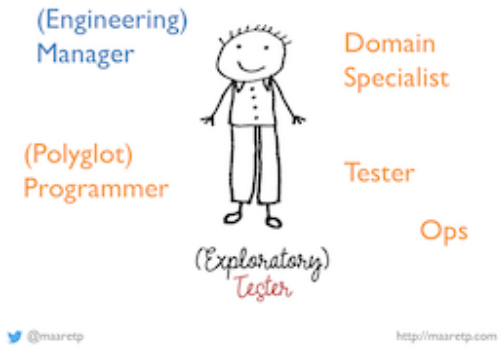
execution choices to the best testing they could do. You are not enabling learning between activities. And it shows up in your results.

I call this idea contemporary exploratory testing. I have come to it with experiences with colleagues who automate but design their tests poorly, and are not learning to optimize their value just as much as with colleagues who refuse to automate emphasizing manual testing, and replacing the time they could be using in automating with arguing with management on how that is not a reasonable expectation.

I believe we need to frame our testing differently. We need to ensure that we have agency in whoever is doing testing, and that agency shows up in them learning while performing whatever design, execution, manual, and automated activities they need to get their work done for impactful results.

Not all testing is exploratory testing. Look for the broken links in activities that inherently belong together.

We have been through a bit of a discussion on what exploratory testing is and we have not yet talked about pipelines, but let's just say this now: pipelines are automation. Understanding that automation is inherent in exploratory testing is important.



Slides for Exploring Pipelines

I may have intimidated some of you with images of having to take on work beyond your scope today at the office. While I talk about agency and sustaining connection between activities that need learning in between them to optimize their value for whatever investment we have, I am also well aware that the scale in which we create software does not fit in a single person's head. We need teams that share ownership. We need to accept that in an industry that doubles in size every five years, half of us have less than five years of experience.

Looking back at my 25 years so far, I started off learning the parts where programming was not in the center. I conveniently allowed myself to forget that I started programming when I was a teenager, and that the university degree I studied for is computer science. I embraced the problem domains, and learned about finances, legal, psychology, sociology, agile, teams, and communication. I grew in being a tester, with solid critical thinking and observational skills, and a researcher skillset to design experiments optimizing learning. To do testing, I often needed to set up my own environments, build my own components with the changes intended, and operate both during development and during

production the complex technical environments just to get my job done. I changed jobs to figure out what testing looks like in different problem domains and business constraints.

I rediscovered my polyglot programmer identity only in the last five years. And while that identity was hidden from me - by myself and no one else - I would look at problems prioritizing what I could do with the skills I had and appreciated. Rediscovering that I code, and that I can work on code even when I don't write it all alone enabled new dimensions for optimizing the time I was investing into being good at my work.

Particularly, I learned that moving around picking up skills - being even an engineering manager and boss of a group of developers - all plays to skills that enable me now as a principal test engineer doing the best possible testing I can with teams I share the work with.

I remind you all on the power of "not yet". You don't know something - yet. And every single one of us is already useful while we are on a journey to grow our impactfulness.

Slides for Exploring Pipelines

This all sums up to a vision that I want to share with a metaphor. The metaphor includes food and animal products, and if that particular area is a source of discomfort, I suggest skipping forward the next slide.

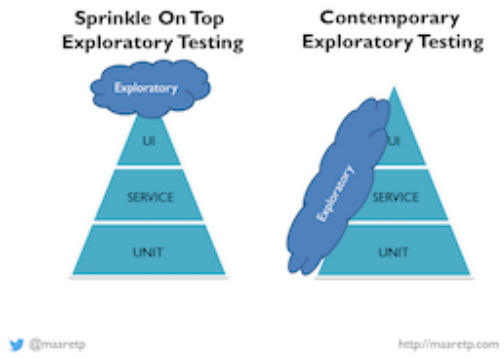
Slides for Exploring Pipelines

What I look forward with contemporary exploratory testing is impactful results, and higher value.

Sometimes people say “testing”, as in all testing is exploratory, but it really is not. There’s varying degrees of exploratory.

Like if you look at two pieces of beef. One is a regular american beef courtesy of the internet, and the other is Japanese waguy beef. If you enjoy a steak on a grill, you would probably enjoy the cheaper version too. The grease that is visible is part of the taste we expect and we don’t want to take it out. Like testing. But the grease in the latter is more evenly distributed thoughout the beef. This particular style makes it more valuable. You could no longer try to cut it out, it is now integrated. And it makes the entire result more valuable.

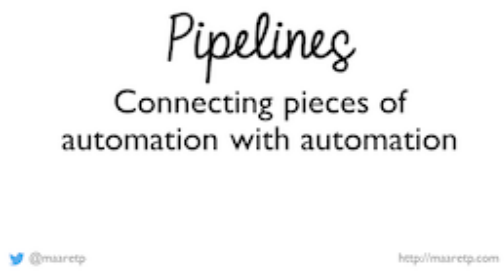
There’s room in the world for both. But with contemporary exploratory testing, I look at moving from the lower value category to the higher value category.



Slides for Exploring Pipelines

The same idea with a less controversial image is how we place exploratory testing on a test automation pyramid. The sprinkle on top kind improves the result and is definitely better option than not having it sprinkled on top. But the kids I share a vision on, contemporary exploratory testing, cuts through all layers and drives learning in all testing we do.

Exploratory testing is more valuable when we allow it its intended place as an approach to testing, instead of turning it into a technique applied on top of everything else. It is exploratory testing in both formats, but the results and practices we would observe are essentially different.



Slides for Exploring Pipelines

Understanding how I view the relationship of exploratory testing and test automation, as two sides of the very same coin, leads us to discussing exploring pipelines.

The way I define pipelines is that they are automation that connects other automation. A lot of times this pipeline automation is *more important* than test automation, and yet to much of our conversations on automation remain on the side of tests.

Let me illustrate this a little more.

Slides for Exploring Pipelines

In building a course where one of the many constraints of exploratory testing is documenting with test automation, I follow people work though creating sometimes the first automated test of their lives. Watching this unfold, makes it very clear it is an exploratory activity where to create automation, we are really exploring all the way.

From writing a single line that opens a browser to the page where our test target is, we learn with every step through designing something, executing it and allowing our learning to change what we would do next.

A usual progression is from a single line, to a few, seeing that the test can fail and the tools work, to having our first test scenario written down in a format that a computer can execute. From the single test, we move to parameters and using test templates, turning it quickly into a way of documenting

multiple tests we can run though the automation tooling. Adding enough, we find a problem and see tests fail for real reasons. We decide what to do with the failing tests and leave it failing, make it pass, or turn it off. And complete our activity by covering a spec with automation, and adding whatever values we design, as well as cover all the tests in multiple browsers in a timeframe where we could not do the same scope manually.

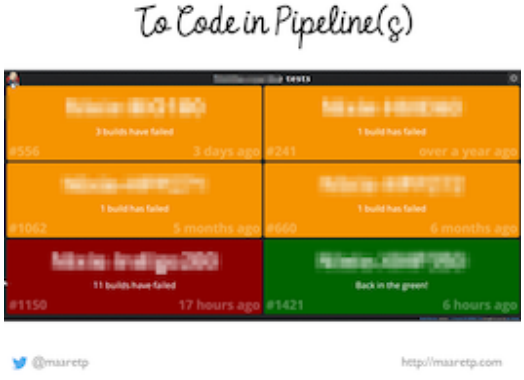
This is test automation, but it is not in a pipeline. The human remains as the person starting the run of automation on demand. And if we left along the red failing test, this set would not serve us well in a pipeline always reminding us that it is still broken until it gets fixed, potentially hiding other issues.

Slides for Exploring Pipelines

So we need to move from a piece of code doing testing for us when we ask to piece of code that does testing for us on rules we have defined.

Looking at this example I use as illustration, the tool in question is Robot Framework and I have a split relationship with it. On one hand, I teach it to new people who have never automated. On other hand, I propose never using it as soon as you're ready for a general purpose programming language.

In the picture, test 6 fails because the program does not work as it should by its specification.



Slides for Exploring Pipelines

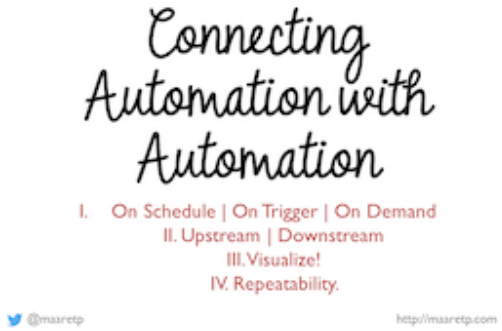
Dragging our code along into the pipelines, the other end of this is human, team and passer-by

manager understandable traffic lights that report - radiate - our status on close to real time. When the code runs in a pipeline, results show what works now.

Back when pipelines were new, a lot of people set up all kinds of fancy ways of alerting on problems. I've seen led displays reporting "N days since last accident" style encouraging the always blue/green state, to lava lamps slowly turning up until a name of the guilty person is revealed, to sound effects calling everyone's attention.

In recent years, we have become more boring. But we put a lot of thinking, particularly exploratory thinking, into what boxes would communicate something that truly helps us. From having any boxes to getting to boxes that are meaningful for the state of the team is usually a path you get through, exploring what would make it better than it is now while respecting it is already doing some relevant work for you.

My previous organization was into blue (with enough color blind people around), and with my current organization I am getting used to green again, and just working towards having less colorful displays in general. That is a culture change we often need to make it through in pipeline building.



Slides for Exploring Pipelines

From that single script, there is glue automation in the pipeline that makes the results continuously available. Remember, we defined pipelines as connecting automation with automation, what are the key aspects to explore then for this type of automation?

Looking at the learning I was going through when exploring our pipeline, I came to four major realizations I want to share today.

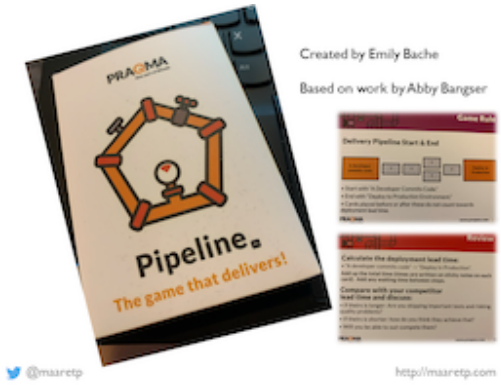
First, we have our choice of when to run something automatically. Coming to an organization that is heavily invested in a concept of nightly build, I can appreciate the daily rhythm even if I have grown addicted to fast feedback. But when I have nightly, I can start turning that to twice a day, once an hour, whatever schedule the resources - and I do not mean people but computers doing the work - allow us to work with. Doing things on schedule gives us a known repeatable point of measurement, and we would particularly resolve to it when we have external dependencies we can't watch over but want to keep an eye on. On trigger would work when we can follow a rule of something happening first, making it essentially relevant to refresh our results automatically. And no matter how we end up scheduling, we can always also go and just say that *now* I want to know,

on demand.

Second, we work with the concept up upstream and downstream. Thinking of things as flow of actions, something happens before, and something can only happen after this has completed. Externalizing that logic out of your tests and into your pipelines - while making pipelines code too - is essential. There is a rich source of exploring right here. What really need to be your dependencies, and how changing them will change the experience you live with.

Third, I can't deal with things that are not pictures. When exploring pipelines, I find myself drawing boxes and arrows. I have a clear favoritism towards tools that make it easy for me to visually see what the flow is. The boxes also allow me to point at something and ask: what if this would be different?

Fourth, existence of pipelines in repeatability of building our software cannot be overemphasized. Sometimes you test the result. Sometimes you test the thing that creates the result. And when the result gets refreshed and recreated multiple times a day, testing the thing that creates the result is a better choice. In recent weeks, we have built pipelines to build a three dimensional matrix of same but different in 3x5x3 dimensions. If I had to test all those 45 end results, I could not deal, even with test automation around. Pipelines visualize the difference, and create a baseline that is repeatable.



Slides for Exploring Pipelines

To get started in playful exploration of pipelines, I recommend trying out the pipeline card game created by Emily Bache. Knowing that Emily created the game inspired by a simulation by Abby Bangser at European Testing Conference I used to run makes me even more happy this game exists.

The idea of the game is quite simple and great for educating teams. You build a pipeline from committing code to deployed in production. You are given a rich set of options, and asked to design your pipeline, and calculating the cost (in wait time) of running your pipeline for different kinds of organizational risk profiles.

Caring for the cost in wait time ties to the accelerate metrics of lead time, noted in research to be linked with general company success.

You can choose the sprinkle on top exploratory testing, but that costs you time in the pipeline. I invite you to think about ways of integrating it everywhere, and remind you that some of exploratory testing we do in production looking at improvement we could propose.

Slides for Exploring Pipelines

I have been through organizations, changing jobs. The three last organizations have each been very different in regards to pipelines.

The first one, two jobs ago, is one where we introduced a build pipeline but absolutely no automated tests. We moved with repeatable pipelines into a process where we could control quality at start of work better on both value and risk, and move from releases happening only a few times a year to releases happening daily. The secret to testing them is to keep features hidden until you are ready to reveal them. Continuous delivery without automated tests is possible, even a worthwhile goal.

The second one, my previous job, had come to a pipeline driven development organization. You could always follow a pipeline. You could trust in making changes in a complex system with a lot of components, and the pipeline delivering your changes to the next release candidate. The pipelines were an organizational memory that enabled working on things built before us. And they captured more than just application code and test code - they also captured infrastructure as code.

The third one, my current job, has a good baseline of pipelines but having experienced the previous, I have many wishes of advancement. This motivates me to rebuild pipelines of my dreams, learning to change my dreams with my team, and find again the future I thought I once had.

I believe that the future is already here in our industry, it is just not at all equally divided. Appreciating what we have today, knowing we can have something else tomorrow is a foundational idea.

Slides for Exploring Pipelines

For me, the strategy for automating starts with the idea of incremental, incomplete, learning. What we have is a good baseline even when it is not that good.

Small flows of value, always for better, create big changes over time.

Even the ones of us who think we know how it should be done would do well in allowing the team to learn together to get to something the team can run with, even when you are gone.

Slides for Exploring Pipelines

Finally, as a departing thought to leave you with. Pipelines are a core of how we automate automation. In this process, we turn things into code. We put it in version management. And we call that shift to being able to see what we have as code - or configuration - DevOps. And that changes testing.

We move from black box to more hints on what have changed. We move from lack of control to some control. And that ability to have a little more control over our systems is what enables the increased complexity and speed.

Slides for Exploring Pipelines

I enjoy connecting with people, and love a good conversation. You may notice I like my work. I also like talking about themes related to my work. I started speaking to get people to talk to me. I talk back, and I invite you all on a journey to figure out how we explore our way into a better place for software creators and consumers.