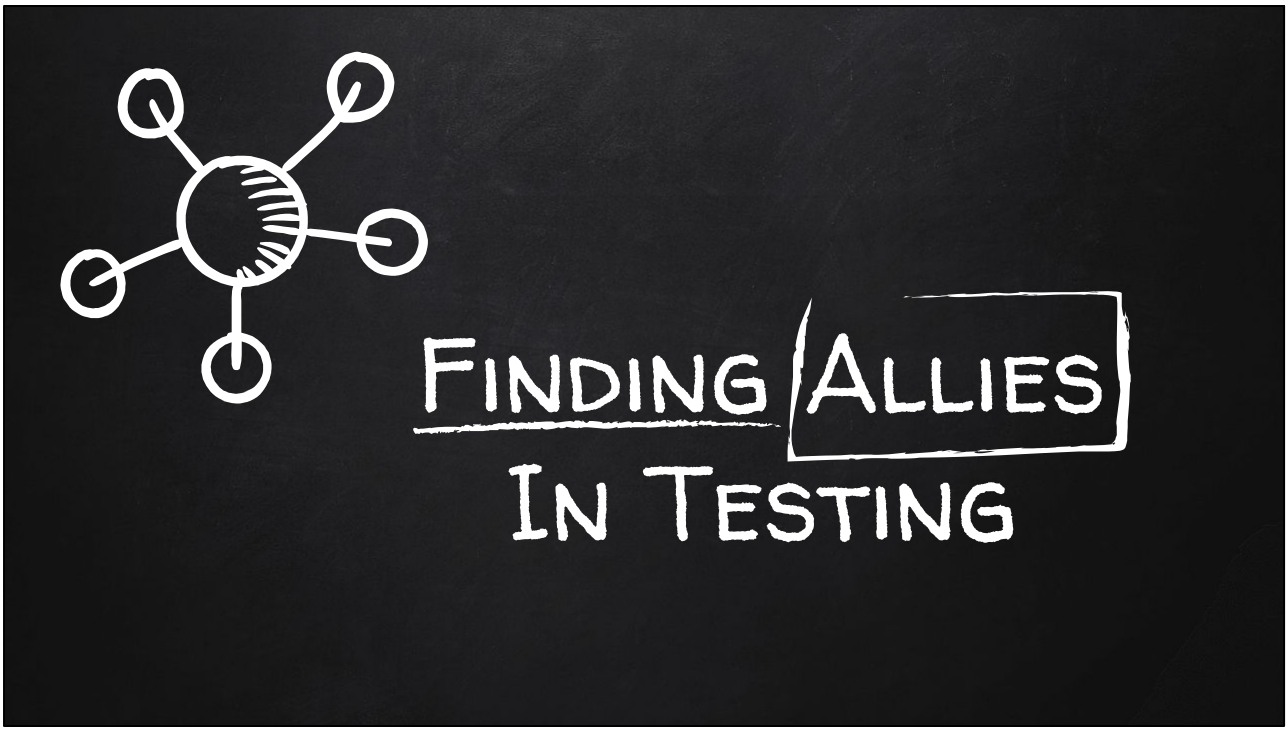

Finding Allies
In Testing

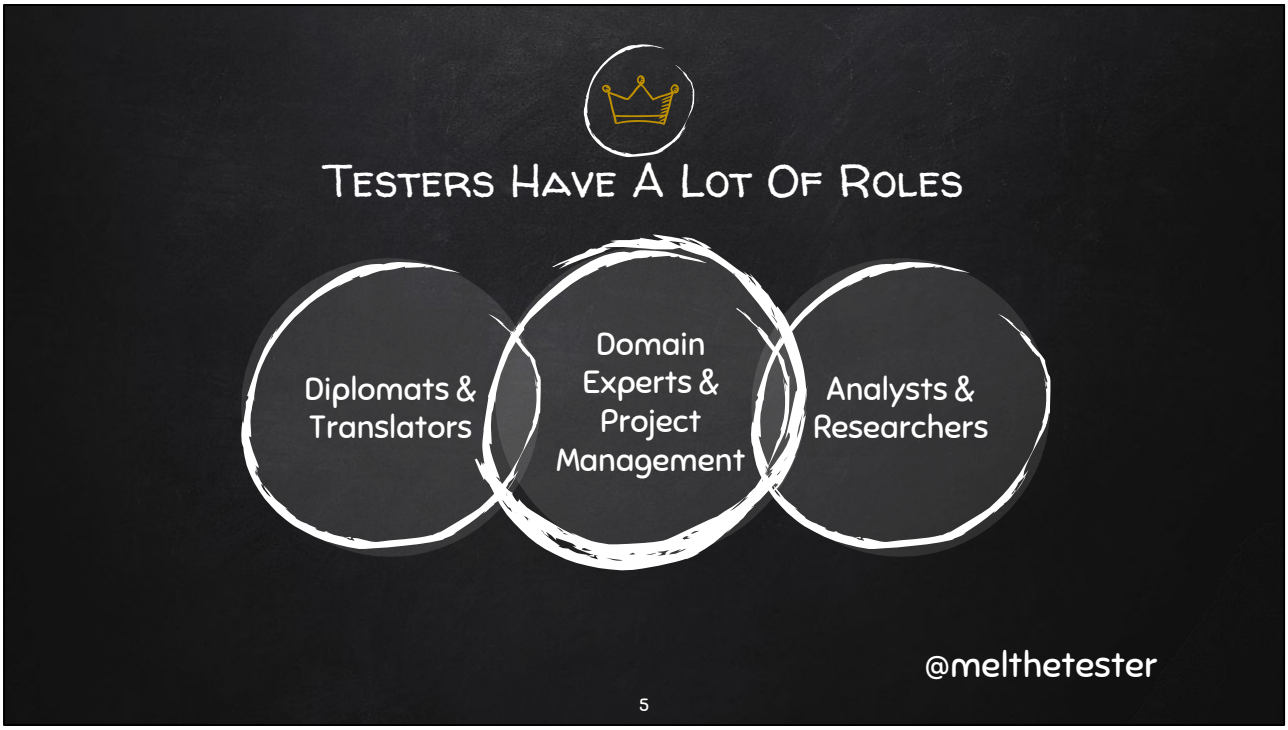
Working for ThoughtWorks
Working with Ministry of Testing (ministryoftesting.com)
I write blogs!

When I first presented this talk, I wasn't aware of the DevOps movement. I wasn't sure what to call what I was doing, but I was sure it was working, and I was able to gain a lot of information in a short period of time. Information that helped stop defects, threaded in customer concerns, and created a culture of quality. All of this was by trial and error. A lot of it was and still are methods I find very useful.

Katrina Clokie wrote this book. I read it, and it was like she was reading my mind. I was already practicing DevOps, but I didn't have a name for it. Her book explains the concept very well and gives the overall picture of what a DevOps culture looks like for a development group and even at a company level. Today, I'm going to give you some ideas to make your DevOps culture work. These are things I've been doing for years as a tester, in various combinations. I try one thing, it doesn't get me as much information as I want, I move to another idea. I keep coming up with ideas because my job is constantly changing, even more so now that I have moved into consulting delivery. In my whole career, I've rarely worked with the same tech stack, the same automation framework, the same process structure.

Additionally - this is a guide to Modern Testing as developed by Alan Page and Brent Jenson. I absolutely agree with this approach to testing. I think testing is evolving and we have to look at different ways testing can engage in different environments beyond what people consider (however wrong) what the job definition is for testers.

These ideas are ways to get to this and help mitigate bottlenecks instead of trying to solve them.



The basics to how you approach testing does not really change, what changes are the technical aspects of the job. Being able to adapt quickly and gain knowledge is paramount to being able to help produce a quality application.

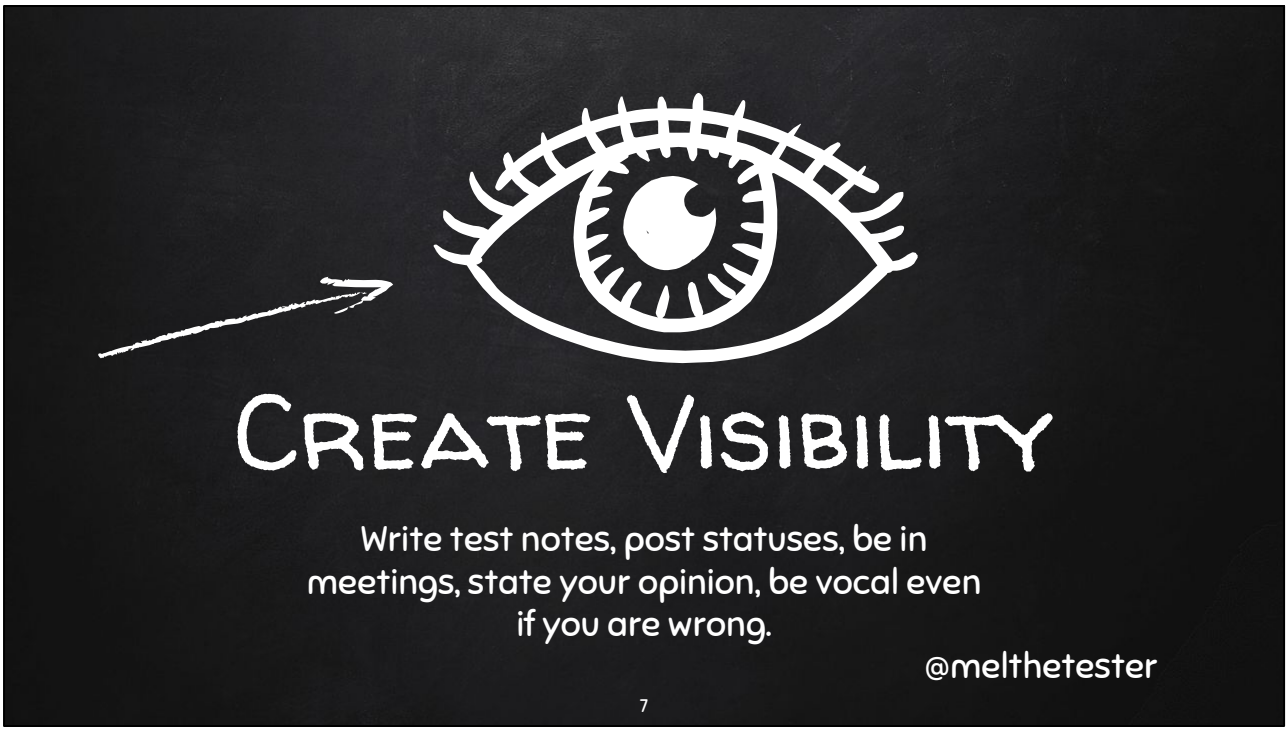
I've broken up the techniques I've used into Community, Team, and Self. If any of it sounds like common sense, or really obvious, that's awesome, you are already on your way to thinking and working in a DevOps world.

DevOps in particular forces someone who practices testing to learn a lot of different skills sets, play a lot of roles. These different skills and roles allow us to work our magic and help every other role or department we interact with.

The first one we are going to look at is Community...

When I talk about community, it's somewhat akin to culture, but I think culture doesn't really cover exactly what we need to develop. We need to be good neighbors. We need to help the customer service team, the sales department, the marketing department, customers, CEOs, CTOs, CFOs… all the C's - help them help themselves. When you create a community, you get less silos, more meaningful interactions, and more information flow in general.

Visibility is the key.



I think this is the key to everything. If you are content with only testing, and you are worried about being phased out, this could very well save your position. The only reason a company phases anything out is because they can't seem to find value in the work you do, or think they can automate it away to someone else.

Once you become involved in meetings and conversations which matter, people start to realize testing means something more than making something functional. It means caring about the whole complex environment you are trying to work in. You can create large effects on an organization by making things visible.

If you are already doing test plans and test cases, that's good - but how can you make those documents easy to read, visible, and accessible for the whole company? Simplify them to a report email - send it out to everybody. Put automation and test case execution on a wall. Let people see it. Put risks on a board and let people read them. Call things out. In meetings, in side conversations… talk about things people aren't aware of.

A great example of this is when I was working for an electronic medical records company and I had been there maybe two weeks. We were discussing reworking a rather physiology template of a form and folks were wondering how long it would take. I mentioned that there were at least 3 other forms that particular template was used, and because the app was still going through a refactoring process, this might affect several areas they might not have anticipated because they were focused on one form. The director of development was impressed. I had done my research. I knew

exactly what they were trying to do.

There was another time, with the same company, where we were doing a data migration for an old set of forms. We wanted the data to be available, because regulations stipulate that the information has to be available online for at least 10 years, however the government was requiring new fields and a new version of the form which required a change in the table structure of how we were currently save the forms in the database. The initial solution the developers and DBAs suggested was a bulk transfer into the new tables. I immediately spoke up and said that it should be triggered as an on-request process. We'd archive the tables, and only move data to the new tables as requests came in to see older forms. This would allow for a faster transfer process and data integrity to be maintained. I was immediately told I was wrong, and that it was too hard to do. I tried to argue a little more for the solution, but eventually it dropped. Two weeks later, my manager came to me and told me they were going ahead with the solution I suggested instead. It was the thought of having to test thousands of documents for accuracy that went through my mind. It forced me to speak up and talk about something I was pretty sure was technically possible. I was able to test the transfer of data, verifying with some very simple automation, rather then verify groups of documents, in addition to creating automation which would have needed to compare tables of information, and would have taken much longer to run.

This is what I think visibility means are things like this.

Another great method of being visible is make the testing position a service. If someone has a question, if someone needs me to go to a meeting, or create a video, or images to help explain a new feature to customer service, I can do that, I have done that… I will continue to do it. Is it testing the software, no, but is it gathering feedback and helping others that can help you make better software - yes!

If I'm available when the developers aren't: I can give someone information. I can direct them to the right person. I can direct them to the right team. I need to know these things anyway. It's even better when you can help other people with that information.

You can do so many things when you start thinking of your job as a service rather than running test cases. I've been able to do any number of things because I've volunteered to do them, to learn from others, and help others do their jobs.

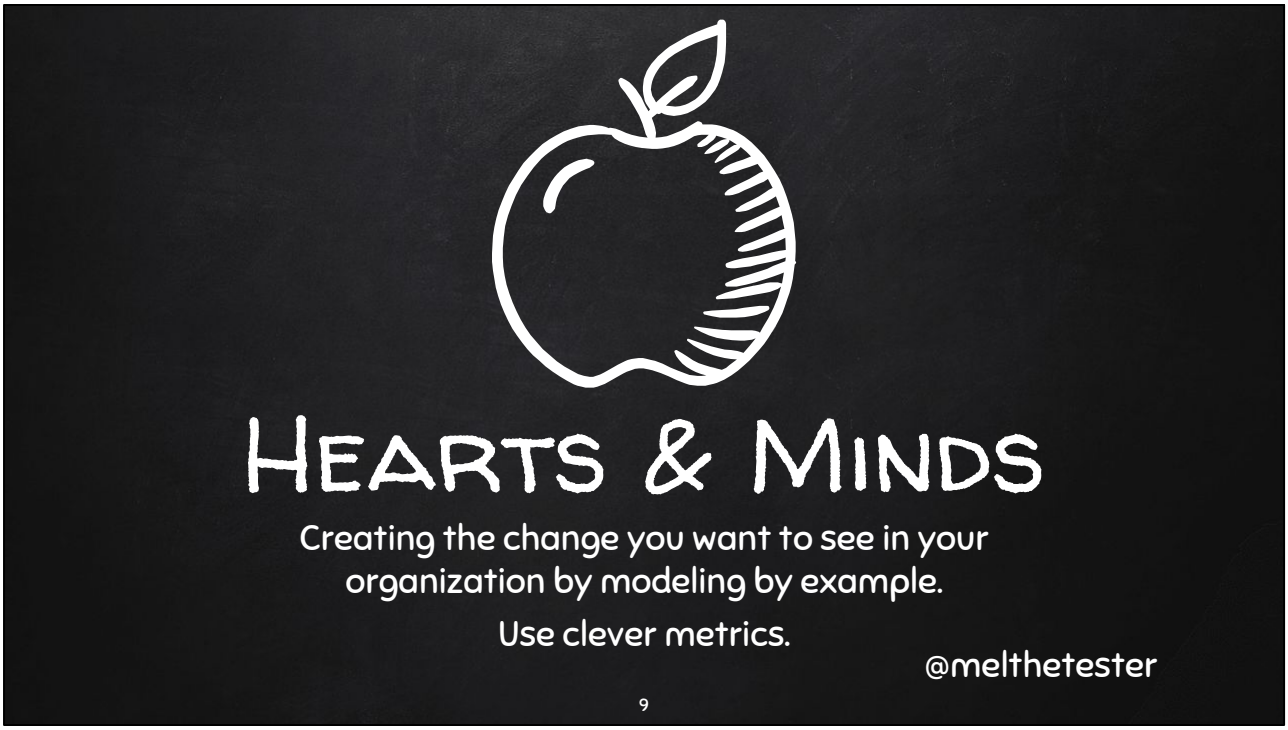
When you start to create an channel of information, it takes time, and it also takes trust. If someone comes to me to see if a particular environment has a defect, then I can do that research. I can help them. Pushing them away or telling them to use the proper channels is crazy.

I worked with one company that would freak out if I walked up to a developer and tried to ask a question, or called a meeting to go over a feature. They also freaked out if anyone came up to the testing department and asked for help, as if a production issue wasn't our problem. If customer service was getting a lot of calls and they wanted

some help to figure them out, we were told not to.

I've worked in a completely different environment where those things were encouraged and if something was wrong with production, everyone pitched in to assess and understand where it was on our severity/critical chart and react appropriately. Not every defect is a fire situation. Having an "on call" rotation that includes a tester is actually fairly good because a majority of testers are going to understand the severity before a lot of other people will. Having a protocol in place, and visible to everyone allows for defects to be handled correctly across the whole organization.

Being a service opens up channels of communication. It allows trust building. It means if other departments trust the testers, then they trust your work. They trust that you are going to do the best work you can. It's an amazing feeling when you or your team earns that trust and you can feel like you can trust others. It can be a huge morale booster too.

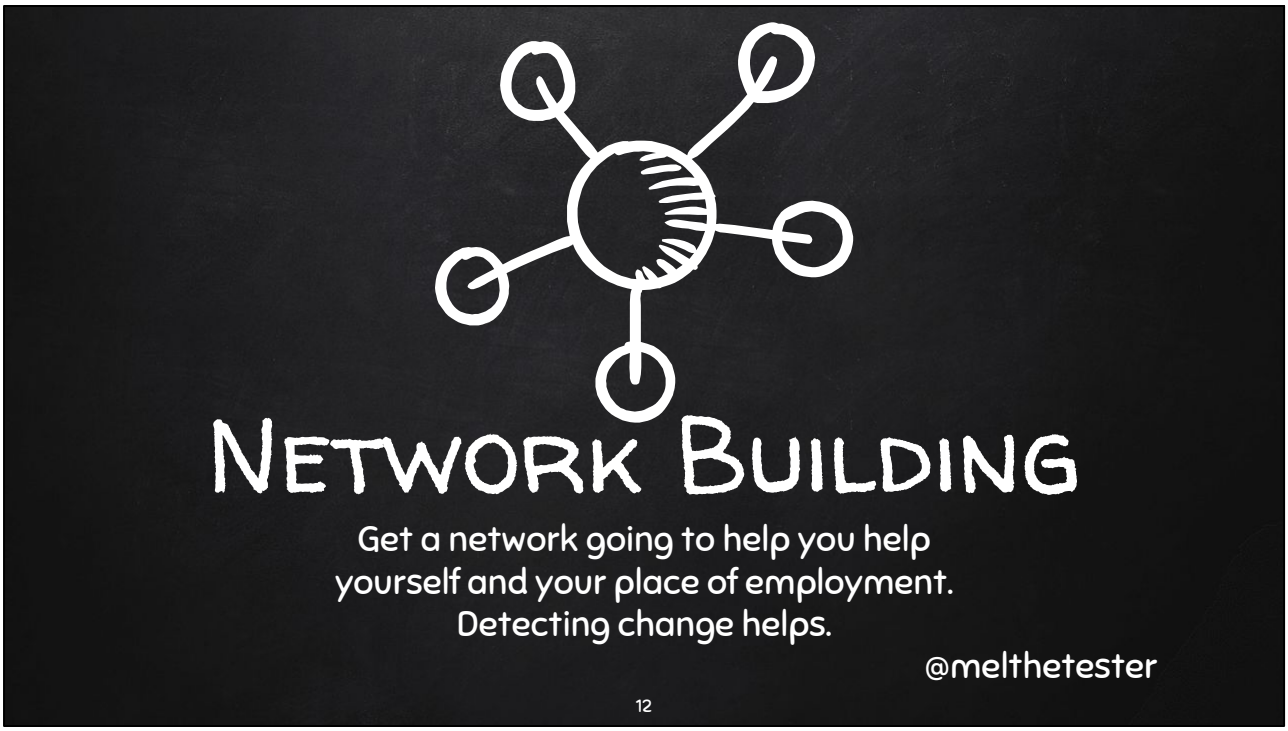


Speaking of morale, this comes along with being visible and creating a service. If you are doing this right, building your community right, you get the added benefit of seeing that community turn into something that generates quality instead of only using it as a guideline. People will want to work there. People will want to be part of the community because they see the positive impact and also get that in return.

You can start digging deeper in to problems - not just the functional ones - but the human ones. The ones that directly impact your working environment and your customer.

Monitoring and metrics play a big part in this - when I can stop being supercritical around functionality working and worry less about causing regressions, and I can focus on targeted testing based on metrics and monitoring. I can focus more on the non-functional aspects of the application, like usability and accessibility, which can generate more customers and therefore more feedback and better interactions.

When we focus on outcomes and metrics, while maintaining stability, a whole new world opens up. That only happens if everybody is visible about their problems and goals, and everybody is willing to help everyone else achieve them.



And when I talk about everyone - their problems and goals, I'm not only talking about the company you work for, but the customers and clients that use the software. Get out of the building and talk to people in the domain you work in. The company will have one opinion about how it works, and the people on the ground using the stuff will have another.

If there is a domain or industry conference, go to that. Gather information. Gather their pain points. Bring that information back to the company. Volunteer to work at the marketing booth for the weekend and listen to people talk about what they are looking for and what they need. You can bring that information back to the development group. You gain understanding of your user in a very unique way.

Other ways of doing this are to visit client sites. I recommend this completely if you work in a software company that has a domain where they are building enterprise software. Health care and education are two I can think of at the moment, but I'm sure there are others. Figure out how they are using the software. What parts they are using more than others.

You can gather metrics from the software sure, but you can't get a customer's opinion about a feature. Surveys can do that, but that doesn't tell you what the person's day is like, or let you see how they interact. Optimizations can happen with surveys and metrics, but the most meaningful ones happen with observation. If it's something someone uses every day, having someone in to observe them and get their opinions can create all kinds of awesome data and create understanding and caring.

Sales people could do this, they do this a lot actually, but you'd be surprised how a client changes when they realize they have a developer or a tester observing them in the field, asking questions, willing to listen to issues. It takes time, sure, but it's valuable, it's insightful, and often all you have to do is ask to go.

Let’s refocus a moment. Let’s talk about how we can build that community from a team perspective. What would need to happen to get a team on board to create or take part in some of the things I mentioned around building a good community.

While I’ve given some high level ways to develop a community, there are immediate actions you can take towards creating it where it might not exist. These focus around teams as they could be team activities or activities where some participate then bring the information back to the team or take to other teams.

Lots of these things are already in play inside of companies, it’s when people actually start using them in unexpected ways is where the magic happens.

I was a the same electronic medical record company I mentioned earlier and they offered a billing training in house. They wanted to do a dry run of a new training method and originally, they invited some people, but when I found out about it, I got myself invited. Management wasn't sure it was a good idea - they wanted me testing, but they let me do it because I had done other things equally as waky that we'll talk about in a minute which proved to be very helpful for them, so they let me go.

The trainer was talking through the training and about a third of the way through, she did something with the software which had never really been communicated to developers or testers before. She demonstrated how they were training people to use it, and right there, in the training - I discovered that the whole development department was passing around bad workflow ideas. It was the worse game of telephone imaginable.

I practically sprinted down the hall after the training session to my own team and showed them what I found out. Jaws dropped.

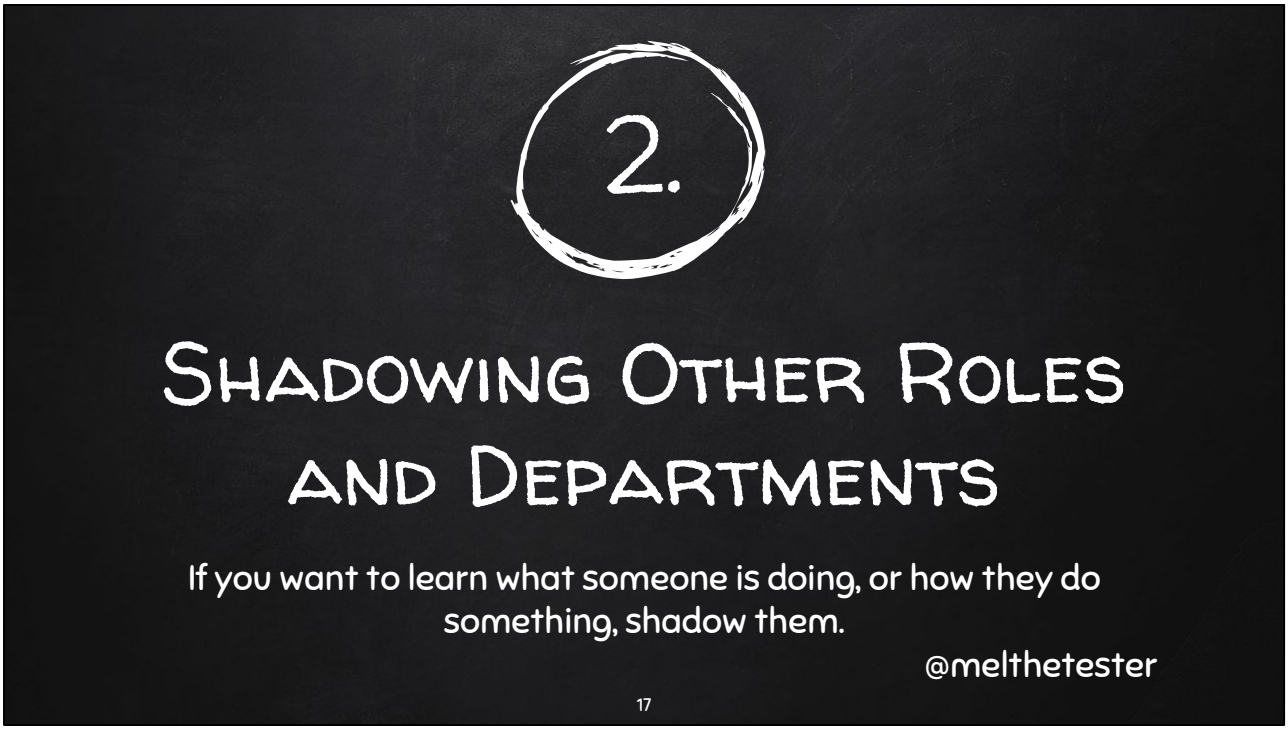
We immediately updated the automation we had in place to handle that feature to be more user flow correct. We also started planning additional checks to cover other bits of the flow we hadn't realized users were doing.

Manual testing of the feature changed immediately. Test cases were updated.

It was as if we had unearthed a treasure. It also affirmed that doing trial runs of

trainings were important for the development group. They scheduled more trainings and allowed anyone from development to go.

It was a fundamental shift in culture simply because I wanted to find out more about the application I was working with and I wouldn’t let them tell me no.



This idea is the reason why I was allowed to do the previous one. At one point, the emr company considered it part of the learning development effort and it was noted in your review if you participated in the shadowing program.

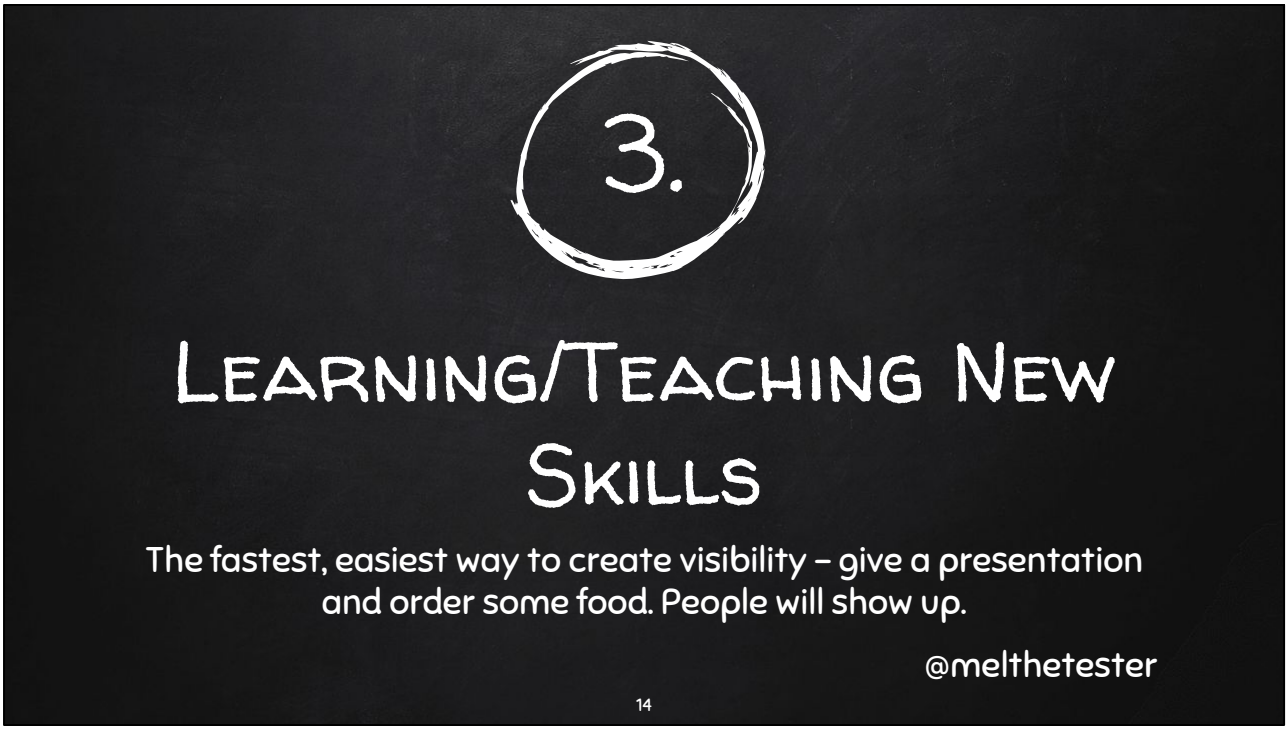
It's now one of the first things I like to do when I'm hired by a company is meet the customer service folks, if they are available(getting access to customer service is sometimes like getting access to Fort Knox). The next thing I generally do is try to sit with them for a few hours, listen to phone calls or listen to their general concerns and ideas.

Customer service folks are also pretty good testers too.

At another company, I had permission to have the whole customer service department execute a large test suite against a new enterprise version of education software. This was a few years back. We had automation, but we also had a lot of custom work and edge cases. We wanted to have a solid summer release, so the whole customer service department got into it and they were great. We found defects, we did a lot of cross-browser testing which helped find other defects we wouldn't have found otherwise. We gave the team a bunch of incentives to make the extra work worth it. They got the extra benefit of learning the new version of the software, which is something they had been asking after for a while, and earned prizes at the same time.

If you are interested in how someone works UX, DBA, Developer - any role you -

could use pairing or mobbing too.



Lunch & Learns, Book Clubs, Test Fests - all of these things contribute to visibility around testing, and letting folks learn new skills, whether that's opening up dialog with folks they haven't met before to mentoring and sharing amongst peers, it's a really easy way to get your information out to other departments and let them give you feedback on what you are doing. Every little bit helps.

I have started several book clubs and encouraged people to start their own. I've done a lot of lunch and learns and test fests - every time I learn something new. These things encourage folks to share their own journeys. Everyone can learn from that.

Start making small meetings or lunches with everyone you'd like to get to know.

I was recently talking with a peer about his lunch schedule. He regularly sets up lunches, not with the people he works directly with, but those around him or those that are a step or two removed from him and his team. He reaches out to other departments and checks in with people, grabs coffee, makes chit chat.

It might look like unproductive water cooler talk, but putting a name with a face can be just as important as testing.

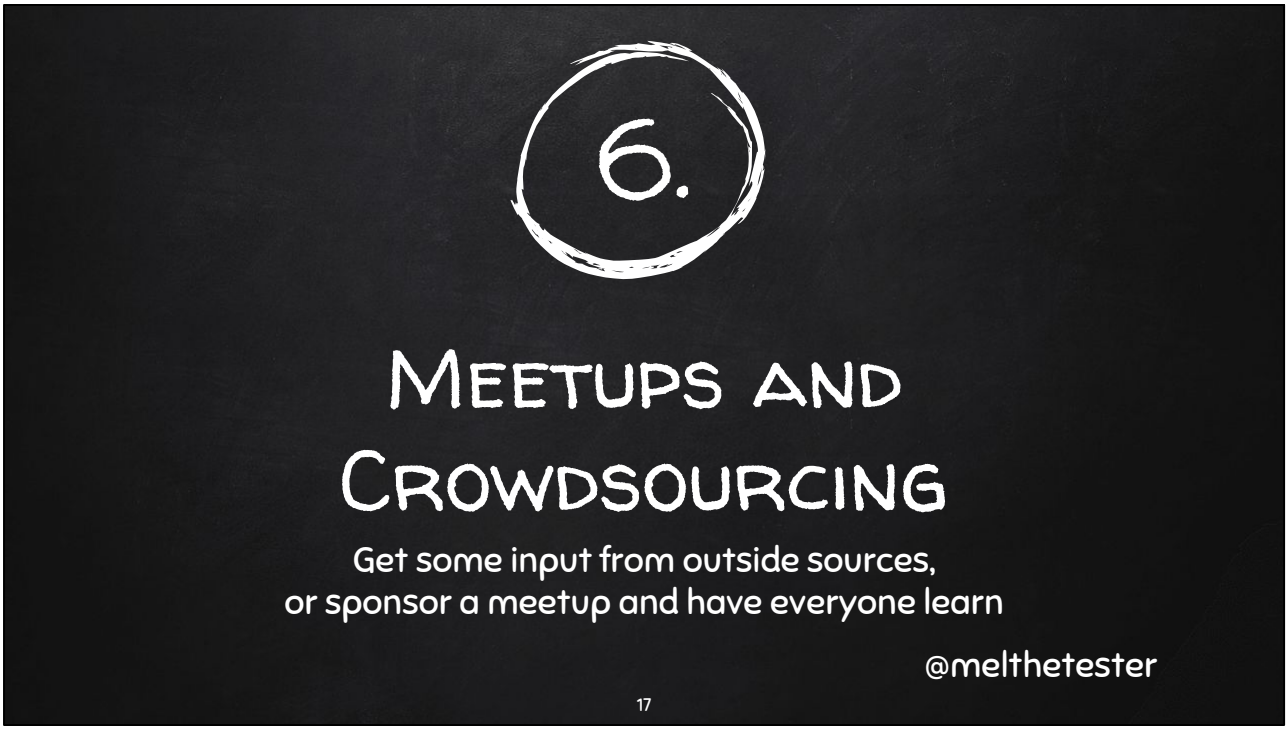
Going to meetings you feel you need to be at should be acceptable too. If you have an open door policy in the office, all the way up to the CEO, stop in, pop by, see if they are busy or want to chat. Sometimes saying hi is enough. It doesn't have to be scheduled like a lunch. Getting to know someone over coffee or a quick walk around the office works too.

Meetings are another great way to get to know other folks in a company. Going to other team's standups are good too. Take advantage of these things. Pop into meetings you don't normally go to every once in a while. You might have to justify it, but often if you do this in a way that's non-disruptive, at first at least, then people won't mind and they get used to the idea of you being there, listening. Then eventually helping out.

It's a way for me, to make my meeting times effective. Going to meetings with the BA

or PO is also good. It serves as a way to be a sounding board or a backup. If they are stressed and they need another set of ears or eyes to help them out, volunteer to go. Your technical knowledge in those areas could be extremely valuable.

This goes with #4 - even if it’s saying hello, getting to know people never hurts. If you’re an introvert you can spread that out over time, don’t push yourself too hard out of your comfort zone, but try, just a little. It makes a world of difference. People recognize people they know. People give projects to people they have met before, even in passing. It’s worth the time.



I think meetups are a great way to learn about new information without having to travel very far. There are all kinds of meetups these days covering a variety of topics. I follow a lot of meetups wherever I'm working because I want to get to know the community around me and what it's like.

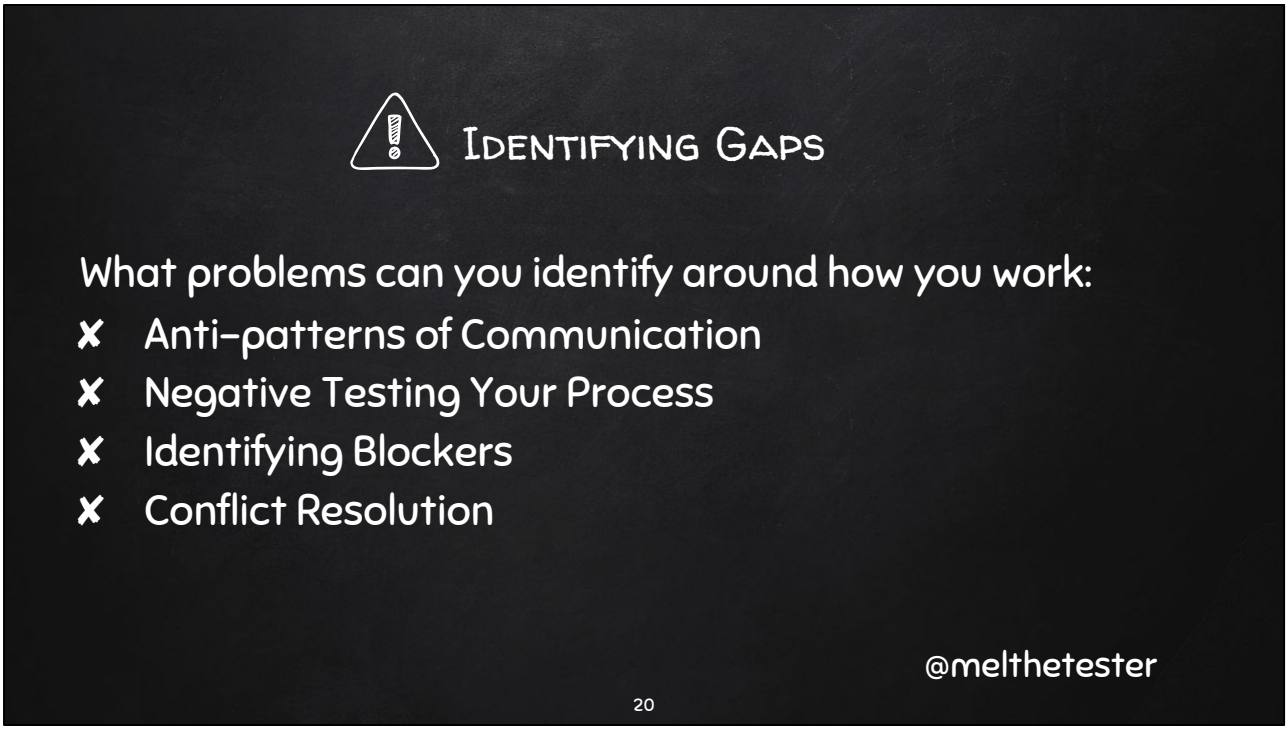
Crowdsourcing via social media is another awesome way to get access to new info and ideas. I read a lot from the tech folks I follow on Twitter and LinkedIn. I hop on Slack and ask questions in the channels. Ministry of Testing has two great Slack communities and I am also a member of Women Who Code and I am on their Slack as well.

Change can start with yourself. It can start small, and it can sometimes be an act of rebellion. The famous Grace Hopper quote: “Ask for Forgiveness rather than permission.” is definitely something I live by. If I know it will be helpful and it will work, then I do it. I set the example rather than asking someone if I can make an improvement to my own working environment.

Conway’s law ties into this:

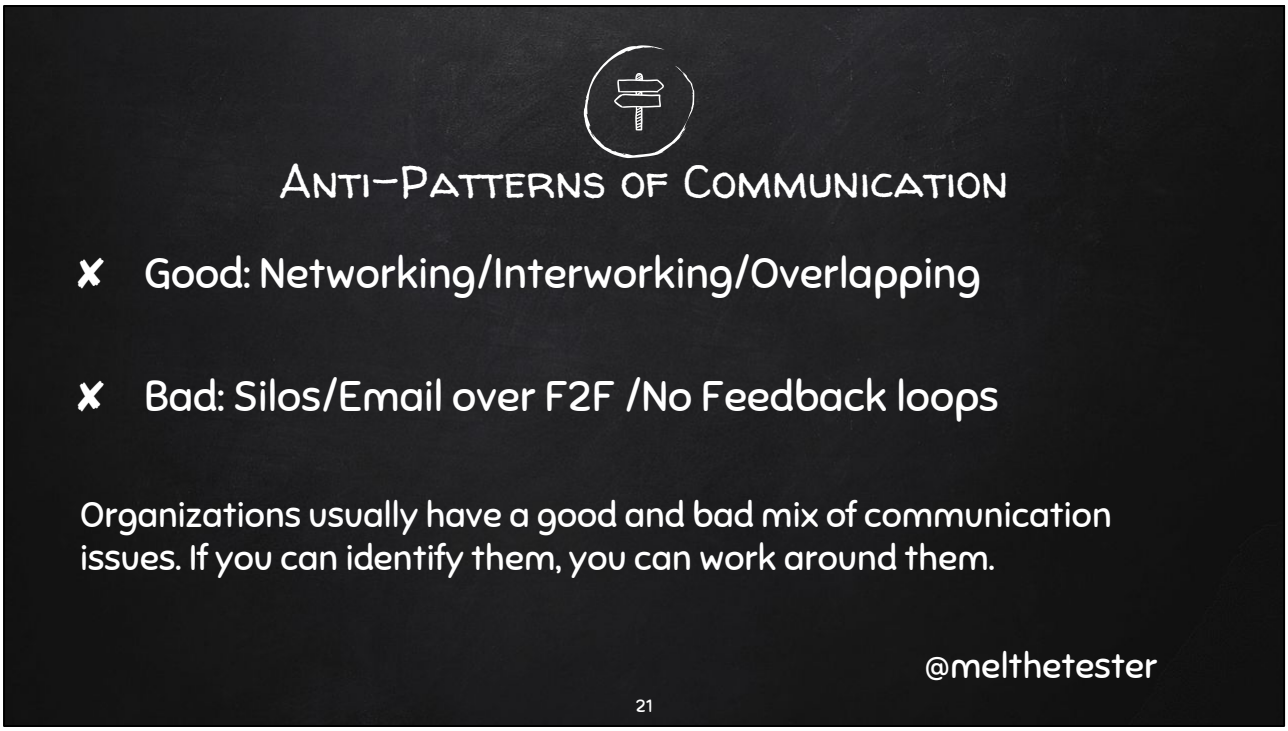
http://www.melconway.com/Home/Conways_Law.html

Communication structures affect software design. That's the basic gist of this definition. If you want to gain more quality and better software, sometimes changing how you communicate with peers, team members, and management is necessary. You have to recognize the structures and understand the positives and negatives of that communication and organizational structure to yourself, and your team. If you can make small improvements on communication, you can have a butterfly effect on quality of an application in general.



These are a couple of things you can do at an individual level. These seem like huge things, but they really aren't. You can affect small changes to big problems. Those small changes add up over time. Those changes effect differences. Don't try to eat the whole problem, you might become paralyzed how epic some of these problems can be. I know I would be. Sometimes I still feel pretty helpless when I know about these things and I'm not sure how to approach them other than to lash out and be angry.

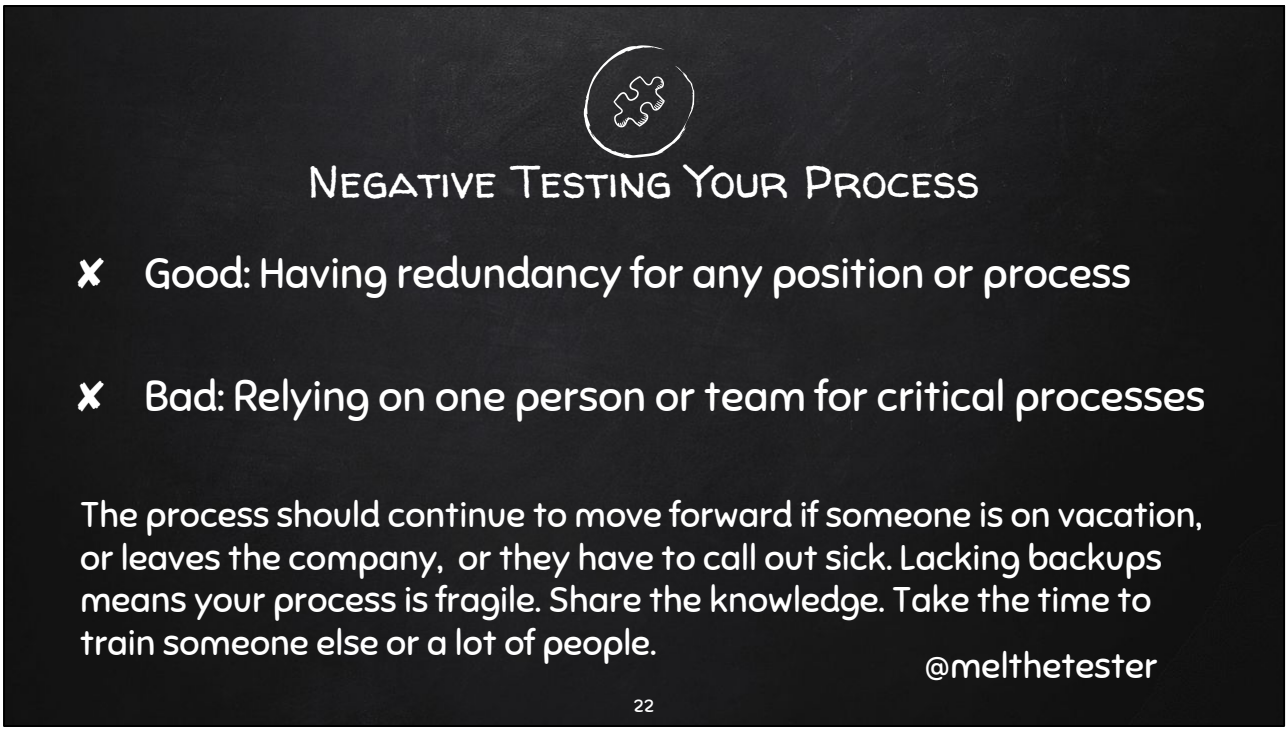
Let's talk about each one of these and some ideas about getting past them.



Sometimes having conversations in person can fix a lot of things. Having feedback loops help a lot too.

Someone asked me yesterday about a problem they were having with a developer being really grumpy when they tried to ask questions. It's not fair to them, which I completely understand. My suggestion was to ask the developer what kind of communication method they preferred, and what kind of communication method he is willing to accept if it's an urgent matter. That's only one idea.

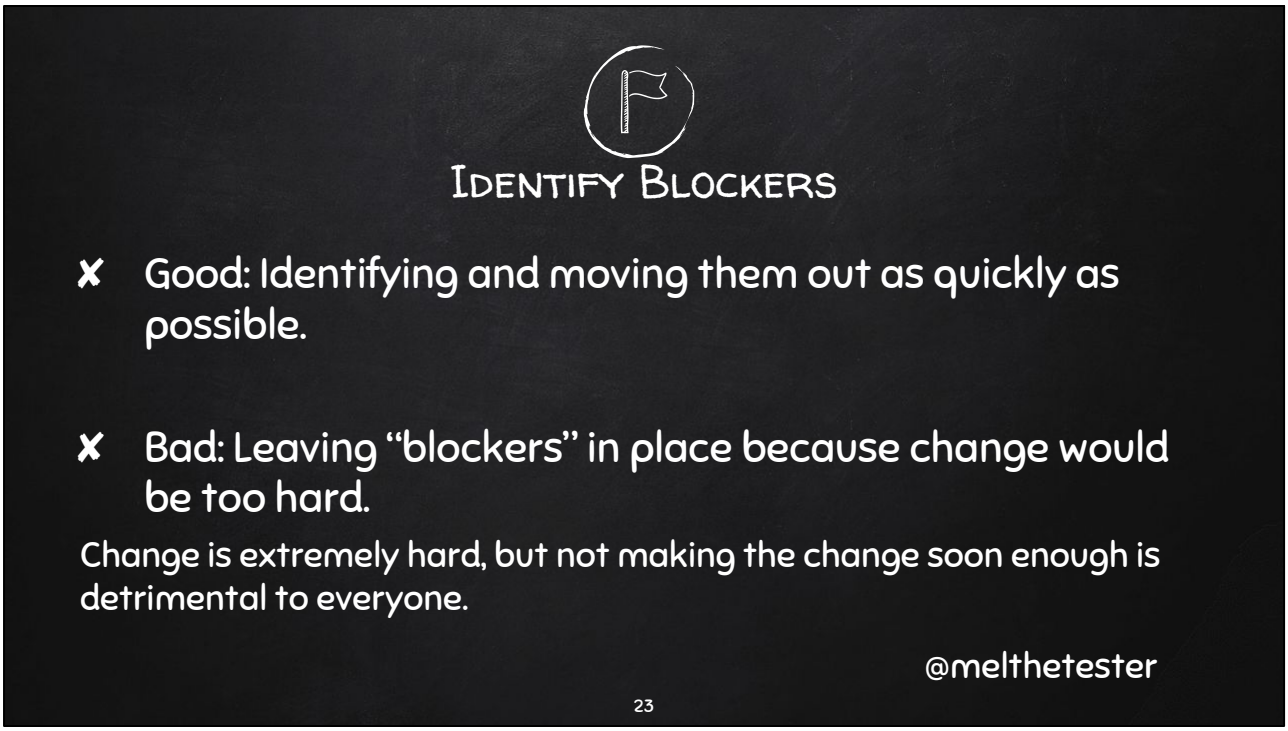
Talk to peers outside of your company and see how they handle it. Talk to others and ask them if they feel like they are having issues with communication and what they wished were better. Silos and lack of feedback can damage a team mentally and cause people to leave.



If you are aware of a situation - or you are the situation in this case, start by seeing if someone else can pick up parts of your job and learn them.

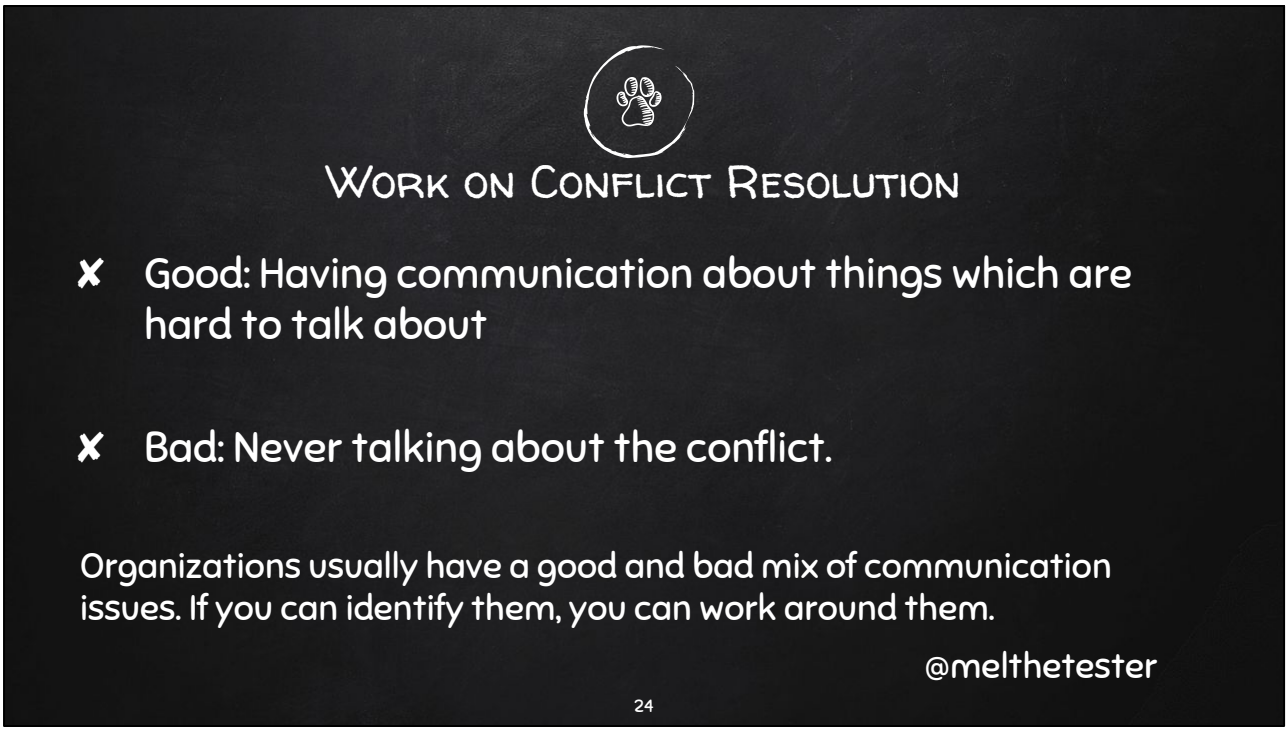
Don’t feel like the weight of the world is on you. You can train someone else to do what you do and they can learn.

Don’t be a martyr to the software, they don’t pay you enough(no matter how much you make).



Examples of blockers - environments not working, unavailable equipment, poor communication between team members.

If your workplace and teams are really proactive about helping you resolve blockers to you or the team, that's great. My experience has been that Testers always have to ask, and keep asking until they are unblocked. Keep asking for whatever it might be you need so you can get back to the problem of testing,



Testers are constantly putting themselves into a conflict of some kind. Whether it's a disagreement with something as small as where an element should be on a page, to something as large as a data migration project, learning how to deal with conflict and constructive feedback is an absolutely awesome skill to have. It's taken years to realize I needed to hone this skill, and even more time to learn how to do it. I've gotten better, without backing down from the position or the idea I'm trying to make.

If you are a tester, and you feel like you're causing problems, in a good way - in an improving the process kind of way - then you are doing your job. If you work with a tester that challenges you, or other testers that challenge you, this is a healthy environment and a healthy way to work. Testers are up for the challenge and they want to be challenged, or we wouldn't have taken the job in the first place.


QR Code
Here

THANKS!

Any questions?

You can find me at
@melthetester