INTERVIEW **PART II**

In the community of thinking testers, Mike Kelly is a familiar name. He is the creator of FCC CUTS VIDS heuristic, popularly known as Touring Heuristic.

Currently, Mike is looking after DeveloperTown as its Managing Partner among other interesting things he does.

Interviewing Mike has been on our wish-list and I'm glad that we could finally do it. Special thanks to Dirk for pairing up with me on this task.

Mike has provided insightful answers to our questions and for the benefit of our readers, we are publishing his interview in two parts rather than editing/shortening the answers to fit in one single issue.

I'm sure that you'll like what we discussed.

- Lalitkumar Bhamare

Over A Cup of Tea with Mike Kelly

Apart from software testing, what are the areas within software development that fascinate you? Why?

I'm currently fascinated by early customer development for a product. It's like this awesome mash-up of sales, marketing, analytics, and testing.

At Tenant Tracker, we built the first version of the product for one specific customer. That's a relatively straightforward process. You work with that one customer to figure out what they need, and you basically build it for them – with a slightly longer view then they might have for what the rest of the market might want.

Once that first client was live, the Tenant Tracker team started getting to work on selling clients two and three. Client's two and three are interesting when it comes to feature requests. You can't tell them what you told client one – "This solution will start off custom tailored to your business." For clients two and three, it's more like, "Tell us what features are critical to you. There's a good chance we have them already, but if not we will work with you to define and build them." But now you need to mash the needs of those new clients with the needs of the first client.

If the needs of those customers conflict, who wins? That's where marketing enters the picture. You have to try to guess what the nameless/faceless "market" wants. You know, all those potential future customers. Your marketing team (or you if you're a startup) now has to go research the heck out of existing market to try to figure out which early adoption customer wins. You have to make the call based on what you think the broader market will want. And maybe you need to design and run some early market validation exercises like: A/B testing marketing site conversions, testing pay-per-click campaigns, market surveys, prospect interviews, etc.

At Tenant Tracker, once we had clients two and three on board, we now had a new problem. We were no longer selling clients one at a time. Now we were trying to sell and onboard multiple clients at one time. It's at this point the early team loses the ability to start selling "customization" as a part of the deal. Your crazy small development team of two people can't juggle new feature development, client on-boarding, bug fix, production support, and sales commitments for the next eight prospects. So what do you do now?

Now you start to do "real" product ownership. You start to build a customer request backlog. When a client or prospect requests a feature, you say "Thank you so much. You're not the first to ask for that. We will add it to the backlog and let you know when we think we can address it." At each subsequent sprint planning, the sales guy argues for the features the current largest prospect wants, the lead developer argues for paying down the current largest piece of technical debt that's going to affect scale, the marketing guy argues for the best feature that will attract more prospects, the analytics guru shares what features customers are actually using, and the support guy argues for bug fixes and "small" client requests that should "only take a couple hours."

That's where we are on Tenant Tracker. Trying to juggle early customers, marketing data, sales data, and our vision for where the product should go. It's awesome! And this is what I think most testers on small teams get to do daily. It's all about triage, data-driven decision making, and communication. The hardest part is making sure you have a team that's committed to following through on the decisions that get made. (The Tenant Tracker team is above average at that I think.)

There are a ton of small strategies to juggling those tensions well. I feel like the exposure I've had to seeing many of our DeveloperTown clients wrestle with this have been invaluable. I have a front row seat to see what works and what doesn't. I can use that data to refine my models of what might work for early sales strategies for certain types of products. I have a list of the critical risks related to balancing customer expectations, technical debt, and revenue. And I keep an ever growing list of market validation techniques.

We learned that you also have a passion for farming. Do you see any similarities between your passion for farming and software quality?

About four years ago I started a hobby farm. We currently raise chickens, turkey, and bees. We have a fairly large garden. And we briefly experimented with quail. (My wife voted them off the island.)

Software testing is a never-ending process. There are an infinite number of tests you could run. As testers, we're paid to think through risk and coverage, so we can (hopefully) determine the most optimal set of tests to run in the time we're given. Farming is a lot like that. There's always more you can do on a farm: something to feed, something to fix, wood to chop, weeds to pull, etc.

You quickly recognize that if you want to have a life outside of farming, you need to assess risk and coverage. What's the important stuff, and how much of it do I need to do if I want a "good enough" farm? Do I care more about how my farm looks? Or do I care more about the health of the animals? Am I trying to make money? Or am I using this as an outlet to "recharge"? It's all tradeoffs.

In the last four years I've picked up hundreds of tools. I'm now an okay carpenter. I can administer first aid to poultry. I can make my own sausage (literally from scratch). I'm figuring out how to preserve food. I've picked up hundreds of little skills, but I'm not the master of any of them. I know just enough to get by. That's exactly like software testing. Testers know a little about a lot. We learn enough to get the job done, and then move on to the next item in the list.

I think farmers have it a bit easier in that the goals/priorities are (typically) clearer. The animals need to stay alive. Ideally we'd like them to be comfortable and happy while they are alive. And ideally, we'd like to make a profit at the end of the summer. If only the goals of a software tester were so simple. :)

In present day scenario, what are the testing skills you think testers must possess?

Before we look for testing skill, we look for ownership, someone who doesn't take themselves too seriously, an interest in learning new things, comfort with making critical decisions with little information, the ability to ask insightful questions, and good communication. That quickly gets us down to a smaller handful of candidates. From there, we start to look at specific skills and experiences.

To rapid-fire some quick critical skills that I think we look for:

- the ability to model a product quickly;
- the ability to extract requirements verbally;
- clarity in writing up and articulating issues;
- the ability to prioritize work;
- and the ability to assess risk.

Let's just look at those last two: prioritization and risk assessment. If you wanted to identify those skills in someone, you might ask them to look at ten defects in a backlog and see if they can quickly provide input regarding which ones likely matter and which the team might be able to safely ignore for some period of time. Their answer might not align perfectly with the product owner, but you get a feel for how they assess risk by what types of questions they ask during the assessment. And you can then see how they prioritize work based on that understanding of risk.

I think we favor testers who have some level of technical background (code, networking, etc.). We find that it makes them more insightful in terms of questioning, and allows them to come up to speed faster when faced with new technology. And I think we also favor testers who have some experience going deep on a product. While we rarely get the chance to go deep ourselves, the experience you can gain by spending a lot of time on one project is unique, and colors how you think about testing a product over time.

Are there any interesting stories from your coaching experience that you would like to share with us?

All of my coaching these days is around the core business. What's the problem you're trying to solve? Do you have the right team? Is the market big enough? Does the business model work on paper? How do we set prices? Do you plan on fundraising or bootstrapping? Do you know what it's actually going to cost to develop the product for the next three to five years? Where does testing fit into all of that? Sometimes it's tactical - "You need to add a tester to your financial model." Sometimes it's strategic – "What if we added a feature that allowed our users to perform quality control on the data without them even knowing it?" And many times it's educational - "When you say things like, 'It works on all Android devices…' you know what that means, right?"

Here's a current example. I'm working with a founder right now who wants to launch a startup with a two-sided marketplace. An example of a two-sided marketplace is eBay. For eBay to work, you need users willing to buy and users willing to sell. This is the scariest of all startups. You have two separate marketing and customer acquisition campaigns going on at one time. It's so hard to do because sellers don't want to enter a market when there aren't buyers already there. And buyers go to the site one time, don't see any sellers and then they never go back. In many startup circles, this is called the "chicken and the egg problem."

It's solvable, but normally it's solved with capital. You just raise enough money that you can afford an overwhelming campaign focused on one side of the market in an effort to get a chicken. Then you can worry about eggs. Sometimes it's solved with elegance. Where the startup launches as a single-sided market, and pivots into a two-sided marketplace down the road. It makes that pivot once it already has a bunch of eggs.

(Okay… I'll stop with that metaphor. Sorry.)

With the startup I'm actively coaching, we're exploring options. Things like:

- What are the value propositions that most resonate with your buyers?
- What are the value propositions that most resonate with your sellers?
- If you were to offer your buyers something else – not your original product – that provided a similar value, what could that look like?
- If you were to offer your sellers something else – not your original product – that provided a similar value, what could that look like?
- Would your buyers pay for that other thing? Even if there were no sellers yet? How much would they pay?
- Would your sellers pay for that other thing? Even if there were no buyers yet? How much would they pay?

Each of those questions might get us zero to ten ideas. Our job is to now test those ideas. We need to validate that our assumptions are correct, and we need to see if any of these ideas actually have traction in the market. If they do – then that idea might be the key to unlocking the two-sided marketplace problem.

Coaching becomes an exercise in educating the founder how to do those tests. What are all the ways we can reach out potential users to validate our assumptions (knowledge of testing techniques)? How do you identify your target market (coverage)? What do we ask them (test design)? How will we know when we're done (coverage, statistics, stopping heuristics)? When we get an answer, how should we interpret it (modeling)? What tools should we use? When should we run the test? How do we want to track results? Maybe we do some of that work, but most of the time we like to have the founder engaged in these activities. It's one thing to have someone else run a test for you and give you the results summary, and entirely another to run that test yourself and see/hear/feel the feedback directly. Are your potential customers passionate about the problem the way you are? Do they really want it to exist? Or is it just a nice to have? Is there something special about this prospect that could inform how I target future prospects? All of those are hard to pick out of a report. You need that direct experience.

The language we use is "market validation." But it's testing. A lot of the lean startup methodology is test-first, build second. My experience as a tester bleeds into a lot of my current coaching work – even though I'm not coaching testers.

From the recommendations you make on your blog, it's visible that you have read lots of books on multiple disciplines. What books/resources would you recommend for test professionals for their further learning?

There are a TON of places testers can go to find recommendations on testing books. I don't want to repeat that here. Instead, I'll offer some gems that I've found that I think can help testers raise their game in some specific areas:

- Want people to take you seriously when you say something? Read "Extreme Ownership" by Willink/Babin and "Pitch Anything" by Klaff. The Extreme Ownership chapters Believe and Prioritize and Execute will be particularly useful to testers.
- Want to build a team that truly delivers ridiculous results? Read "Creativity Inc." by Catmull/Wallace, "Drive" by Pink, and "Cleaning House" by Wyma. Which one of these is not like the others? Cleaning House is about raising your kids. It's dead on for crushing entitlement in your team as well. Read it through the filter of your work life.
- Want to change the conversation about how your organization should approach software testing? Read "Antifragile" by Taleb and "Good Strategy / Bad Strategy" by Rumelt. All of Teleb's books should be required reading for software testers, but Antifragile is the "capstone." Read it.
- Want to change the way you think about offshoring? Read "The Lexus and the Olive Tree" by Friedman and "The 4-Hour Workweek" by Ferriss. Each will challenge you in a subtle way to change the way you think about the future of how software will be developed – and what your role in that might look like.
- Struggling with depression, motivation, or a stressful work situation (like a startup)? Read "Resilience" by Greitens. This was my favorite book of 2015. It's exactly what I needed while struggling with some serious depression.
- Not into all that non-fiction and want a fun read that might teach you how to be a better tester? Pickup any Sherlock Holmes anthology and start readying. (The books are better than all the TV shows – and that's saying something.) I also recommend the "Gentleman Bastard" series by Lynch. Each book is themed a bit differently, but I certainly get an Arthur Conan Doyle vibe when I read them.

What is the next cool thing we can expect from Mike Kelly in coming future? (You may want to talk about some ongoing project or so…)

Well, probably nothing specific to software testing. At DeveloperTown, we're excited to continue growing as a consulting business, and we're launching a couple of new companies/initiatives in the next few months. If we are successful, over the next five years you'll see our brand grow nationally, and we will have more than just one of our own startups out in the market. While it's crazy what we've done in six years, we still see a ton of potential in front of us for what else we can do.

Personally, I'm excited to keep learning more about investing in early-stage startups, each year I get a deeper appreciation for what it takes to scale a business, and I'm constantly learning (mostly the hard way) better ways to lead. Maybe someday I'll get back to blogging about some of those experiences, but it's really difficult to find the time. Writing has always helped me firm up my ideas. Even this interview has helped me organize my thoughts around certain topics. So maybe this is a catalyst to get me writing again. That would be cool.

What would be your general advice for our readers who are mostly test professionals?

If you're not already, become a leader on your team. Leadership isn't about positional authority; it's about knowledge and experience, ownership for your work, caring for the people around you, and caring about outcomes. To be a leader, you don't have to be the best tester, the best coder, or the best anything else. You just need to be competent, drive things forward, and be the person the rest of the team can rely on.

When you think back to the teams you've worked with over the years, who stands out in your mind? I'm willing to bet that if it's more than five years ago, it's someone who was a leader on that team or in that organization. It wasn't the person with the most technical knowledge. It was likely the person who drove efforts forward. They likely didn't complain all that often. They didn't make excuses if they missed a deadline. They likely took a little risk now and then – spending some personal capital to try to create a better outcome for the team around them. They likely coached better performance out of you and others.

If that's not you on your current team, why not? In general, as an industry we need more leaders. In software testing, we definitely need more leaders. We're in a perfect support role for the rest of the team around us. Most testers touch every aspect of the product and get involved in almost all phases of a project. If you're great a testing (modeling, asking questions, critical thinking, etc.) then you likely have all the skills you need to be a great leader. Make that jump.

All of my opportunities came from stepping up and leading. Early on I wasn't asked to lead – I just did it. Up until DeveloperTown, my ability to influence was always based on my street cred – it was never based on my positional authority. Even here at DeveloperTown, I think a lot of my influence has to do with credibility that comes from doing – not talking. (I'm sure I'm wrong about that… my role probably has more direct influence than I'd like.)

Be someone who gets stuff done. Help others get stuff done. Make your team successful. If you do that, you'll always be in demand as a software tester.

Our last question for today, do you read Tea-time with Testers? We'd love to know your feedback.

I'm sad to say that I don't read it on a regular basis. I still follow a number of testers on social media, so if I see something talked about I'll read it. But most of my reading these days goes toward topics where I have much less experience.

That said, if any of your readers didn't read your Kaner's interview in your September 2015 issue, they should go back and find it. Kaner's insights into how the "state of testing" hasn't progressed are dead on.