The Future of Testing

Digging in the Past of Software Testing and Unearthing the Future

Kaspar van Dam

Abstract Many articles have been written on this topic. Many people have been trying to predict the future of software testing. Only time can tell who's right and who's wrong. Being a software tester today and a former archaeology student, I've decided to start digging in the past of Software Testing in an attempt to predict the future of this profession. What will change? What will stay the same? And what things from the past might resurface? Let's make a timeline starting around 20 years ago in 1998 (which is the moment when I myself started digging around in the world of computers and software) and try to stretch this all the way to 20 years in the future with 10-year intervals.

Keywords Software testing · Software quality · Software testing history · Software testing future

1 Introduction

Many articles have been written on this topic. Many people have been trying to predict the future of software testing. Only time can tell who's right and who's wrong. Being a software tester today and a former archaeology student, I've decided to start digging in the past of Software Testing in an attempt to predict the future of this profession. What will change? What will stay the same? And what things from the past might resurface? Let's make a timeline starting around 20 years ago in 1998 (which is the moment when I myself started digging around in the world of computers and software) and try to stretch this all the way to 20 years in the future with 10-year intervals.

K. van Dam
Improve Quality Services, Zwolle, Netherlands

S. Goericke (ed.), *The Future of Software Quality Assurance*,
https://doi.org/10.1007/978-3-030-29509-7_15

2 The Past

1998

Steve Jobs has returned at Apple just a year ago and presented the first iMac. Google is founded and the first MP3 is created. It's also the year in which MySQL and XML 1.0 are introduced. Windows 98 is released and Bill Gates gets a pie in the face. Seti@Home is started up and Internet Explorer becomes the most popular internet browser. The world gets to know blogs and it's also the start of ISEB Software Testing Certification (IEEE 829) and the third anniversary of TMap, which has changed the world of software testing quite a bit.

From the first day software was being developed, software was also being tested. However, around 1998 the first little steps were being taken in an effort to make software testing into a serious profession and people became aware quality assurance was something of importance in an IT project. But in 1998 a lot of software was still being released without any mentionable testing. At IT-related study programmes, software testing wasn't even part of the curriculum. But the introduction of TMap and other methodologies made an impact in the software development industry. People were trying to get a grip of what software testing was all about. They tried learning and discovering which steps to take to get software tested and get a grip on the quality of the software that was being developed. In the years that followed, software testing became an actual profession. However, there weren't yet any real requirements for the job. If someone could read and write he/she could become a software tester. If you failed at developing software you could always give software testing a shot. But most software testers back then didn't even have a background in IT. Strangely enough (looking back at it), this was actually considered something good. The consensus was that software testers should be strictly independent. Therefore they should not be hindered by any technical knowledge. They should focus on the functional requirements which were being written down in massive documents and by trying out newly developed software as an end-user, they should try and find out if these requirements were being met. In reality this often meant software testers considered it a sport to find as many bugs as possible. Some software testers might even remember the bumper stickers that were popular back in the days stating things like "Software Testing: You make it, we break it!" or "Every tester has the heart of a developer, in a jar on his desk".

2008

The world has changed. Computers and the World Wide Web are now commonplace. The first iPhone is barely a year old and Android and Chrome are being born. Facebook has been available to the general public for 2 years and no one can imagine a world without the Google search engine anymore. In 2008, both Spotify and GitHub are founded and it will be another year before the Bitcoin appears. TMap Next has been around for a few years now and ISTQB is celebrating its sixth anniversary. The Agile Manifesto is already 7 years old, but is now starting to get some feet on the ground. Also test automation is emerging more and more.

Most probably for most testers 2008 just went by like every other year. However, it has been quite an important period for the profession. Testing became more about involving people instead of endlessly scrutinizing the different documents and making it a sport to file as many bugs as possible in some bug tracking tool. From 2008 onwards, testers (and other people in the IT work field) slowly started embracing the agile methodology. After spending years and years making software testing into a serious, but especially an independent, profession all of a sudden, the profession started moving back to its origins. Because in the 1980s (and also before that) software testing wasn't considered a specialist job at all. A software developer wrote some software, tested it and went on with the development. As long as the system didn't crash and seemed to be doing what it was supposed to be doing it must be working as intended. With the introduction of Agile, SCRUM and later on things like DevOps, testing went back from being a strictly independent specialism to something that's part of the entire software development process. Looking back at it you may say the profession of software testing had lost its way for a decade or two and from around 2008 onwards it has found its purpose again. Software testing isn't about finding bugs. It isn't about being independent and having a developer's heart in a jar on the desk. It's about working together to create the best possible software. Software quality isn't about finding out what's wrong with the software (compared to what was written down in a functional design document). It's about the software meeting the wishes and demands of the person who will be using the software, the end-user. Instead of being this separate testing department spitting out bugs, software testers became part of the development teams. This also meant more and more software testers actually did get the heart of a developer . . . Until ca. 2008 it was said software testers tolerated boredom while developers automated it. When developers and testers started working together it turned out there wasn't any need to keep repeating the same manual testing tasks over and over. They could be automated! And many testers embraced this, they no longer needed to tolerate boredom! To many it must've seemed like test automation was born around 2008, in reality test automation is as old as software development itself. It was just forgotten (by most) for a period of time . . .

3 Present Day

We can't imagine a world without the internet anymore. Smart phones are part of everyday life and it's hard to even start comparing software development with what it was back in 1998. Hardly any IT project is done without at least some elements of the Agile way of working and things like DevOps and Continuous Development and Continuous Integration are now becoming commonplace. Words like communication and teamwork are (almost) considered being magic words in the world of software development nowadays. So, what have we learned from the past, implemented in our present and what can it tell about our software testing future?

Software testing has come a long way, but right now developments in this field of work are gaining momentum. Things are changing. Fast. And the people involved need to change along with it or be left behind. Most testing professionals we see today are nothing like the software testers from back in 1998. A question however could be: aren't we changing too fast? Aren't we forgetting the things we learned during the last 20 years (and before that)?

Today test automation may very well be the most sought after expertise when it comes to quality assurance in software development. Software development is all about continuously producing sufficient quality software. Time is everything, we want to release new functionality as soon as possible without being hindered by exhaustive manual testing. Therefore test automation is the only way to go. However, as some people may have thought for a while, tooling will not replace the software tester entirely. Simply because tooling isn't able to test software at all, it can just execute certain pre-programmed checks on the software. This means it's still up to a specialist to come up with the right test cases (and then decide if they should be automated or not). For software testers this means they should shift their attention from just software testing (e.g. TMap) to a much broader area. They should know about automation, tools and frameworks. But they should also keep a focus on their (testing) skills, including soft skills. Instead of following some testing technique out of a book, a software tester today should possess a certain mindset in which they understand people, product and process (P3). And besides just looking at functionality testers are often expected to keep an eye on non-functional requirements like performance, security and reliability as well. The shift from waterfall to agile software development has had a major impact on everyone involved with IT; however, it may have had the biggest effect on software testers. And I think this is just getting started . . .

4 The Future

2028

Let's just try and predict the future. At least, when it comes to software testing. Continuing with 10-year jumps we go to 2028. If we take a look at the origin of software testing and how it has developed the last few decades, than where do we expect the expertise to go next? From what mistakes have we learned? What things will we still be doing in 2028? What will probably change?

One thing is certain: We cannot predict the future accurately but we can be pretty sure that a software tester 10 years from now will still have to be flexible. During the last few decades software testers have changed from being a bit of rigid personalities who required to be independent on their own little testing island into possibly the most flexible people participating in any IT project. It's often the tester who makes the connection between businesspeople and IT people. It's often the software tester who starts working together with business analysts to change the way

requirements are written down (in order to get a better shared understanding of what needs to be built and tested by the development team). A software tester is required to understand what the business is talking about but should also be able to spar with a developer on how to implement some code and how to automate the tests that should check if the code is actually working. Therefore a software tester in 2028 might not even be 'just' a software tester. The role might be more about being the linking pin between business and IT and being the quality conscience of the entire BizDevOps team. This is something entirely different than back in the days. In 1998–2008 everyone with some analytical skills could become a software tester. If you failed at being a good software developer, you could still make a career move to software testing. But in the upcoming 10 years the role current software testers occupy might even become the most challenging role within software development. Being the quality conscience, you should always be adapting yourself to new realities. You should constantly be changing strategies and always keep track of things changing. Both from a business perspective and a technical perspective.

And even though this may sound like a lot of fun to many people, changing is actually the hardest thing to do for a human being. Simply put, our brains aren't built for coping with change. We were built to be organized, to find structure and patterns. To do things exactly like we always did them. To understand this we need to look a bit further back than 1998. More than 10,000 years ago we humans were still hunter-gatherers. The only thing we needed to worry about was surviving and the best way to do that was to follow strict patterns. We often went to the same places to hunt or gather food and we could trust our brain would warn us if anything was out of the ordinary. Because that's how the human brain has developed. It's always more or less in a relax mode, until something changes, when a chemical called cortisol is released, also known as the stress hormone. It induces a state known as the fight-or-flight reflex. Adrenaline starts pumping, our field of vision narrows, muscles contract. We get hyper-focussed on the one thing that has changed. Was it a threat? A predator? An enemy? Some other form of danger?

Nowadays we're working in our twenty-first century offices, but our brains are still the same as they were 10,000 years ago. They haven't yet adapted to this new environment. Evolution doesn't like to be rushed. So, even though we might think we embrace change. Even though we think constant change makes our day-to-day work more fun. Our brains don't like it at all! Knowing this, it's no wonder people on the work floor are clinging to old patterns. There is a reason we often still prefer to have a structured Ways of Working, preferably documented in some handy Excel checklist ... There's a reason we may feel lost when things all of a sudden change when we're in the middle of something. The Agile way of working is all about adapting to change. But we should be more aware that this is actually something we humans aren't really good at! (And if any reader is now shaking his/her head mumbling 'no, no, no, that doesn't apply to me!', trust me: it applies to you as well. It's simple biology! [1, 2]).

So, will we be abandoning the Agile way of working in 2028? I don't think so. However, I do suspect we will change it quite a bit. When looking at software development teams I see a lot of developers struggling with all these Agile meetings,

things changing all the time. When they just started putting a shoulder to the wheel with some technical challenge there's someone at their table inviting them to some refinement or 3-Amigo session because some new insights have popped up and everything needs to be different. Once again! The problem is, this process of continuous changing will remain in the future. I think it will even get worse. Technology had many limitations in the past, but they're vanishing really quickly. In the past you could ask an end-user to wait a year or two before expecting some new functionality. But in 2028 (or really even today already!) when an end-user wants something of the software, he/she expects it to be there tomorrow. Or possibly even sooner. And if not: there's always someone else who can make it happen that fast. This means in 2028 we IT people need to be even more flexible and more adaptive to change than we are today. And someone needs to make sure the people writing the actual code can keep their focus on the code. To make sure the developer knows where to pick the berries and where to hunt a rabbit without constantly having to look over his/her shoulder for some predator approaching, figuratively speaking. And that someone might very well be the person we call a software tester today.

This means software testers have a lot to do in the upcoming 10 years. Because as Darwin stated: "It is not the strongest of the species that survives, nor the most intelligent; it is the one most adaptable to change." If you still want to be part of the game, if you want to survive in the world of software testing, a software tester should keep learning, training and adapting to change. Even though our very own brain is resisting this. The difference between a 1998 software tester and a 2028 software tester may be comparable to the difference between a 'common' soldier and a commando. A soldier is trained to do as he's being told, a commando however is trained to stay alive and accomplish his mission by constantly adapting to change. This doesn't mean one of these two is better than the other, but it does require a different type of person, a different training and especially a different mindset to be a commando instead of a 'common' soldier. The software tester of 2028 might very well be the commando, while the software developer may remain a soldier.

In this (brave?) new world of software development in 2028 I don't think we'll still have the job title 'Software Tester'. The role will be about so much more than just testing the software. It might even be probable that the actual testing of software isn't the main focus anymore in 2028. Nowadays a lot of manual testing is already being replaced with automated testing. However, as stated before, test automation today is just about executing pre-programmed checks on the software. I'm not sure if we're there already in 2028, but I do believe artificial intelligence will influence test automation a lot in the future. Which means that in time a computer might actually be able to at least do a small amount of actual testing of software instead of just checking some predetermined things. This would mean that the actual testing of software would become even less important within the job description. So what shall we call this 'software tester' in 2028? Maybe 'Quality engineer'? Or 'Change specialist'? Maybe even 'Dev Commando'? I personally wouldn't vote for any of these new job titles. I think in time some new term will pop up describing this new

role. After all, back in 1998 who would have thought we would be having business cards today with terms like 'Scrum-master' or 'Product-owner' . . .

2038

It may be near impossible to predict the future 20 years ahead. But let's give it a try. In 2038 we're probably driving autonomous cars, and robots may very well be part of everyday life. Artificial Intelligence will be a lot more intelligent than it is today and may even be replacing knowledge workers. Will there still be software development as we know it? And will there be software testing?

I personally believe we will still be needing people to develop software even though it's probable we'll be able to produce a lot more software with a lot less people. It is possible software has become intelligent enough to be testing itself. Software may be continuously running self-diagnostics which indicate when something's going wrong and the software in question might even be able to fix itself, at least to a certain degree.

But I don't think computers and robots will have replaced everyone involved with the development of software. Simply because of one thing: Artificial Intelligence is only intelligent about certain things, but really unintelligent when it comes to some other things . . . This is clearly visible today, but I don't expect it will be all that different in 20 years from now. Take a look at current-day tools like Google Assistant or Siri. These AI's know a lot more than any human being knows (because they have an endless supply of information constantly at their disposal). Some robots today are amazing at interpreting their surroundings and figuring out what's expected of them. Current prototypes of autonomous driving cars may very well already be safer than human drivers. However, even with all this computer power and all this data and intelligence there's still one thing at which every AI sucks: understanding human behaviour.

A great example is Honda's humanoid robot Asimo, which was developed a few years ago. In every single way this was a great feat of engineering. However, during a demonstration it failed horribly because of one simple misunderstanding of human behaviour. The robot didn't understand why people would want to take pictures of it and thus concluded that when people were raising their camera or mobile phone to take a picture, they were raising their hands to ask a question. The robot froze, repeating over and over "Who wants to ask Asimo a question?" [3]. And this interpretation of human behaviour is something current AI still doesn't know how to do, even though it's something we humans find very easy! We immediately understand what's happening when someone is hanging out of the window of a train that's about to leave to hug someone outside. They're saying goodbye. Pretty straightforward, right? However, an AI might mistake it for someone trying to pull another person out of the train. There might be something wrong in the train, an evacuation might even be necessary . . . . I don't believe AI will be much better at interpreting human behaviour in just 20 years from now. And therefore it's very likely we'll still be needing humans to develop software that will actually do what an end-user is expecting from it. And when we're still building software we'll also still need someone to act as the earlier mentioned quality conscience. Someone who's

able to translate what an end-user wants from the software and who's capable of telling if the software is actually serving its purpose. And even though computers might take over a lot of work in the software development industry, I think there will still be a lot of work to be done by human beings, especially when it comes to quality assurance and software testing. However, it will most probably be a completely different job compared to the job today.

5 Conclusion: This Is the Future

To conclude things. During the last few decades the world of a Software Tester has changed dramatically. Looking at the history of software testing and developments in the work field today, I don't expect the upcoming 20 years will be any different. Just like many software testers today can't even imagine what they were thinking back in 1998, I believe that in 2038 people will have a hard time imagining what software testers are doing today. Things like Artificial Intelligence will change the software development industry completely. And it is expected AI will, in time, be a part of about every piece of software. We will find ways to have software adapting itself to the world around it. The software can change according to the needs of people using it and it might even become able to test itself and fix possible bugs it detects in its own lines of code. However, we will still be needing people to act as a quality conscience. People who can make sure software being developed is meeting the requirements of human beings using the software. Human beings whose behaviour will most probably remain a mystery to even the most intelligent AI. It might even be the software testers, or whatever we will call them in the future, who could one day prevent AI from becoming too intelligent (Skynet, anyone?). But what does all this mean for current day software testers? What should we be focussing on? For what things should we be preparing ourselves?

First off, we should keep investing in test automation. It can only be expected test automation is something that will stay and it will keep improving itself until eventually test automation frameworks will actually be able to do (some) testing instead of just checking some pre-programmed test. But, more importantly I believe we should keep investing in communication and collaboration. It's incredibly difficult to get a good understanding of what an actual end-user of a piece of software wants and needs. And it might be just as difficult to make sure everyone involved with creating that software shares the same understanding of what it is the software should be doing to meet the requirements from this end-user. Even today software testers should already act as a quality conscience and make sure software is meeting the requirements. This means the tester is already a linking pin between business and IT today, and will be this linking pin even more in the (near) future. Software testers today might consider broadening their expertise and start looking more at non-functional requirements like performance and security. These things are already really important today but will be even more important in the (near) future. We don't want our current-day laptop to be hacked, nor do we want it to fail at

certain tasks because it's incapable of executing some calculations within a required time frame. However, when something similar would happen to an autonomous car driving at a great speed on the motor way in 20 years from now it would predict disaster! It's up to the software testers of the future to make sure chances are as small as humanly possible that something like that can happen with software that might be responsible for human lives. I don't know what the future of software testing will be bringing us, but I do know it will require us to keep staying ahead of the game. To keep investing in knowledge, to keep improving in our skillset and to remain able to adapt to change. Whatever change that may be. Only then will we be able to survive in this future world of software testing!

P.S. Anyone up for a cup of coffee somewhere in 2038 and a look back with me at the things I've written down here?

Acknowledgments The author would like to thank Joris Meerts, Pascal Maus, Pieter Withaar, Piet de Roo, Huib Schoots and Berry Kersten for their input and feedback on this chapter, which was loosely based on an earlier published interview with some of these people.

References

1. Gray, K.: What Goes on in Brains: How an Understanding of Neuroscience Makes a Difference When you Advocate Agility; P3X Conference November 8th 2018, London (2018)
2. Levitin, D.J., Luke, D.: The Organized Mind. Penguin Group US, New York (2014)
3. https://www.theverge.com/2013/7/6/4498808/honda-asimo-disappoints-when-it-confuses-phones-for-hands. Accessed June 19 2019