Toggle navigation

- Events
- Club
- Dojo

- All The Things
  -
    - Ministry of Testing News
    - Community News (Testing Feeds)
    - Latest Testing Jobs

* Search

Search the Dojo Catalog

Search

- Newsletter
- Handbook
- Press Kit
- Advertise

Continuous Call For Papers!
Our Awesome Merch Store
MoT Slack Invite

- Join Now

- Support
- Sign In

Login email or username
Password password
☐ keep me signed in
Sign In
Forgot password?

- View Series: The Testing Planet 2017

30 Things Every New Software Tester Should Learn

by Heather Reid

This is a guide to learning more about software testing. As you start on your journey you will have tasks you can work through. Software testers are always learning but we cannot always quantify it. We learn about the product we are testing. We learn and develop relationships with developers, managers and testers. This makes us great at what we do. We are also chameleons. We change based on the environment we are in or the product we have to test. We have to continue to educate ourselves about the tools we need to test each product.

I have split the tasks up to create more digestible chunks of information. This is by no means a complete list. It will get you started on your journey. Take each task and work towards checking off the activities.

After completing the activities listed here, there is further exploration to do. As you follow the steps, reflect on what you have done so far. Maybe you would like to redo a task or adventure down another course of research. Tasks could even be done in a different order. Every tester is different and that is an asset to your testing ability.

✓ (Task 1) Are you interested in software testing?

Let's start with a little self-reflection.

An important part of being a tester is your mindset and being aware of it.

- What led you to this article?
- Why is software testing valuable to you?
- What do you believe software testing is? We'll talk about this

more in the next Task but it's good to think about what you currently believe it is.

Take a moment to think about the questions above and other questions you might have. By the end of this guide the aim is to have the majority of those answered. You're going to have more questions as you go too. Questions prompt a search for answers which results in learning.

We are surrounded by tech daily. There may be things you use in your everyday life that bother you. Perhaps a software package at work that takes longer than a minute to load. The app on your phone that freezes just after an update. Or perhaps when you get locked out of your emails and discover you can't recover or reset your password. You might have started to think more about that software package by maybe looking at your task manager as it loads.

Can you be a person who strives to make software better? Can you be an advocate for the users? The business? Your work colleagues?

Do you know what it takes to get there? Do you know what you need to learn? Think about all the good and horribly bad software you have come across in your life. You are the chameleon in both. In the good situation you want the product to stay good so you keep learning to stay on top of it. In the bad situation you want to learn more so you can feel like you did everything you could before the software was released. Learning is the key. No matter the situation you will always be learning even if you aren't aware of it.

This sounds daunting, don't let it put you off. Start with the first question and think about it. By the end of this guide you will be more confident in your answers to all of these questions.

What to do next:

- Commit to print why you are here (a post-it or whatever medium you prefer)
- Where do you plan to be at the end of this?
- What skills do you have now that you think are useful for testing?
- What do you currently perceive software testing to be?
- Plan this guide into your schedule. Be realistic about what you can achieve. Consider your day to day work and your energy

levels.

✓ (Task 2) What is software testing?

Start by writing down what you think software testing is and what a software tester would do.

There are many aspects to software testing. It does not always involve using the product. It is not just about finding bugs. Testing can start around the requirements stage. Thinking about what the product should do, where risks could be, and how the user/customer navigates the product is all part of testing.

One discussion I recently had, about whether it is fair to assume that customers are your users, gave me perspective on terms we use to represent stakeholders. Stakeholders, to me, covers everyone who has an interest in the software product. In the end we agreed that the person who pays for your software may not be the one who actually uses it. This is a great example of how software testers should look at things from different perspectives. Below are examples gathered from discussions with other testers about what they think testing is or is not.

What Testing Is	What Testing Is Not
Verifying whether software meets the expected value to its users. Investigating the product to find any information that is of value to our stakeholders.	Simple, quick, predictable. Only verifying that the product matches the description.
Mitigating surprises - Trying to find problems before software is released to stakeholders.	Increasing quality - this is a whole team exercise. Prove that the software has worked once.

What Testing Is	What Testing Is Not
Exploring products. A valuable activity in software development but often misunderstood due to its unpredictable and creative nature.	Well understood or valued by everyone. A process that is fixed, unimaginative and best kept under strict rules.
Communicating - with stakeholders (customers, users, developers, etc.) and work together to find out if the product is improving.	A phase that a project needs to go through in order to be successful.
Infinite. All testing is sampling. For every non trivial product, there are an unimaginable number of parameters with a great number of possible values. How do you know you are testing the important ones?	Ever finished - we can make decisions about stopping criteria but there are an infinite amount of combinations that could be checked.

What to do next:

- Read this list of things of what software testers do.
- Try asking people around you about the title of this task. Ask people you work with testers and non-testers, ask people on Slack and maybe even ask your family. Has this changed much from what you originally thought? How different were the responses you got?
- Research other industries such as pharmaceuticals, appliances etc. What is their approach to testing? How different or similar is it to the tech industry? Do any of the differences change your thoughts on how you would approach testing?

✓ (Task 3) Testing for non-testers

When I started in software testing I had no idea what testing was. I also had no clue of where to start. One of the most useful resources I came across was the pathway Testing for non testers by Katrina Clokie. Using Katrina's pathway I was able to understand testing and the value it provided. I was also able to investigate her references further to start expanding my knowledge and list of people I should look to for advice.

This is one of the things that will not be done overnight! I recommend reading it and referring to Task 1 of this guide each time. Reflect on each link you read within Katrina's blog post. Are there any things you want to practice? Are there any things you don't and if so, why? Has this changed the thoughts you wrote down on Task 2?

What to do next:

- Share the link with your team
- Read an article a day from Testing for non-testers
- Create your own list of things to learn and practice. You will most likely be updating this constantly

✓ (Task 4) Get social

This will be a slightly easier task to get started with but you will need to put in effort to keep it up. Start as simply as you like or feel comfortable with. I do recommend achieving all of the steps below when you can.

Engaging in social activities has enabled me to meet local testers who are willing to help me when I come across a tough problem. Through these social gatherings, I have made some friends in a city I am new to. It's nice to have a sympathetic ear with friends who understand what you are talking about.

What to do next:

- Go to Meetup.com and look for a local software testing meetup
- Join the testers.io or Ministry of Testing Slack
- Join Twitter, start finding testers to follow and get involved in

some conversations
- Join The Software Testing Club
- Read Andrew Morton's blog posts on attending the Bristol STC meetup and the Cardiff STC meetup
- Start looking up testing conferences such as TestBash, Agile Testing Days, EuroStar, STPCon, Global Testing Retreat, TestCon, STAREAST, Let's Test, Nordic Testing Days. We'll cover these more in Task 23. They are a big part of getting social.

✓ (Task 5) Get organised

It can be difficult to stay organised when there is so much information being thrown at you. I struggled with this a lot, I still do occasionally. Tiny Habits was recommended to me by another tester when I was getting started. I go back to it frequently and set myself new tiny habits to achieve, particularly when I feel like things are going awry. This does not take a lot of time but it will save you in the long run.

You might also want to think about writing mind maps. There are a lot of free tools out there to help you with this. I find a mind map can help me streamline a random thought process a lot easier. Ever had a million ideas running through your mind that are loosely related but you can't get them into a sensible order to make something of? Mind maps can help with this, so can the good ole reliable post-its on a wall approach.

If you're more of a gamer, or the above don't appeal to you you could also try Habitica. It's a free app where you can earn points/coins when you achieve your daily goals. You also lose health if you miss a daily goal. Kinda cool!

What to do next:

- Sign up to Tiny Habits and do it for a week
- Start a mindmap
- Investigate if Habitica is for you
- Create a personal Kanbanflow or Trello board. Start with the column headers "To Do", "In Progress" and "Done". Add a card for each thing you would like to achieve and move them across the columns as you progress with them.

✓ (Task 6) Sign up to the Dojo

Sign up to The Dojo and explore it. The Dojo is run by the Ministry of Testing. I have found it to be an amazing resource for my career. I can watch training videos, access testers blogs and I get notified of upcoming events that they host. I started with a free membership and finally graduated to a paid membership this year.

What to do next:

- Start with a free membership to the Dojo
- Read these web testing 101 articles on the Dojo
- Watch this video from Michael Wansley at TestBash Brighton 2016
- Watch this video from Danny Dainton a nice 99 second talk about how he stopped wasting his time
- Investigate the content on the Dojo further

✓ (Task 7) What is a bug?

I know a bug in software when I see one, or at least I like to think I do. I thought it would be easy to explain what a bug is. When I tried to find a definition I found many that just didn't cover the scope enough. James Bach defines a bug as "Anything that threatens the value of the product. Something that bugs someone whose opinion matters". That is a nice high level definition. On Testing Computer Software you will find "most bugs cause a program to change its behavior when the programmer didn't want or expect it to, or cause the program not to change its behavior when the programmer did expect it to".

In most of the definitions I found them to only mention the code or the developer. None of these are wrong. Going back to the definition from James Bach and looking at the ideas of the three amigos we could also have bugs in specifications. This would threaten the value of the product and it would not then be a bug with the code.

Think about the technology you use:

- Have you encountered bugs in any of it?
- Why did you feel like they were bugs?
- Did you do anything about it (e.g. report a bug in an app

through the app store)?
- Can you think of three different things that help you decide whether a bug is actually a bug or not? These are called Oracles and we will look at them in more detail in Task 13.

✓ (Task 8) Recommended Reading

This is a starting point for your reading list. You won't achieve all of this in one day. Some of the books I list here will be referred to in relevant sections later in the series. This is to make you aware of the existence of the books and work towards purchasing or borrowing them from the friends you will make when you get social.

I asked people what books they would recommend for testing and I got the following (this is by no means a comprehensive list):

- Explore It! by Elizabeth Hendrickson, an excellent book about exploratory testing.
- Evil by Design by Chris Nodder, I love this book! I learn something new every time I read it. I'll cover more about this later.
- Agile Testing by Lisa Crispin and Janet Gregory is one of those books that everyone raves about and you wonder why it took you so long to read it.
- Thinking, Fast and Slow by Daniel Kahneman which I have not personally read. This has been recommended to me by enough people that I will be starting it as soon as I finish this article.

The Dojo also has a list of free and paid for eBooks that gets added to over time. Check that out now that you're a member.

Next:

- Start to read one of the books mentioned above. If you're unsure where to start, I suggest Thinking, Fast and Slow.
- Look at the recommended reading list from the CAST 2015 speakers.
- Ask on one of the Slack channels or on Twitter for more suggestions of books to read.

✓ (Task 9) Write a user story

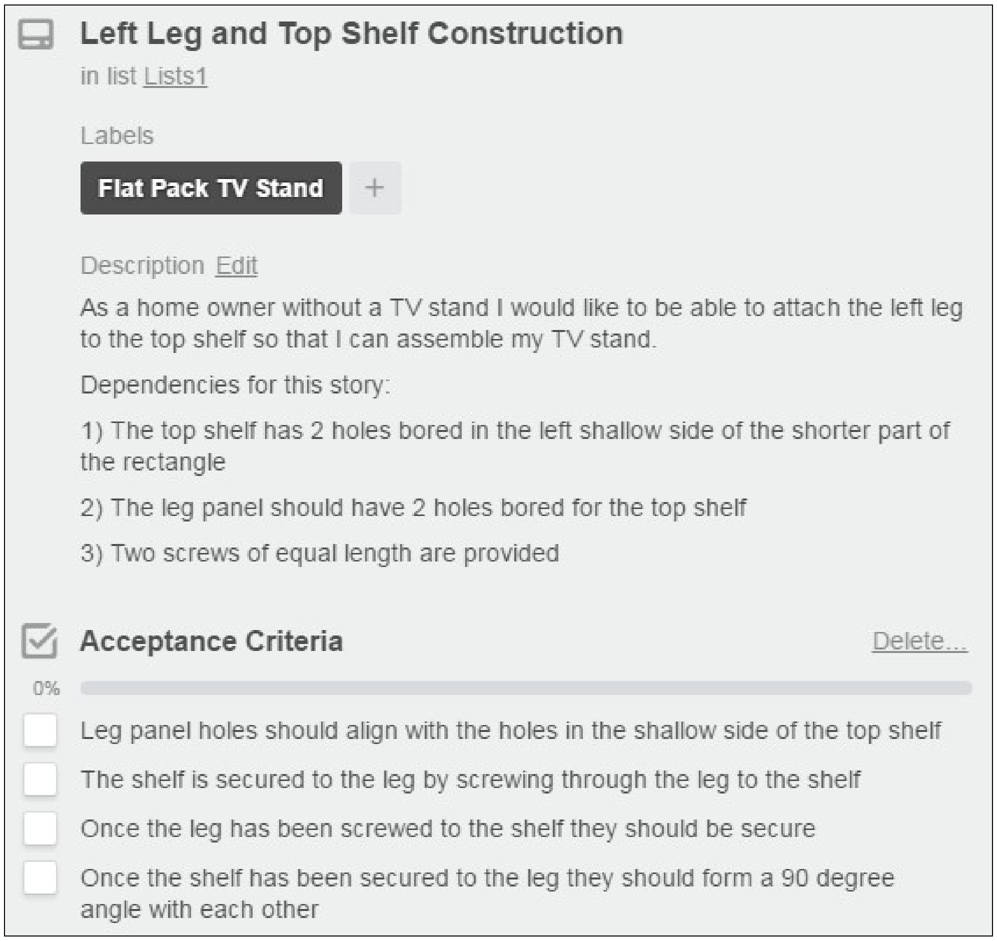
A user story is like a set of instructions you get with flat pack furniture. By writing a user story it will help you to understand their purpose.

It should provide:

- A description of what needs to be done.
- Acceptance criteria outlining what the story needs to accomplish.
- Any criteria which are out-of-scope for the story to be considered accomplished.
- Note any dependencies required to start the story.

Mountain Goat Software has examples of agile user stories to give you an idea of what to expect when you're out in the wild. EPICs are essentially a large user story. They are broken down into smaller digestible or deliverable pieces called user stories.

Take the user story below as an example, the overall EPIC would be "Flat Pack TV Stand". The user story is then a smaller piece of that "Left Leg and Top Shelf Construction". The dependencies are things that we need but not things we would do, for example we would need a screwdriver to be able to assemble the shelf. The acceptance criteria are things that we actually have to do.



Left Leg and Top Shelf Construction

in list Lists1

Labels

Description Edit

As a home owner without a TV stand I would like to be able to attach the left leg to the top shelf so that I can assemble my TV stand.

Dependencies for this story:

1) The top shelf has 2 holes bored in the left shallow side of the shorter part of the rectangle

2) The leg panel should have 2 holes bored for the top shelf

3) Two screws of equal length are provided

Acceptance Criteria Delete...

0%

- [ ] Leg panel holes should align with the holes in the shallow side of the top shelf
- [ ] The shelf is secured to the leg by screwing through the leg to the shelf
- [ ] Once the leg has been screwed to the shelf they should be secure
- [ ] Once the shelf has been secured to the leg they should form a 90 degree angle with each other

Writing a user story will help you to think about writing requirements for users more. It will also help you to think about the approach to take for testing these user stories. As a tester you will have a unique set of skills when it comes to analysing user stories. You will be able to highlight missed details, misuse cases, abuse cases and how it can be misunderstood. Not everyone gets the chance to provide early input with user story writing. It is an excellent process, you are testing before there is ever a piece of code written!

Your tasks for today:

- Write a user story for making toast, a peanut butter sandwich or even Ikea furniture using the templates above.
- How do you know if you missed something?
- Have you made any assumptions?
- Think about the order things need to be done in. Are there any optional inputs?
- Are there any requirements or prerequisites (assume the user has bread for example)?
- Try to make toast or a sandwich from scratch following your

user story exactly.

✓ (Task 10) Testing personas

When you are testing you shouldn't only test for you. You need to think about impatient users, users who take their time, users who have a bit of a tester in them and like to "break" things and many more. Katrina Clokie has an excellent guide to getting started with testing personas. As you get familiar with an application, things like this may slip from your mind and you may go into autopilot somewhat. I find trying to get into the user persona more helps me to avoid this pitfall.

Before thinking that personas may not already exist within a project, it may also be worth looking into whether personas research and documentation already exists. You can then use it to build things from a testing perspective.

For today:

- Pick an app on your phone or a program on your computer to test
- Navigate through it using the different user personas in Katrina's article
- What did you learn from this?
- How did the application perform under each persona?
- Think about and research how others approach personas

✓ (Task 11) Test a user story

This is linked to Task 9 and a follow on from Task 10. There are many ways to document testing but we will not focus on those here. If you've never written testing documentation, you could look at the test case templates on Tutorialspoint.

Task 9 and 11 have many similarities: While writing the user story, you were testing it just as well, weren't you? But the thing that has changed is that you now have an actual product at your disposal and that makes a world of difference.

You now have something to get hands on with, you can leave the theory behind and put it to practice. This is where you start using your ability to learn. The speed and precision with which you can learn new and interesting information will define your skill and value as a tester.

Concentrate on your mindset as you test the story. This is the most important part when testing a user story.

Questions to ask yourself as you test this story:

- Who is this story important to?
- What is the one problem this User Story tries to solve?
- How could this be misunderstood?
- How could accidental bugs find their way into this?
- What behaviour should it absolutely NOT show?
- When are you done?
- At what point did you feel like quitting your analysis and want to start testing?
- Does this feel reasonable?

✓ (Task 12) Heuristics

Cem Kaner and James Bach describe a heuristic as "a fallible method of solving a problem or making a decision". Karen Johnson says that a heuristic "Refers to experience-based techniques for problem solving, learning, and discovery. Where an exhaustive search is impractical, heuristic methods are used to speed up the process of finding a satisfactory solution. Examples of this method include using a rule of thumb, an educated guess, an intuitive judgment, or common sense."

James Bach has an excellent document about how to design a test strategy using heuristics. Read over it and think about it for a while.

It's a lot to take in when you first read it. If you think about how to apply and understand heuristics in testing, understanding these concepts get easier with time and experience.

Michael Bolton has an interesting blog post about heuristics to help you know when to stop testing. This is important in testing as we can become so absorbed in testing one feature that we may miss others.

- Can you think of any other heuristic? Explain it to someone and give it a name. Test Sphere and Explore It! from Task 8 could help you with this.
- Do you know of any heuristics you have used before? Use the articles above as guidelines.
- Research and share other heuristics you might have found.
- Pick any of the heuristics and start a discussion with peers.

✓ (Task 13) Oracles

According to Michael Bolton "An oracle is a principle or mechanism by which we can tell if the software is working according to someone's criteria; an oracle provides a right answer--according to somebody". Cem Kaner describes it as "a software testing oracle is a tool that helps you decide whether the program passed your test".

Michael Bolton also talks about how all oracles are heuristic but not all heuristics are oracles. This can take time to understand so do take your time with it. Katrina Clokie's article about Heuristics and Oracles may help you to think about this in everyday terms.

What to do next:

- Read Testing Without A Map by Michael Bolton. It gives a step by step guide on using oracles to determine if you are 'done' testing.
- Check out Testing Mnemonics list that Lynn McKee has put together. Also read FEW HICCUPS by Michael Bolton which expands the HICCUPS mnemonic.
- What oracles have you used? Can you see any oracles in your testing work?
- When an app on your phone updated, did it maintain a consistent behavior? If not, did you like this inconsistency?

Was it an improvement on previous behavior or did you consider it to be a bug?

✓ (Task 14) White & Black Box Testing

White and Black box testing are two approaches to software testing. Think of them as two umbrellas for types of testing to fall under.

White or Glass box testing is a testing method where the tester knows the internal workings of the program.

The tester writes a strategy to test the program based on knowledge of the application's internal code. They understand what the function is doing and what inputs will produce what outputs. This type of testing typically focuses on coverage of statements, lines of code, branches of code, etc. White box testing most often applies to unit tests.

One of the biggest risks with this type of testing is the inability to find missing features. You cannot cover a line of code with a test if the line of code never existed. The tester is required to have knowledge of the language that the program was written in. With this type of testing you are also naturally biased with your thinking patterns given the time you have spent within the product. As a result, you might miss obvious or non-obvious problems due to being too close too it. Think about signing in to your emails. It's something you do every day and you follow the same pattern every day. When was the last time you noticed a change on the login page or process? You become so used to what you expect to see that you may not notice the unexpected.

Black box testing is a method where the internal workings of the program are unknown to the tester. Tests are based on the requirements and functionality. One of the advantages of this over white box testing is the possibility of identifying missing code statements. This approach does not require the tester to have any knowledge of the language that the program was written in. A risk with this type of testing is the unknown level of coverage that testing gives.

I had much fun and frustration at a workshop by Alexandra Casapu trying to solve puzzles like these Black Box Puzzles. These are a good way to hone your skills as a black box tester. Try some of them. Did you manage to figure any of them out? If you did, can you figure them out a second time? Can you remember your approach? Did you change your approach the second time?

What to do next:

- Try out doing some of the Black Box Puzzles
- Try testing any mobile app you have on your phone, you would essentially be doing black box testing
- Go to the Khan Academy Computer Programming section, try out some of their tasks and see how you can create or identify problems with knowledge of code

✓ (Task 15) Accessibility Testing

Do you know anyone who is colourblind or maybe uses a screen reader? I know people who are colourblind, visually impaired, deaf, dyslexic or have photosensitive epilepsy. As a result I am always thinking about these users along with the 'standard' user set for our application. Luckily there are tools that can test an application for its accessibility level. A fun and easy trick to try is navigate through an application you are testing with no mouse or trackpad. Sounds easy right? Give it a go and see what you think. There are a few more quick things to try in this 6 Simplest Web Accessibility Tests Anyone Can Do. There are various plugins you can add to your browser to view your application the way someone with colourblindness would.

What to do next:

- Pick an application and try some simple accessibility testing on it. Did it perform as you expected? Were there any issues you encountered?
- Read A Software Tester's Guide to Web Accessibility.
- An Alphabet of Accessibility Issues is a great article to give you insight into the variety of needs people have.

✓ (Task 16) Podcasts

Ever wanted to learn about new things while you work, drive or take the train? I always want to learn more in work, but have so many things to do. I can't always sit down to watch a video of a talk or read a book or paper. Podcasts have given me the opportunity to learn while I work or commute.

I met Vernon Richards at TestBash and started to follow him on Twitter. He shared a link of a podcast he had contributed to which was related to testing. I had some time to spare so I investigated it. From there I started to explore all of the other testing podcasts that are out there. Now I try to make a point of listening to at least one per week.

There are two sets of podcasts I love to listen to. Joe Colantonio Test Talks is the first one. Go there and start picking podcasts to listen to.

Testing in the Pub is another one of my favourites. The first one I ever listened to was An Intro to Cynefin and I've been slowly researching the topic ever since. Some of the topics in both places can be a bit heavy. Find the ones you are interested in the most and start with those.

Podcasts are an excellent resource. Try to get into the habit of listening to them. I'm sure you'll soon have your own favourite ones.

What to do next:

- Check out the Ministry of Testing Podcast Testing Feeds
- Explore the content on Let's Talk About Tests
- Listen to a podcast on the Dojo podcast series

✓ (Task 17) User Experience (UX) as a tester

The user experience is pretty much what it says on the tin. It is how the user experiences your application and how they respond when they use the application. Don Norman describes it as even more than that . Testers are often tasked with understanding user experiences and helping ensure the user has a good experience by reporting back any possible issues to the rest of the team. Even before the product is out on the market, testing can help to highlight UX issues. An example would be a page takes a long time to load. A user wouldn't wait that long because the competitor product hasn't got the same issue. We should investigate the delay and try to fix it.

The Nielsen Norman Group have an article about heuristics for user interface design which can help with the user experience.

At the end of the day it is users who will give you feedback on their experience with the product. As a tester you can help to ensure that their initial experience is a good one.

What to do next:

- Start to really think about user experience.
- Can you recall a stand out positive user experience you had?
- Can you explain why you felt it was positive?
- Do the same for the negative case.

✓ (Task 18) Regression testing

A big part of software development is getting things to work together and keeping them working together. Even if loads of time and effort go into harnessing these implementations you'll find cracks within the surface.

Regression testing is a technique that tries to find problems in software that has worked before, but is now broken because something was changed.

Many Testing concepts are introduced because of regression and the uncertainty this gives developers and managers. Automated checks, rerunning Test Cases, elaborate smoke tests,... are all in place because of regression issues.

Regression testing involves re-testing features that previously worked to ensure that they are not affected by the addition of new features or refactoring of existing ones.

What to do next:

- Think about a time when one feature in one of your favourite apps didn't work anymore. This might be in a game such as Pokémon Go, Facebook, Netflix application, or any other product that does frequent updates.We've grown quite used to these frequent updates and bugfixes. Additionally, a malfunctioning feature might be fixed tomorrow. Sometimes,

this backfires and users get fed up and discard the product entirely.
- Consider the strategies these companies use to tackle their regression issues and compare them with strategies used in your company. Are they very different? Does that make sense to you? Can you come up with other strategies?
- Do you have something that you can practice regression testing on? Let's think back to that app on your phone. Did it recently update? Have a look around at the new functionality that the app creators say they have added. Has it broken functionality that previously worked? If so report it to them. They might already be aware of it but if not you've helped improve someone's experience.

Also, read the following articles on regression testing:

- Cem Kaner and James Bach have written about Regression testing in great detail
- Thinking about how to get a good regression test set by Patrick Prill
- A heuristic for regression testing with Karen Johnson
- Get smart about your regression tests' value by Leanne Howard
- Read the series from Iain McCowatt rethinking regression

✓ (Task 19) Agile testing

If you're quite new to the software scene you will not have heard of development practices such as Waterfall, Agile or DevOps. These are methods or approaches to software development. A key thing to notice in the diagram below is where testing fits into each of the approaches. DevOps is relatively new to the scene. We'll start with Agile to ease you in and you can research DevOps more later.

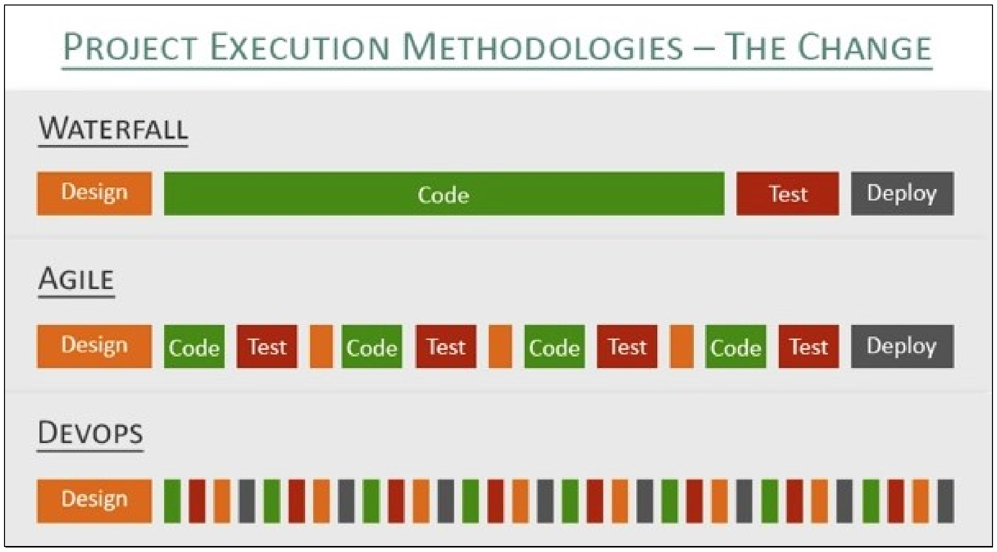


Image Ref: http://blog.spec-india.com/wp-content/uploads/Project-Execution.jpg

I'll direct you back to the books I mentioned on Task 8, specifically the one from Lisa Crispin and Janet Gregory. Elizabeth Hendrickson has a useful overview of Agile Testing to help you get a feel for it. Augusto Evangelisti talks in this interview about how Agile has changed the role of the tester.

Your tasks for today:

- Start reading Lisa and Janet's book
- Watch this talk from Lisa and Janet where they debunk Agile testing myths
- Read Esther Derby's seven Agile best practices
- Read Huib Schoots' what makes Agile testing different?
- Agile Testing by Elizabeth Hendrickson may also be useful to you
- Huib has more resources for Agile testing that you may want to explore

✓ (Task 20) Pairing

Pair testing involves two people working together at the same machine with a single keyboard to test a piece of software. One person has the keyboard while the other suggests ideas or tests, pays attention and takes notes, listens, asks questions, grabs reference material, etc.. There should be a shared goal with this approach and the two people will constantly communicate to ensure that the goal is achieved. Communication is key with this testing approach. Both people involved should have a shared understanding of what they are doing and why they are doing it.

This is easy and fun to do. You only need you and one other person. Katrina Clokie has some excellent material surrounding this from beginning to end. I try to do this as much as I can in work. It is a great way to get a different perspective on your work and maybe even learn a new skill.

What to do next:

- Read the articles from Katrina and find a pairing buddy. If you're not in a software house ask someone from the meetups you attended or off Twitter or Slack, you could even try remote pairing.
- Pair with this person and reflect on your experience.
- What did you learn from it?
- What could be better next time?
- Would you do it again?
- What did you teach the other person?
- Katrina also spoke about this at TestBash Brighton 2016 watch it if you can.

✓ (Task 21) Mob Testing

This is not something you can do by yourself but you should read up about it in the Mob Programming Guidebook. If you can, try to do it in work or get a few people together to try this out. A good chance to do this is at the end of a sprint demo session. Start with a 15-30 minute slot with the whole team. Get them to start asking questions and testing the product as a team. This is a great way to get non testers involved in testing and possibly help generate some new ideas or approaches.

If you're lucky the meetup you attend might try this. If they don't try it, ask them! Any meetup organisers I know are crying out for feedback from the community of what they would like to see. Maaret regularly blogs and speaks about this. It may not be something you can do at the moment but it is definitely something you need to be aware of.

For today:

- Download and read the Mob Programming Guidebook

- Try mob testing if you can
- Maaret has an excellent blog about how mob testing is organised to get you started
- Maaret has also spoken at a Ministry of Testing masterclass and TestBash Philadelphia, watch these if you can

✓ (Task 22) Dark Patterns

Dark patterns are "user interfaces that are designed to trick people". These are well thought out tricks that are meant to benefit a business but rarely consider the user. Have you ever noticed that the option to opt out of mailing lists is usually the smallest item on a page? You wouldn't notice it unless you are really looking for it. These are dark patterns.

Think back to you user experience task. Dark Patterns are a form of anti-pattern to that. They have been a topic in the spotlight of news websites for some time now.

Have you ever tried to sign up for a free trial of something only to be asked for your payment information? Only after reading The Sinister Side of UX did I realise I had fallen victim to this.

Emma Keaveny has an excellent video about this! Another useful link for this is Dark Patterns. Melissa Eaden has also blogged about spelunking dark patterns.

Continuing on from the theme of Task 5, I can highly recommend the book Evil by Design -- Chris Nodder. There is also a website linked to this book which will give you a flavour of what is in the book. I'm still not finished reading this myself but I learn something new every time I pick it up.

Your tasks for today:

- Watch the video of Emma's talk.
- Read the linked content and start to think about dark patterns.
- Can you think of any dark patterns that you have come to accept as the norm in your life? You can use the examples from Melissa as a guide but I'm pretty sure once you start to really think about this you will have a pretty long list.

✓ (Task 23) Conferences

There are many testing conferences out there to choose from. Some even post videos afterwards of the talks. This is amazing, but honestly it is not the full experience. When you attend you get to network, this sounds awful business like but seriously it can be the best part about attending a conference. Another advantage of actually attending is the chance to ask the speaker questions at the end of their talk or find them afterwards to discuss it in more detail. Speakers usually welcome the opportunity to discuss their talk in more detail, sometimes it can even give them ideas about how they might change their talk in the future.

Eric Jacobson and Rob Lambert have both written about how to enjoy and get the most out of testing conferences. These will be valuable to you once you pick your goal conference.

My first software testing conference was TestBash Brighton 2016. I blogged about it from the viewpoint of is it just for testers. Kristine Corbus asked the reader to imagine a conference. Beren Van Daele felt like it was a homecoming. We all took a different angle based on our experience of the day. Cassandra Leung blogged about TestBash Manchester as a testers first TestBash. Danny Dainton has posted twice about TestBash Brighton, first from a volunteer perspective and second as a testimonial to how it changed his career. Dan Billing has also written about his TestBash 2014 experience, he went from experiencing it as an attendee to speaking at it.

TestBash is not the only conference out there. Erik Brickarp kindly linked me his blog from Let's Test 2013, I love the line in it "You are never alone". Helena Jeret-Mäe also wrote about her experience of Let's Test 2014. Chris Kenst has a wonderful in depth look at CAST 2015. Finally, Keith Klain has written about his experience of Agile Testing Days. There are even more conferences out there, these are some first hand experiences of a few to help you get a feel for what to expect.

Going back to Danny's volunteer blog, many conference have some form of volunteer option if you are willing to put the hard graft in. This may help you to achieve your goal so put yourself out there and see.

What to do next:

- Find a testing conference to attend.

- Network online with people who have or will attend the conference.
- Search for existing online videos of previous testing conferences.

✓ (Task 24) Exploratory testing

Exploratory testing is one of my personal favourites. You get to explore areas of the application that you most likely didn't cover with other testing methods or covered, but you felt like you could investigate more. There are some really interesting articles I will advise you to read for this. Most of them are quite recent!

James Bach has written his thoughts about What is Exploratory Testing. Simon Tomes recently wrote an excellent article where he gave three digestible diagrams to describe exploratory testing. I particularly like this article as someone who learns more from diagrams than text. I would also like to highlight the book Explore It! that I mentioned earlier. This book is definitely a go to for exploratory testing. Maaret Pyhäjärvi has a useful article about how to explore with intent. This can help you stay guided in your exploration and have a clearer picture at the end of what you achieved.

What to do next:

- Read the linked articles and start Elizabeth's book.
- When you are done reading them pick something you would like to explore, maybe an app on your phone as you take the train to work.
- Explore with intent for the day!
- Did you find anything unusual?
- Did you feel like you explored it enough?
- Would you approach it differently the next time?

✓ (Task 25) Mobile testing

I would call this a specialisation. I've put it in here so that you have resources to be aware of the specialisation. Maybe you'd like to specialise in mobile testing or your job requires knowledge of it. The Dojo has some resources in the mobile-testing section. Check out this quick introduction to mobile application testing. There are channels on both Ministry of Testing and testers.io Slack for mobile testing where you can ask for guidance if you wish.

Think about the apps on your phone. If you go into the about section for them does it say anything about the operating systems they support? Given what you now know about mobile testing does this surprise you?

When you're finished getting to grips with mobile testing you might want to try the 30 days of mobile testing challenge. Designed by Daniel Knott who is definitely someone you want to follow on Twitter if you plan to pursue this.

What to do next:

- Check out The Dojo's mobile testing resources page
- Do the 30 days of mobile testing challenge
- Check out Daniel Knott's book on mobile testing
- Try Mobile Testing 101 on the Dojo
- Investigate the mobile testing pathway by Katrina Clokie

✓ (Task 26) Security testing

Dan Billing recently released a sandbox to practice Security testing in called Ticket Magpie. He also has a lesson series on the Dojo to get you started. We've all seen the big hacks over the past year or more: TalkTalk, Ashley Madison, 3pay, Tesco. Security testing is becoming more important than ever before.

As the world around us changes and we start to (sometimes unknowingly) put more information out there about ourselves.

This task is just a starter to get a flavour of security testing. Again, it is probably one of those tasks that won't be just a day.

Your mission for today is:

- Watch or read about an introduction to security testing and see if it is something you would like to explore further
- Go and read some of Troy Hunt's work
- Check out OWASP and the incredible resources they have
- Check out the 30 days challenge by Ministry of Testing for security

✓ (Task 27) Automation

Automation is a huge topic of debate and discussion within the software testing community. When you delve into this topic be prepared to hear such things like "automation fixes everything". These perceptions are the possible cons of automation. You may also read things about "testing vs checking", please do not be put off by the language used, I know I found it overwhelming at the beginning. Richard Bradshaw gave a talk at TestBash 2015 about automation in testing, I highly recommend you start there. The Dojo also has a podcast talk with Richard about how to get started with automation. Maybe there are more podcasts related to it that you stumbled across from Task 16.

One of the pros of automation is that it is a way to enhance the coverage you have of your application. You are only one person and you can miss things in re-checking of features, you are human. Computers can be very good at doing some things that people often fail at, Angie Jones wrote about the things that can be missed by automation if it is scripted straight from a test case.

Possible tools to use for automation are varied and can depend on the type of application you are working with. One of the main things I struggle with for automation is how do you decide what to automate.

If you want to give automation a try I recommend checking out the course from Richard Bradshaw or Alan Richardson to get you started. Of course Selenium is not the only automation tool out there, but I have personally taken these two courses and can highly recommend them. Mark Winteringham also has a three part blog series about building an automated API test framework. Have you found any others? Did you find them useful? Share them with others looking for these resources.

✓ (Task 28) Performance testing

There are quite a collection of resources about performance testing on the Dojo. Think back to that app on your phone and the software package in work from Task 1. It is estimated that the average user loses patience with a web page that doesn't load after 4 seconds. Now think about Black Friday sales or those concerts that sell out in

2 minutes. Have you ever been in an online queue for either of these? Load and performance testing can help the whole team to get an estimate of possible weakness in the product before it is released to customers.

In the lesson series Performance Testing 101 on the Dojo, Simon Knight gives a useful breakdown of these terms in the performance testing patterns video. I'll provide a brief summary here for context, but I recommend watching the lesson series for a more detailed explanation. I have left out Soak/Endurance, Stress, Spike and Capacity testing, which Simon covers in the video.

Performance testing aims to determine how responsive an application is under **normal** load conditions. Examples of questions that performance testing tries to answer are how long a transaction takes to complete or a page takes to load. It can help teams see if a site can handle the projected number of users. A performance test in a tool like Jmeter would keep the number of users at a normal level, where normal would be based on the requirements for the system.

Load testing is done to determine how the system responds under **heavy** load conditions. It would measure response times and resource utilisation under this heavy load. Load testing also helps to determine when the system breaks and if it is graceful when it does so. A question we may like answered when we do load testing would be how many transactions are we actually capable of dealing with when the system is under heavy load. When running a load test the level of users is kept high to get an idea of how the system will respond under the level of usage.

What to do next:

- Check out PerfBytes podcast
- Mark Tomlinson also featured on Test Talks with a bucket full of bottlenecks
- Read this Load Test Article by Scott Stirling which gives a nice breakdown of load, background, stress and performance testing
- Watch Performance Testing 101 on the Dojo
- Investigate the JMeter User Manual

✓ (Task 29) Ask a question

After all this information you probably have a lot of questions.

Ask a tester a question.

You can ask on Twitter, Slack or at a meetup. Expand the question into a discussion if you can. You never know what you might learn from it. This sounds like a simple topic for a whole task but it can be a daunting task for some. If you expand the question into a discussion it could continue on long past this one task. If you need help with where to start with questions check out TestSphere on Twitter.

- Take a seemingly simple testing concept. Talk to someone about it. Ask for their understanding and explain it for yourself. What aspect do you stress and how does that differ from the other person's explanation?
- Did a blog or podcast cover the topic you have questions about? Leave a comment for the host to answer.

✓ (Task 30) Share your experience

People and even just yourself, will find inspiration and strength from writing about your experience, even if it is not for public consumption. Writing is a great form of self retrospecting to better plan for the future.

Think back to the first task you completed, why you were here, what you hoped to learn. Think about all the things that you have learned. Were any of them particularly useful for you? Why? Did you find any uninteresting? You may be surprised at how useful others will find it if you share some of your experiences.

How to share your experience:

- Tweet about each task with the #starttesting hashtag
- Post updates to the #starttesting channel on Ministry of Testing Slack
- Write a blog post
- Write an email and send it to someone you may feel would benefit
- Maybe even submit your own 99 second talk to the Dojo

About Heather Reid

Heather is a tester at Exploristics. Her passion is helping the software testing community in Belfast. She is a co-organiser of the Belfast testers meetup and of TinyTestBash Belfast. This all started when Heather discovered the Ministry of Testing through TestBash 2016. When she's not testing she's usually exploring or working on restoration projects. You can find Heather on Twitter where you will also find a link to her blog

Pro Subscribers Get More Want in? Sign Up.

Toggle navigation

- Terms
- Privacy
- About