Software testing is a process of searching for software errors. Have you ever stumbled upon some silly bug on some of your favourite website, and was wondering how it got there?

The other day I went to McDonald's and used their kiosk to order some healthy food. After changing from one language to another, the text within one of the buttons did not fit the container any longer. The error looked silly and would be an easy fix for developer, but it was there. And sometimes silly errors can be a big backout criteria for user retention.

Software companies spend much money on software developers, but then most of those companies spend quite a bit on making sure their products are working correctly by hiring armies of software testers or QA engineers.

When you are looking for a job as a tester, you would likely see two terms used interchangeably: Software Testing and Quality Assurance. Although, many companies use the two terms as synonyms, they are different things. Let's get a sneak peek of what they mean in the software world.

Quality is how far the product is from what it should be.
Software testing is finding bugs in the code that is testable. Meaning that the code is already written and represents some part of the product, and you can run and interact with that part.
Quality Assurance is taking a look at the process of software development and the product itself. It aims at improving the process in order to increase the quality of the final product. It is rather about structures and rules than the code.

With these differences noted, realise that most likely the job description you are looking at would say they are looking for a QA engineer, while they are actually looking for a software tester. Yet let's commit to the ignorance and use these two terms interchangeably in this course.

Why work in this field

Let us start with the money aspect of the profession. According to payscale.com, the entry level median for the U.S is $50,315. The national average overall (not only for the entry level professionals) is $60,728 per annum, as stated by glassdoor.com. Not so bad for a job that does not usually require a higher education, right? In fact, even if the company you are interested in marks a higher education as a requirement in the job description, they probably don't need it

anyway. You cannot get a degree in software testing as such, only courses, so don't let silly requirements fool you.

The IT market is booming, small startups become unicorns and their founders worldwide celebrities. However, in the beginning startups do not have much money and many benefits that they can provide to employees, therefore, many experienced professionals ditch startups for safe nine to five jobs with good salaries. This opens an opportunity for less experienced people to come and join the whatever revolution that company is undertaking. Startups tend to grow very fast as well, which means you can rapidly grow with them.

If you are interested in IT overall, a quality assurance job can be a great gateway position to the tech world and other roles within it. You will have to learn so many aspects of your company's products, that would sometime find yourself to be the only holder of some information. For example, developers usually do not need to know everything about the system, only the parts they are working with. Perhaps a few senior software engineers who have the whole system architecture in their heads would know the most of the answers. People from other departments have probably never wandered in many areas of the system, so sometimes, colleagues would come to you for answers. Use it to your benefit.

You would be mostly working with developers, you will learn many tech terms and get a good understanding how software development works. If you are eager to become a developer later on, such job would a nice starter.

Testing types

There are different kinds of testing. Each of them comes in handy in specific situations and usually is used along with a few other ones. Let's take a look and what the different testing types are.

Positive testing - you must be happy while testing. Joke. This type of testing assumes that you are doing what a user is expected to do. For example, when testing a field for a year of birth, you would provide a number representing some year. Easy. This type of testing is a must to make sure that some functionality of the product works. Works under the condition that a user is doing the right thing.

Developers would usually do positive testing when they are checking that what they wrote works, mostly because it is one of the least time consuming testing methods. Positive testing helps to catch bugs that are lying on the surface.

Negative testing - aims at discovering if the product can handle a user doing something they are not supposed to. Coming back to the example with a year of birth, how would the system behave if instead of a number, a user were to provide a sentence, collection of random characters or even trying to paste a picture of their id card. You never know what they can come up with, and you need to try to predict everything and make sure the system will output a correct error message.

This type of testing involves significantly more work, as you need to look at the problem from many different angles, yet it is likely to bring more results than other types of testing and, therefore, can be considered the most productive way of finding bugs.

White box testing - white box testing assumes you can understand and use the code to detect a problem. For example, you see that a reset password link in the system you are testing is not represented correctly. You know that the front-end part is written with React.JS which you understand, so you look into the code and try to figure out where the problem is. As you are not a developer, you don't need to fix the problem, however, the ability to point the exact place where something went wrong is highly valuable.

Black box testing - testing done without access to the code. You might have the access, but you are not using it and just checking the functionality from the user perspective. This type of testing is common for the most software testers. Testers who understand programming languages well are quite rare.

Unit testing - making sure that the software functions as designed on the unit level (a unit being the smallest testable software part). This is done purely programmatically and is the major testing that developers do. Usually a developer who builds a component would be responsible for writing a unit test for it.

Functional testing - this testing is done to ensure that the software meets business requirements. You would be testing against the specifications and making sure that the system provides correct outputs for the inputs you supply.

Regression testing - this testing type aims at making sure that what used to work still works. It often happens, that by fixing one thing, a developer breaks another. Therefore, whenever some functionality is changed, it is necessary to check other things around that can be related to it. I would also ask a developer what parts were touched during the fix, and would test them.

Acceptance / smoke testing - this type of testing is usually done before a new version of the application is released to the production environment. The testing type aims at making sure the condition the application is in now is good enough to be shipped to customers.

Ad hoc testing - testing performed without planning or documentation, sometimes referred to as a monkey testing. Although, might sound useless, it can help to discover issues that are impossible to find when using existing formal documentations and guidelines your company has.

Exploratory testing - this type of testing could be thought of as a thinking activity. As with ad hoc testing, you are not following any beforehand written test cases, but you have a pattern in your head that you are trying to follow.

QA Business Systems

Working with the QA business systems is more from the quality assurance area as they allow to sustain a high quality of the product. I suggest to make a research what systems are the most common and perhaps what the company you applying for a job is using. If the QA department is just forming, it is likely the company has not made their final decision on what systems to use. Your knowledge of QA business systems can help the company to make a good decision and position you as a knowledgeable person.

Here are some of the common types of business systems:

1. **Test planning** - allows to write documentation for test suites and test cases, track how many times the functionality was tested and how often certain bugs occured
2. **Bug reporting** - a system that allows to report bugs, usually used with the next category of business systems

3. **Bug tracking** - helps to keep control of the reported bugs, nowadays would usually have an agile board where the progress can be tracked
4. **Test automation** - software that helps to automate and execute tests. This could be a wide range of software products as there are many different types of test automation

GUI testing

To find functional errors, you need to know the expected result and some cornerstones of the system, but if you are new in your workplace, you can start with **graphic user interface (GUI)** testing, which refers to problems the product has visually.

GUI can help you quite a lot during the first days of work, or perhaps even during the interview.

If you manage to find 10-30 bugs in the first day of work, even though the bugs are not functional but related to the appearance of elements, your new managers will be astonished.

Where and how the elements should be presented throughout the system cannot be random. In fact there are a few guidelines for the quality. If your company has no specific guidelines provided (or even if it does) read through the Google's https://material.io/design/ and through the official U.S government's usability standards https://guidelines.usability.gov/

Both of the guidelines are research-driven and can be very helpful to your future employer. You would be surprised, but chances are that many developers or managers have never read those guidelines, so here is a chance for you to be a clever one. Do yourself a favour and read through the research, it will be very helpful in the future.

What would you be looking for when testing graphic user interface?

- **Consistency**

This is what helped me to surprise employers in my first testing job on the first day of work, as well as keep finding bugs while I was just beginning to learn about the product. Does the company have a rule to capitalise a first letter of every word in a title or do they only use lowercase letters? What fonts are used for titles, which ones are used for citations? There always should be a consistency. For example, one project I was looking into did not have a clear policy (or perhaps the policy was ignored by developers in the beginning) on capitalising letters

in articles. That led to having titles like these next to each other: 'Example Example' and 'Example example'. We needed the second option, but the first one was present almost everywhere. You might not realise it in the beginning, but such little bugs are very annoying and can harm the user experience quite severely. Also, look for consistency across **text formatting, abbreviation colours, fonts and icons**, and anything else you might want to keep consistent.

- **Overlapping elements** - often happens when you change the screen size or zoom in or out.

One element would go on top of another.

- **Alignment of elements** - some elements that should be aligned are not aligned.
- **Intuitivity** - you don't want user to spend minutes to figure out how to use your product's
- **Grammar and spelling -** seems to be self-explanatory, but sometimes underestimated, like in this course.
- **Delivery of error messages -** if a user is doing something wrong, the system should let them know what exactly is wrong.

Software Development Life Cycle

Software development life cycle is a structured approach to how software is designed, developed and tested. This comes from the area of project management, and understanding how it works in your company is important to adjust your work activities as a tester.

There are a few common models in the tech world. One of the older yet still popular is the Waterfall model. Another, that is more common nowadays is the Agile model. There are some similarities and differences between the two. It might be helpful for your interview to know a little bit about the two. We are not going to cover them in this course, but I suggest to do some research.

In order to become a good software tester, you need

1. Learn the product well. Understand what the product should do, how it should do it and why.
2. Find and report bugs
3. Once the bugs are fixed, confirm they actually were, and the fixes did not break anything else

4. Check that the fix didn't break anything else.

A Day at Work

Let's take a look at the day in a life of a software tester. Your schedule will usually depend on the software development life cycle. For example, many companies deploy their changes to the production environment once a week, this would mean your every week at work would have a similar schedule. You would likely need to devote what activities need to be done on a particular day of the week. Your job is to make sure that as few bugs as possible are released to the production.

- Come in the morning and check what was happening during the night (if any new bugs were reported, fixed, features added etc).
- Some companies would have daily meetings that you as a QA person need to attend. Yet it's likely that you will have only two or three meetings to attend during the week. Meetings are extremely crucial to synchronise everybody in the company and ensure that everybody is on the same page. That being said, many tech people despise meetings as they might feel it is a waste of their time. As you will mostly work with developers, I would suggest not to be too enthusiastic about spending hours in meeting rooms as it might suggest you are looking for an excuse to procrastinate.
- Most of the day will be dedicated to finding new bugs and making sure the reported bugs are fixed. Before the deploy, there is usually a period called **feature freeze**, it is when no new features are added to the product, and the current product's condition is amended only if there are bugs in it that need to be fixed. Otherwise, this is what will go to the production and what users will use. Once feature freeze happens, you need to make sure that the product is in a good condition and can be shipped to users. This is essentially the **acceptance** testing mentioned earlier. You should have a structured documentation of **critical functionality**: something that absolutely must work for the user and if it breaks, it would cause massive problems. You would go through this list during every feature freeze and it is likely to take a few hours, depending on your company's product(s).
- It is likely, especially in the beginning, that you will not understand the meaning of some issues that are being closed by developers. In such case, you would usually go (or send a slack message) to the developer who was working on the issue. Depending on the amount of work and complexity of the issues, you might end up spending the most of your day like that.

Writing a Bug Report

Another crucial part of the job is an ability to write good bug reports or opening issues using your company's bug reporting system. If the QA department in your company is very new, it is likely that they have not stopped on something in particular. Many companies would use GitHub for opening issues or other systems such as YouTrack. I recommend to take a look at what systems exist, perhaps even try their demos so that you appear confident during the interview and your first days of work.

The more concise your bug report is, the more developers will love you. Nobody wants to waste their time reading novels about how errors occurred.

Make sure to use correct terminology. If you are not yet familiar with it yet, devote some time to it. Otherwise, you are risking having quite a bit of misunderstanding with the developers.

1. **Describe the problem that occured -** give a very brief description of what happened. One or two sentences should be enough.
2. **Provide additional details -** you should aim to make life of people who will be fixing the problem much easier. When testing a web application, specify what browser you used, what version of the browser. If the problem occurs only in some particular browser, I'd recommend to mention it in the title or/and issue description. Your team might not be aiming at supporting that browser, so stating where the error happened might save developers and project managers time. If the product has different test environments, specify which you used.
3. **Provide steps to reproduce -** list steps needed to reproduce the bug. Make sure to minimise the steps and keep it as concise as possible, yet still be easy to follow.
4. **Suggest a solution -** a good bug report will also suggest a solution to the reported problem. If you have a strong opinion how it should be fixed, state it. If you are not sure, sometimes it is better to skip this step and allow others to come up with the solution.
5. **Specify the importance of the bug -** usually companies will have this in place and the marking will include labels such as *blocker*, *normal*, *easyfix* or something similar. This is necessary for better resource allocation. The company would not want their only developer spending time on changing dot to an exclamation mark for some element

(*easyfix*), while there is something that prevents the whole system from proper functioning (*blocker*).