**AUTOMATED TESTING**

The practice of **writing code to test our code, and then run those tests in an automated fashion. (using test scripts).**

PROS:

* You can test your application code on a frequent basis and in less time
* You can catch the bug before deploying your action/reduce number of bugs that will go in the production

Source code consists of:

* Production code
* Test code

Verifying that the function returns?

MANUAL TESTING

* Launch the app
* Login
* Navigate
* Fill out a form
* Submit it
* Verify the result

CONS: time consuming

**Refactoring** – changing the structure of the code without changing its behavior, to make it cleaner and more maintainable, without changing its functionality.

With automated tests, every time you refactor your code, you run your tests and make sure you didn’t break anything that used to previously work.

It makes you focus more on the quality of the methods that you’re writing.

**TYPES OF AUTOMATED TESTS:**

* **Unit**
* **Integration**
* **End-to-end**

**UNIT tests** – tests a unit of an application **without** its external dependencies, such as files, databases, message queues, web services and so on. They test a unit of work. **Smallest part of a software application is called a unit.**

* You can run 100s of them in a matter of seconds
* They are cheap to write and execute fast
* Do not give much confidence

External dependencies –

Example: When you register for a website there is a unit of code that checks your email, by verifying it has an @ character in it.

Multiple units will form a component. Any testing within this component will also be called unit testing. A modern web application consists of lots of components like this that communicate to each other. E.g.: ***user registration, user login or the database*** in which the users are saved.

**Integration test** – tests the application **with** its external dependencies. Basically, testing that **two or more components (of units) work together.**

* Take longer to execute (often involve reading or writing to a database) and less stable
* Because they test more of the application, so more things can go wrong,
* Give more confidence in the health of your application

Takes one or a few units or classes and test their behavior as a whole.

Example: if you test that when a user registers he is added to the database.

**End-to-end test –** drives an application through its UI (user interface).

Most modern applications also have UI like a website or a phone app. Testing that most of the components interact correctly, usually done through the UI, is called **End-to-end testing**.

Selenium – a tool, which allows us to record the interaction of a user with our application and then play it back and check if the application is returning the right result or not.

* Gives the greatest amount of confidence about the health of your application
* Best at simulating a real user

2 big problems:

* Very slow – launching the application, testing it through the UI, logging in, navigate to internal page, submit a form and inspect the result
* Very brittle - a small change in the user-interface can easily break these tests.

**Rule of thumb (recommended):**

* Unit testing – 70%
* Integration testing – 20%
* End-to-end testing – 10%

**A unit test is a way of testing a unit -** the smallest piece of code that can be logically isolated in a system. In most programming languages, that is a function, a subroutine, a method or property.

**Class testing** is the base of object-oriented software testing. It involves three aspects: testing each method, testing the relations among class methods and testing the inheriting relation between class and subclass.

Unit tests vs class tests: <https://blog.arkency.com/2014/09/unit-tests-vs-class-tests/>

**A user interface**

Also called a **"UI"** or simply an "interface," is the means in which a person controls a software application or hardware device. A common example of a hardware device with a user interface is a remote control.

Nearly all software programs have a **graphical user interface**, or **GUI**. This means the program includes graphical controls, which the user can select using a mouse or keyboard. A typical GUI of a software program includes a menu bar, toolbar, windows, buttons, and other controls.

**The Macintosh and Windows operating systems have different user interfaces, but they share many of the same elements, such as a desktop, windows, icons, etc.** These common elements make it possible for people to use either operating system without having to completely relearn the interface. Similarly, programs like word processors and Web browsers all have rather similar interfaces, providing a consistent user experience across multiple programs.

**Which Test Cases to Automate?**

Test cases to be automated can be selected using the following criterion to increase the automation ROI

* High Risk – Business Critical test cases
* Test cases that are repeatedly executed
* Test Cases that are very tedious or difficult to perform manually
* Test Cases which are time-consuming

**The following category of test cases are not suitable for automation:**

* Test Cases that are newly designed and not executed manually at least once
* Test Cases for which the requirements are frequently changing
* Test cases which are executed on an ad-hoc basis.

**Automated Testing Process:**

Following steps are followed in an Automation Process

**Step 1)** **Test Tool Selection\***

**Step 2)** **Define scope of Automation** - the area of **AUT** that which will be automated.

**Step 3) Planning, Design and Development.** Creating the automation strategy and plan.

**Step 4)** **Test Execution.** Automation scripts are executed.

**Step 5)** **Maintenance.** When functionalities are added to SUT (System Under Test), automation scripts need to be added, renewed **for each release cycle.**

**\*Automation Testing Tools:**

Test tool selection largely **depends on the technology the Application Under Test is built on.** It’s good to conduct **Proof of Concept of Tool on AUT** (Application Under Test). POC in Automation Testing is usually conducted when the team wants to adopt a new automation testing tool. As a part of the first phase of the automation transformation, the team may consider implementing a POC after [evaluating and selecting an automation tool](https://www.katalon.com/resources-center/blog/automation-testing-tool-strategy/).

* Ranorex Studio
* QTP (leader in Fuctional Testing Tool)
* Selenium
* Rational Robot (IBM tool used to automate regression, functional and configuration tests)
* Kobiton
* Zaptest
* Lambdatest
* Avo Assure

**Criterion to help you select the best tool for your requirement:**

* **Ease of Use (Scripting Language used)**
* **Support for various types of test, including functional, test management, mobile, etc.**
* **Support for multiple testing frameworks**

**HOW TO SELECT YOUR TOOL**

1. Identify the requirements, explore various tools and its capabilities
2. Set the expectation from the tool and go for a POC (Proof Of Concept)

**Framework – a set of automation guidelines** which help in **maintaining consistency of testing, less maintenance of code,** **improves re-usability.** In other words, a framework is a **kind of template, scheme or technology-based conceptual structure** that allows working in a much simpler way. In this way, possible programming errors are avoided.

**Four types of framework used in Software Automation Testing:**

1. **Data Driven** Automation Framework
2. **Keyword** Driven Automation Framework
3. **Modular** Automation Framework
4. **Hybrid** Automation Framework

**Best Automation Practices:**

* **Scope of Automation** needs to be determined in detail before the start of the project.
* **Selecting the right automation tool**
* **Choosing an appropriate framework**
* **Scripting standards**. Standards need to be followed while writing the scripts for automation.
* **Measure metrics.** Success of automation cannot be determined by comparing the manual effort with the automation effort, but by also capturing metrics like **percent of defects found, productivity improvement, etc.**

**The scope of automation** is the area of your Application Under Test which will be automated. **The extent to which automation is used** in the testing process defines the scope of automated testing. Among system tests, automation best **facilitates security, configuration, and load testing**.

**Software testing metrics** are a way to **measure and monitor your test activities**. More importantly, they give **insights into your team's test progress, productivity, and the quality of the system under test.**

Reference: <https://www.guru99.com/automation-testing.html>

<https://medium.com/katalon-studio/proof-of-concept-in-automation-testing-best-practices-for-implementation-7f95f8083ff7>

**Automated testing cons:**

* Lose their value if you keep on piling them. They start to take longer and longer. Affects its properties of being fast.
* A high number of tests = more maintenance is required. Each time a functionality is changed, you also need to update the automated test that verified that functionality.
* A lot more up-front work compared to manual testing. Takes more knowledge to instruct the test tool how to test compared to manually running it.

**If you would run it once, keep it manual. If you run it repeatedly, an automated test is way faster.**

**IS A TEST WORTH AUTOMATING?**

* Will the test be run often?
* Will the same test run on multiple browsers?
* Does the functionality test have an important business impact?

**Functional testing**

In the simplest words, functional testing checks an application, website, or system to ensure that it is doing exactly what it is meant to.

In the planning stages, every project creates a document listing functional or requirement specifications. Essentially, it is a list of what the app/system/website is supposed to do, from a user’s perspective.

**Functional testing** is the process through which QAs determine if a piece of software is acting in accordance with pre-determined requirements. It uses black-box testing techniques, in which the tester has no knowledge of the internal system logic. Functional testing is only concerned with validating if a system works as intended.

Reference: <https://www.browserstack.com/guide/functional-testing>

**QA vs QC (testing)**

Your goal of **testing** is to ensure the functional **requirements** and **metrics** are met. This could be computational performance, clicks-per-action, user throughput, feature-completeness, defect rate, etc.

**Quality-Assurance** is a broader-reaching goal that simply asks "Is the **customer** going to be **happy** with this?" Obviously, you need to meet the metrics and requirements detailed in testing in order to achieve this. But there's a certain emotional aspect that testing can't cover. Is it pretty? Is it innovative? Does it get the job done? (It can meet every requirement and still not get the job done!)

**Quality Assurance** - **Prevention** of faults by inspecting & testing the **process**  
  
**Quality Control** - **Detection** of faults by inspecting & testing the **product**

**How do you Inspect and Test the *Process...***  
You might inspect the process **by auditing what people are doing and ensure they are within guidelines,** frameworks or following your (agile) processes... also, making sure that, when cutting corners, the benefits and the risks have been considered... You might *test* the process by, for example, error seeding to see how effective the testing (Quality Control) processes are.  
  
**How do you Inspect and Test the *Product(s)...***  
As you generate artifacts through the SDLC, these can be inspected (e.g. on the fly as in Pair Programming) and/or tested (e.g. with automated acceptance tests)...  
  
The term "Quality Control" comes from more mature industries that would discard a product or correct its faults (if possible) if it fails certain tests... it isn't really "Control" because all it does is provide information. it is the information that facilitates decision making - hence providing controls... In some industries, these decisions are more easily determined from the outcome of tests... we all know that this isn't always so clear with software...

Reference: <http://web.archive.org/web/20120807142038/http://www.testingreflections.com/node/view/827>