Module 6

Software Testing

Software Testing?

- Software testing is the process of evaluating and verifying that a software product or application does what it is supposed to do.
- The benefits of testing include preventing bugs, reducing development costs and improving performance.

Why software testing is important

- Few can argue against the need for quality control when developing software.
- Late delivery or software defects can damage a brand’s reputation — leading to frustrated and lost customers.
- In extreme cases, a bug or defect can degrade interconnected systems or cause serious malfunctions.

- Though testing itself costs money, companies can save millions per year in development and support if they have a good testing technique and QA processes in place. Early software testing uncovers problems before a product goes to market. The sooner development teams receive test feedback, the sooner they can address issues such as:

- Architectural flaws
- Poor design decisions
- Invalid or incorrect functionality
- Security vulnerabilities
- Scalability issues

Cases:

- Consider Nissan having to recall over 1 million cars due to a software defect in the airbag sensor detectors.
- Or a software bug that caused the failure of a USD 1.2 billion military satellite launch.
- The numbers speak for themselves.
- Software failures in the US cost the economy USD 1.1 trillion in assets in 2016. What's more, they impacted 4.4 billion customers.

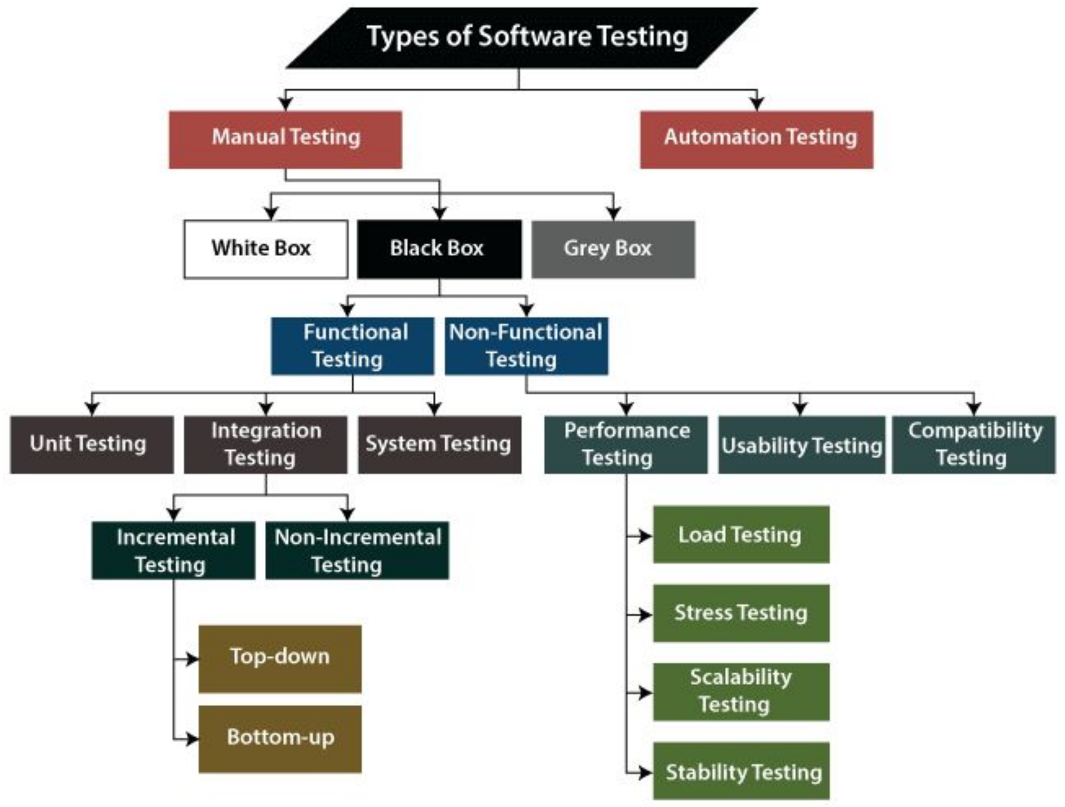
Software testing best practices

- Software testing follows a common process.
- Tasks or steps include

1. defining the test environment,
2. developing test cases,
3. writing scripts,
4. analyzing test results and
5. submitting defect reports.

- Testing can be time-consuming.
- Manual testing or ad-hoc testing may be enough for small builds. However, for larger systems, tools are frequently used to automate tasks.
- Automated testing helps teams implement different scenarios, test differentiators (such as moving components into a cloud environment), and quickly get feedback on what works and what doesn't.

Types of Software Testing:



- 1. Unit tests
- Unit tests are very low level and close to the source of an application. They consist in testing individual methods and functions of the classes, components, or modules used by your software. Unit tests are generally quite cheap to automate and can run very quickly by a continuous integration server.
- 2. Integration tests
- Integration tests verify that different modules or services used by your application work well together. For example, it can be testing the interaction with the database or making sure that microservices work together as expected. These types of tests are more expensive to run as they require multiple parts of the application to be up and running.

- 3. Functional tests
- Functional tests focus on the business requirements of an application. They only verify the output of an action and do not check the intermediate states of the system when performing that action.
- There is sometimes a confusion between integration tests and functional tests as they both require multiple components to interact with each other. The difference is that an integration test may simply verify that you can query the database while a functional test would expect to get a specific value from the database as defined by the product requirements.

- 4. End-to-end tests
- End-to-end testing replicates a user behavior with the software in a complete application environment. It verifies that various user flows work as expected and can be as simple as loading a web page or logging in or much more complex scenarios verifying email notifications, online payments, etc...
- End-to-end tests are very useful, but they're expensive to perform and can be hard to maintain when they're automated. It is recommended to have a few key end-to-end tests and rely more on lower level types of testing (unit and integration tests) to be able to quickly identify breaking changes.

- 5. Acceptance testing
- Acceptance tests are formal tests that verify if a system satisfies business requirements. They require the entire application to be running while testing and focus on replicating user behaviors. But they can also go further and measure the performance of the system and reject changes if certain goals are not met.

- 6. Performance testing
- Performance tests evaluate how a system performs under a particular workload. These tests help to measure the reliability, speed, scalability, and responsiveness of an application. For instance, a performance test can observe response times when executing a high number of requests, or determine how a system behaves with a significant amount of data. It can determine if an application meets performance requirements, locate bottlenecks, measure stability during peak traffic, and more.

- 7. Smoke testing
- Smoke tests are basic tests that check the basic functionality of an application. They are meant to be quick to execute, and their goal is to give you the assurance that the major features of your system are working as expected.
- Smoke tests can be useful right after a new build is made to decide whether or not you can run more expensive tests, or right after a deployment to make sure that they application is running properly in the newly deployed environment.

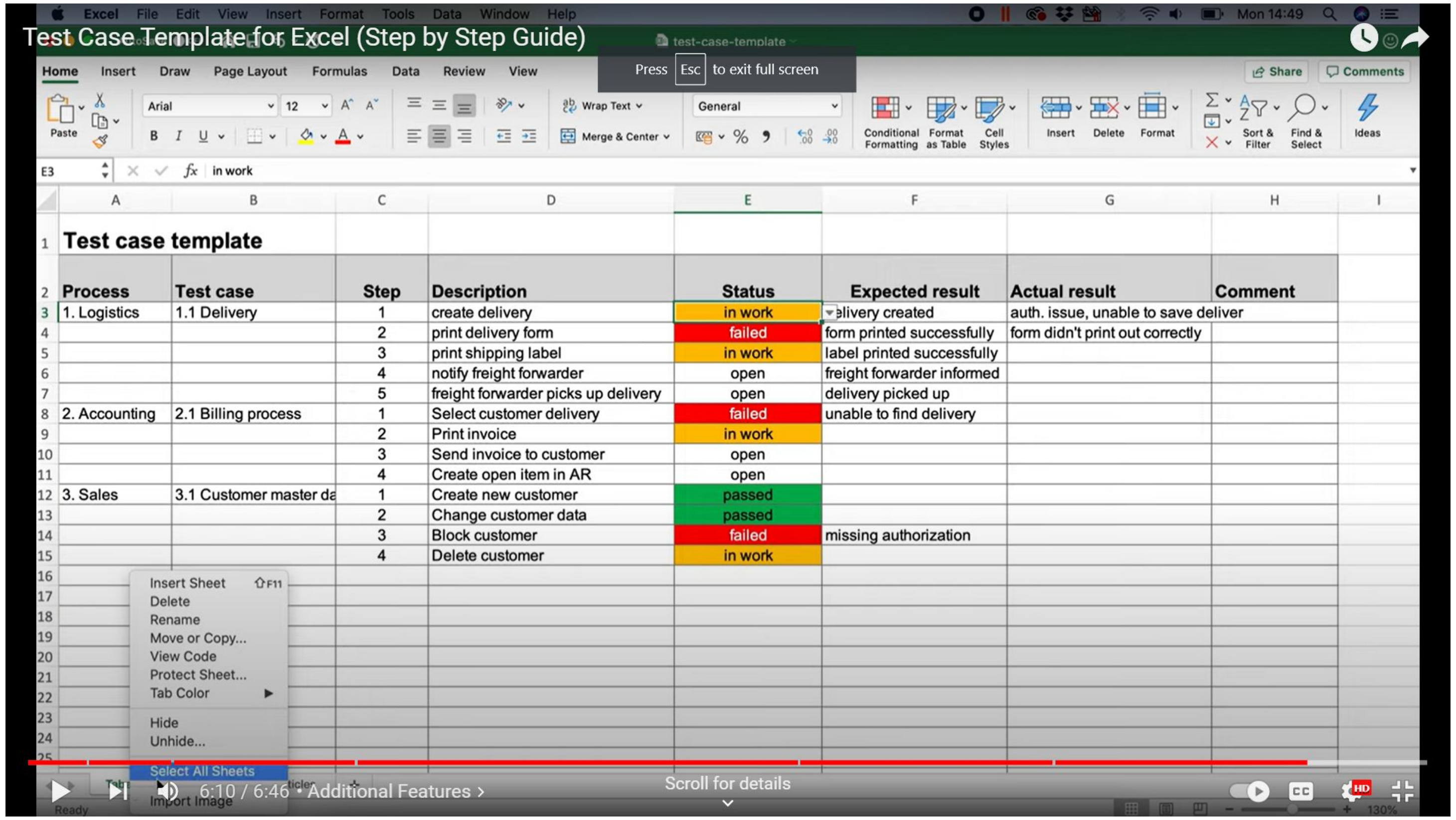
Exploratory testing

- The more features and improvements go into your code, the more you'll need to test to make sure that all your system works properly. And then for each bug you fix, it would be wise to check that they don't get back in newer releases. Automation is key to make this possible and writing tests sooner or later will become part of your development workflow.
- So the question is whether it is still worth doing manual testing? The short answer is yes and it might be best to perform exploratory testing to uncover non-obvious errors.
- An exploratory testing session should not exceed two hours and should have a clear scope to help testers focus on a specific area of the software. Once all testers have been briefed, various actions should be used to check how the system behaves.

Manual Testing

- https://www.youtube.com/watch?v=AjkYTJklAa8

Test Script using Excel: https://www.youtube.com/watch?v=F3fkRnqSj1A



Test case template

Process	Test case	Step	Description	Status	Expected result	Actual result	Comment
1. Logistics	1.1 Delivery	1	create delivery	in work	elivery created	auth. issue, unable to save deliver	
		2	print delivery form	failed	form printed successfully	form didn't print out correctly	
		3	print shipping label	in work	label printed successfully		
		4	notify freight forwarder	open	freight forwarder informed		
		5	freight forwarder picks up delivery	open	delivery picked up		
2. Accounting	2.1 Billing process	1	Select customer delivery	failed	unable to find delivery		
		2	Print invoice	in work			
		3	Send invoice to customer	open			
		4	Create open item in AR	open			
3. Sales	3.1 Customer master da	1	Create new customer	passed			
		2	Change customer data	passed			
		3	Block customer	failed	missing authorization		
		4	Delete customer	in work			

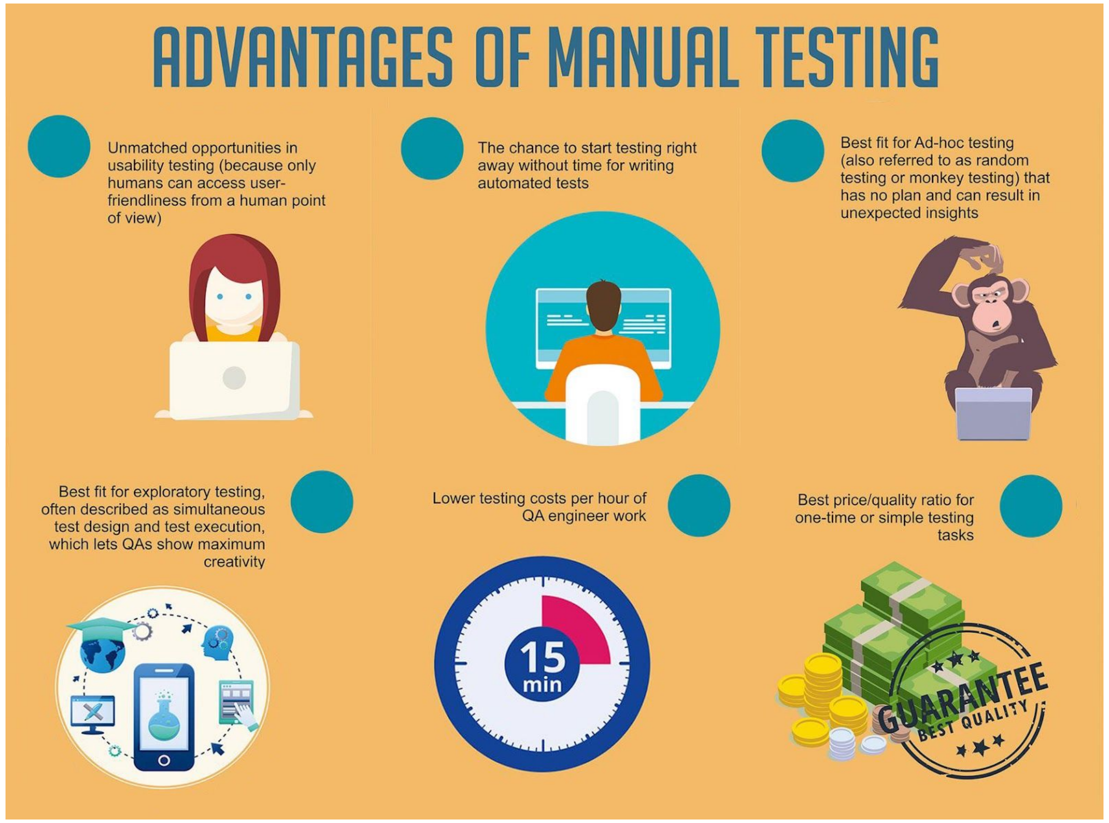
Manual Testing:

- Manual software testing is when human testers check the quality of a new application without using automation tools or scripting. The purpose is to identify bugs or defects, ensure the product is error-free, and check it conforms to specified functional requirements.
- The process compares the behavior of a software application (or one of its components or features) with the expected behavior which was defined in the initial phases of the software development life cycle (SDLC).
- Manual testers design test cases or scenarios with 100 percent test coverage, and execute them one by one before verifying the results. They ensure that any reported issues are passed to the development team to fix, and then tested again.
- One of the fundamental principles of software testing is that 100 percent test automation is not possible—so manual testing is an essential part of your quality assurance process.

What is a manual testing example?

- Manual testing has many real-life applications, and is especially handy for assessing usability and accessibility. For example, if you were launching an ecommerce website, you would need to check things like:
- Optimization for a range of browsers and devices
- Smooth checkout process
- Fast-loading hi-res images
- Links to social media channels

- Under manual testing, testers would check the codes that drive each of these functions to ensure they work as the client intends.
- Manual testers are also able to comment on the look and feel of the website, evaluating it from the user's perspective.



ADVANTAGES OF MANUAL TESTING

Unmatched opportunities in usability testing (because only humans can access user-friendliness from a human point of view)

The chance to start testing right away without time for writing automated tests

Best fit for Ad-hoc testing (also referred to as random testing or monkey testing) that has no plan and can result in unexpected insights

Best fit for exploratory testing, often described as simultaneous test design and test execution, which lets QAs show maximum creativity

Lower testing costs per hour of QA engineer work

Best price/quality ratio for one-time or simple testing tasks

Pros of Manual Testing:

- **1. Accurate**
- Automated tools are smart, but they're not as smart as humans. There are certain things that only a real person with real-world experience can spot. So when it comes to identifying bugs and glitches in software, manual testing is more likely to catch 'em all.
- **2. Gives human insight**
- Manual software testers bring a valuable human perspective as well as accuracy, by focusing on the look and feel of a product. They can evaluate the application's visual components and highlight UI and UX issues by adopting the mindset of the end user.
- **3. Adaptable**
- The manual method is particularly useful in ad-hoc testing, as it's easily adaptable when unplanned changes are made to software. The human input means test cases can be redesigned or tweaked with relative ease. Manual testing is also agile enough to be performed on all kinds of applications.
- **4. Saves money**
- Although manual testing requires skilled labor, it can actually save your company money as it doesn't need any expensive tools. Automation tools can be costly to install and will take time to set up and learn.

Cons of Manual Testing:

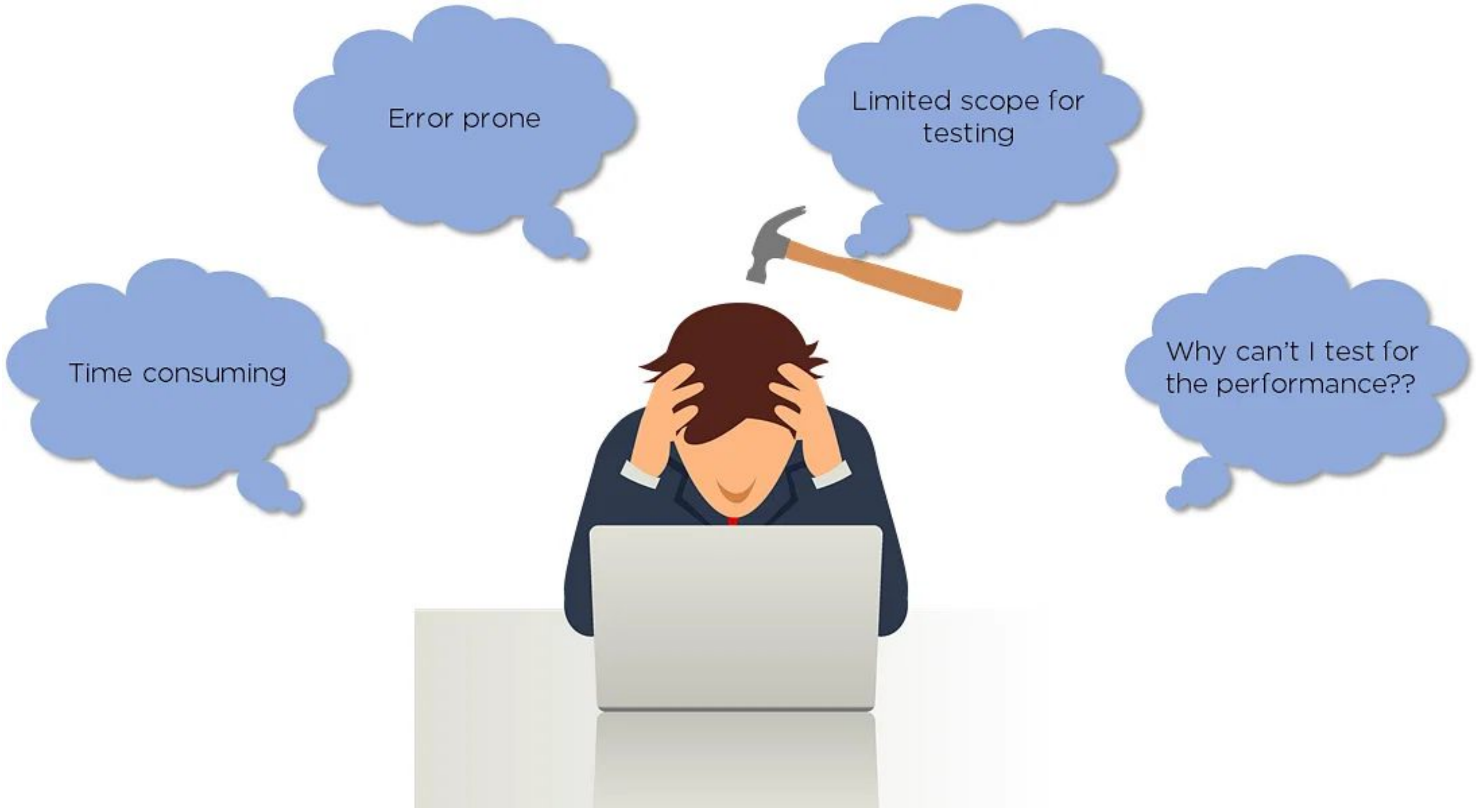
- **1. Resource-heavy**
- Manual testing is undeniably more time-consuming than automation, which means the testing process is slower and can sometimes be more costly. It also requires a large number of human resources, with testers requiring high analytical and creative skills.
- **2. Not always suitable**
- Certain types of testing, such as performance and load testing, are not suited to manual methods. For example, humans cannot simulate a large number of users for a performance test in the way a machine could. Large amounts of test data, too, are more efficiently handled by automation.
- **3. Potential for error**
- This is the flipside to #1 in the "pros" column. Humans are smarter than machines in many ways, but they're also prone to human error. Since manual testing is repetitive and boring, it's possible for testers to lose concentration and miss something.
- **4. Not reusable**
- As the manual testing process can't be recorded, manual tests are not reusable—you'll need to develop separate test cases for each new application. It's much easier to do this in automated testing where the scripts are reusable.

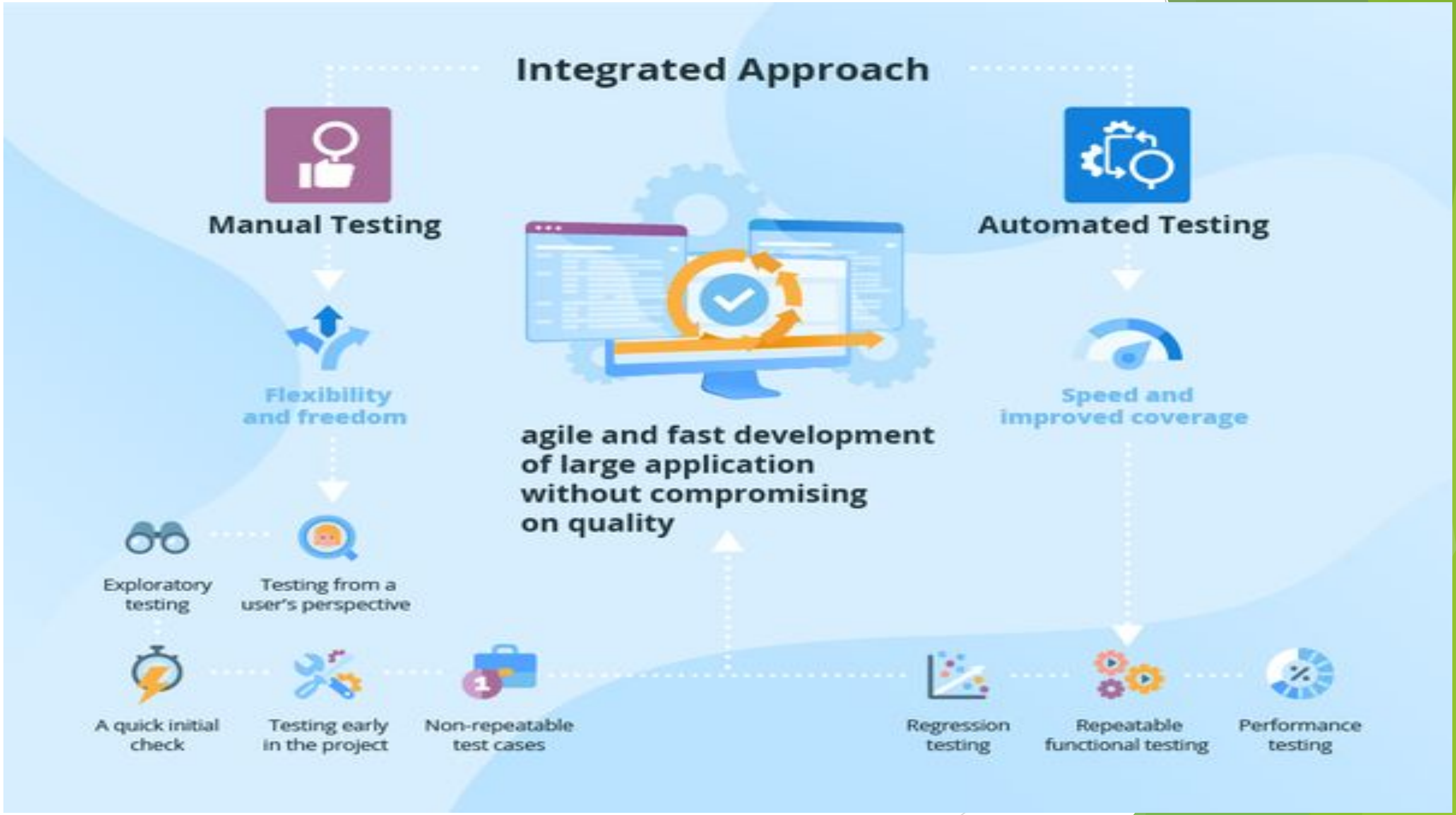
How manual testing is performed

- There are roughly six basic stages in performing manual testing:
- 1. Understand requirements
- The first thing testers need to do is to fully understand the project’s requirements. What does the client expect from the application? What problem is it aiming to solve for end-users? Testers must analyze all requirement documents in order to recognize the expected behavior of the software and exactly what needs to be tested.
- 2. Prepare test cases
- Once the requirements are understood, testers can draft test cases to cover various scenarios, such as what happens when a user enters an invalid password or how the software would cope with a crash. These test plans will set out a sequence for testing functionality and usability across the entire application, measured against expected results.
- 3. Review test cases
- It’s helpful to review the draft test cases with team leaders and the client, to check that they will cover all bases and make any amendments before commencing the execution. This will save time in the long run.

- **4. Execute test cases**
- Manual testing can now be carried out, using any of the techniques listed in the previous section. As well as finding bugs, the aim is to identify potential pain points for users and loopholes that could be exploited by hackers. Testers execute the test cases one by one, sometimes using bug-tracking tools like Jira.
- **5. Report bugs**
- When bugs are identified, the testing team will pass the metrics to the development team in the form of a test report. This contains details on how many defects or bugs were found, how many test cases failed, and which need to be re-run.
- **6. Test again**
- Once the development team has fixed the bugs, the software is handed back to the testers. They carry out the test cases again to check that the problem is resolved.

- Manual testing has its own drawbacks, however, a few of which can include:
  - It’s extremely time-consuming
  - There’s a high risk of error
  - It requires the presence of a tester 24/7
  - Requires manual creation of logs
  - Has a limited scope


Error prone
Limited scope for testing
Time consuming
Why can't I test for the performance??



agile and fast development of large application without compromising on quality

- Although automation is a time-saver, manual testing remains a vital part of software development. Human testers can think like an end-user, and imagine multiple test scenarios for an application or function.
- It’s worth remembering that while software testing attempts to find as many bugs as possible, identifying all possible defects is literally impossible. Manual testers can often spot issues that a machine could overlook, but they are also susceptible to human error.
- Using a combination of manual and automated testing is the most effective way to catch the highest number of bugs and defects.

Automated testing : Selenium

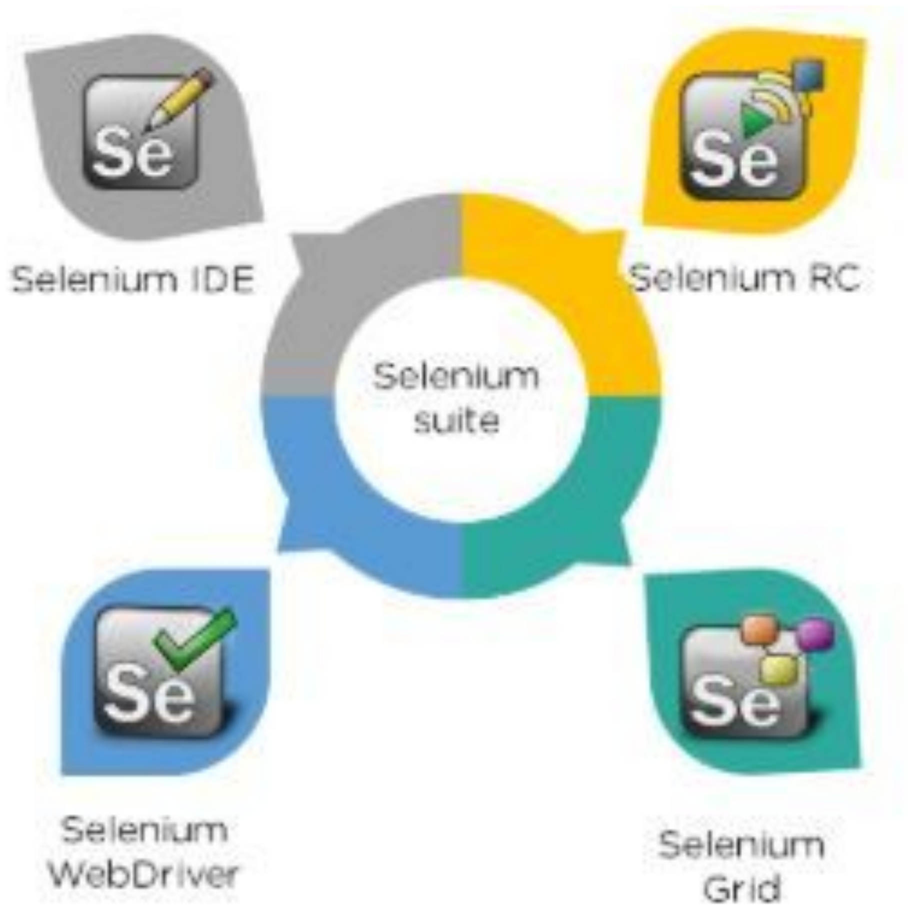
- Selenium is an open-source, automated testing tool used to test web applications across various browsers.
- Selenium can test web applications against various browsers like Firefox, Chrome, Opera, and Safari, and these tests can be coded in several programming languages like Java, Python, Perl, PHP, and Ruby.
- It is platform-independent, meaning it can deploy on Windows, Linux, and Macintosh, and can be integrated with tools like JUnit and TestNG for test management.

What makes Selenium Such a Widely Used Testing Tool?

- Selenium is easy to use since it’s primarily developed in JavaScript
- Selenium can test web applications against various browsers like Firefox, Chrome, Opera, and Safari
- Tests can be coded in several programming languages like Java, Python, Perl, PHP, and Ruby
- Selenium is platform-independent, meaning it can deploy on Windows, Linux, and Macintosh
- Selenium can be integrated with tools like JUnit and TestNG for test management

Selenium Suite of Tools

- Selenium has a dedicated suit applications.



Selenium Integrated Development Environment (IDE)

- Developed by Shinya Kasatani in 2006, Selenium IDE is a browser extension for Firefox or Chrome that automates functionality. Typically, IDE records user interactions on the browser and exports them as a reusable script.

- IDE was developed to speed up the creation of automation scripts. It’s a rapid prototyping tool and can be used by engineers with no programming knowledge whatsoever.

- IDE ceased to exist in August 2017 when Firefox upgraded to a new Firefox 55 version, which no longer supported Selenium IDE. Applitools rewrote the old Selenium IDE and released a new version in 2019. The latest version came with several advancements.

- Selenium IDE has a few shortcomings:

- It does not support data-driven testing
- It can’t perform database testing
- It can’t provide a detailed test report
- It can’t export to WebDriver scripts

Advantages of Selenium Testing

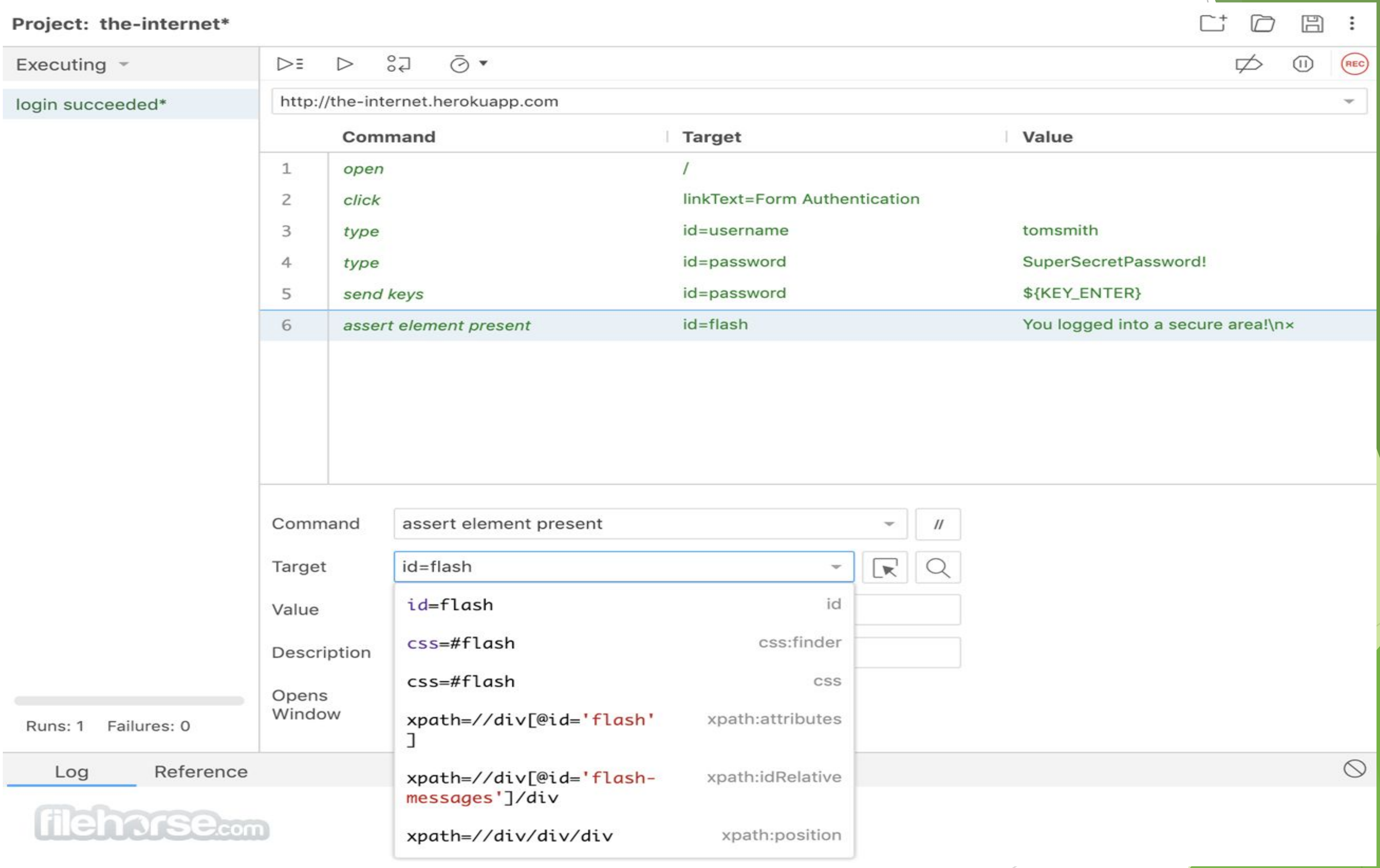
- Selenium automated testing comes with several benefits, such as:

1. Selenium has proven to be accurate with results thus making it extremely reliable
2. Since selenium is open-source, anybody willing to learn testing can begin at no cost
3. Selenium supports a broad spectrum of programming languages like Python, PHP, Perl, and Ruby
4. Selenium supports various browsers like Chrome, Firefox, and Opera, among others
5. Selenium is easy to implement and doesn't require the engineer to have in-depth knowledge of the tool
6. Selenium has plenty of re-usability and add-ons

Limitations of Selenium Testing

- Selenium has a few shortcomings, which can include:

1. Since Selenium is open-source, it doesn't have a developer community and hence doesn't have a reliable tech support
2. Selenium cannot test mobile or desktop applications
3. Selenium offers limited support for image testing
4. Selenium has limited support for test management. Selenium is often integrated with tools like JUnit and TestNG for this purpose
5. You may need knowledge of programming languages to use Selenium



Project: the-internet*

Executing

login succeeded*

http://the-internet.herokuapp.com

	Command	Target	Value
1	open	/	
2	click	linkText=Form Authentication	
3	type	id=username	tomsmith
4	type	id=password	SuperSecretPassword!
5	send keys	id=password	${KEY_ENTER}
6	assert element present	id=flash	You logged into a secure area!\n×

Command: assert element present

Target: id=flash

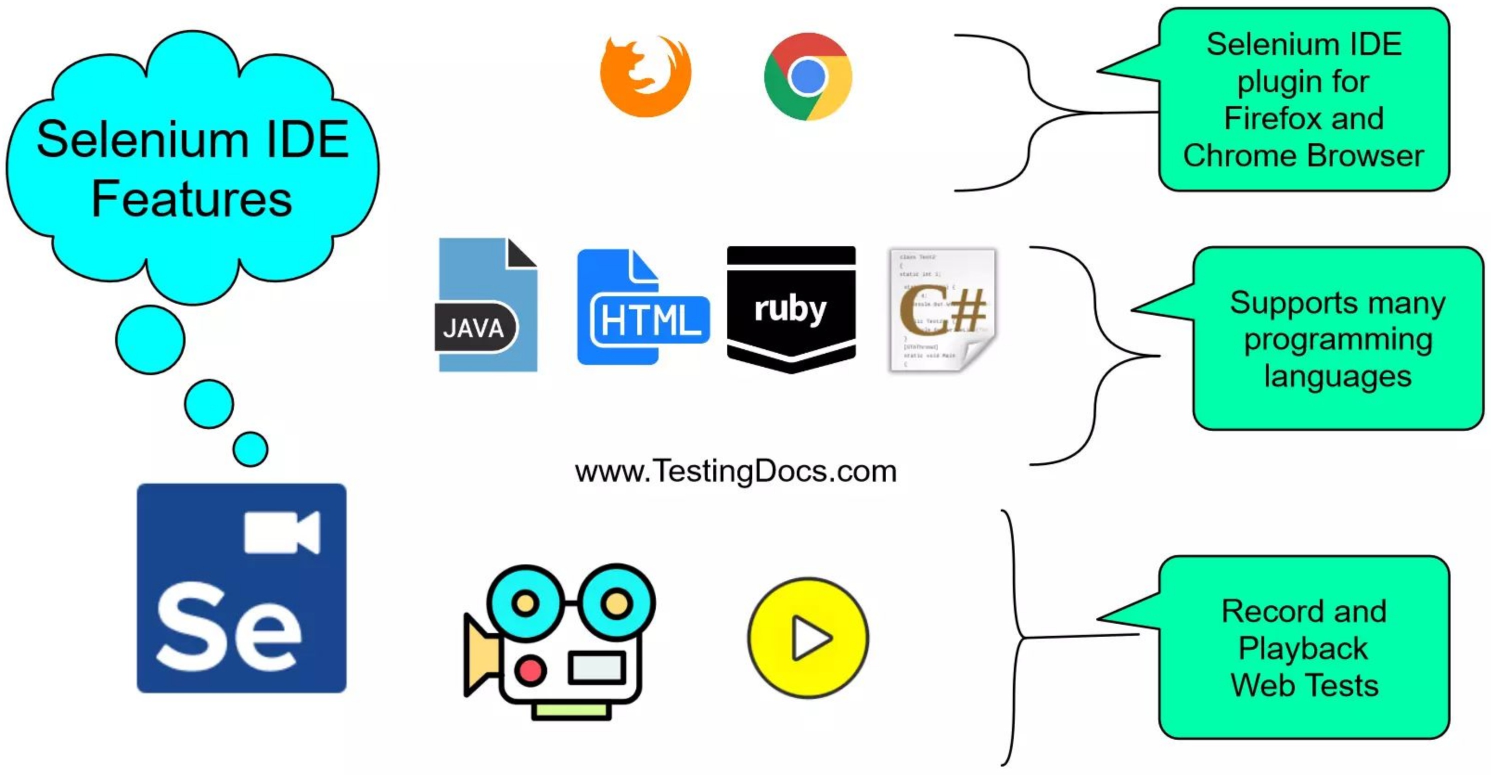
Value

Description

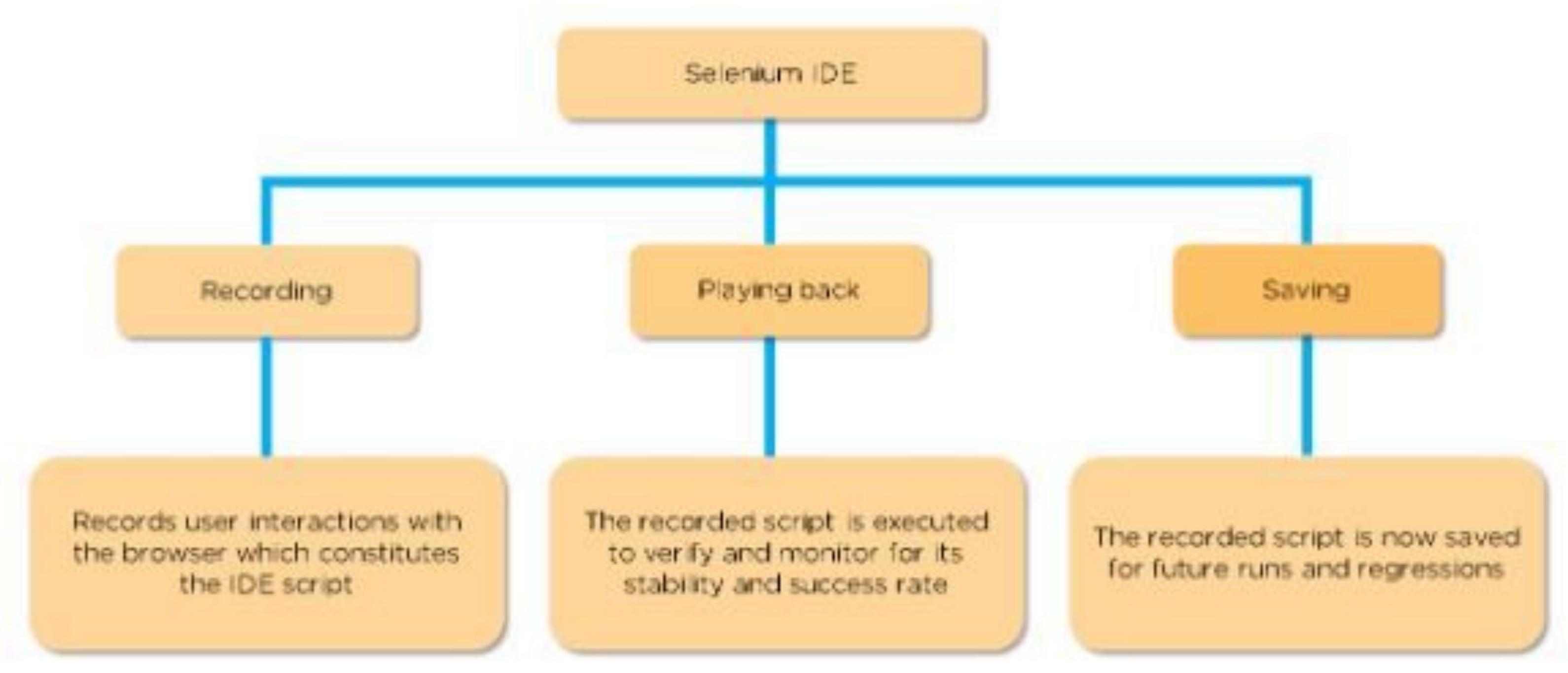
Opens Window

id=flash	id
css=#flash	css:finder
css=#flash	css
xpath=//div[@id='flash']	xpath:attributes
xpath=//div[@id='flash-messages']/div	xpath:idRelative
xpath=//div/div/div	xpath:position

Runs: 1 Failures: 0


Selenium IDE Features
Selenium IDE plugin for Firefox and Chrome Browser
JAVA
HTML
ruby
C#
Supports many programming languages
www.TestingDocs.com
Se
Record and Playback Web Tests

Working Principle of Selenium IDE


Selenium IDE
Recording
Playing back
Saving
Records user interactions with the browser which constitutes the IDE script
The recorded script is executed to verify and monitor for its stability and success rate
The recorded script is now saved for future runs and regressions

- Recording
- IDE allows the user to record all of the actions performed in the browser. These recorded actions as a whole are the test script.

- Playing Back
- The recorded script can now be executed to ensure that the browser is working accordingly. Now, the user can monitor the stability and success rate.

- Saving
- The recorded script is saved with a “.side” extension for future runs and regressions.

Resources:

- https://www.techtarget.com/whatis/definition/software-testing