<https://www.geeksforgeeks.org/how-to-debug-database-in-android/>

<https://www.geeksforgeeks.org/how-does-android-app-work/>

<https://www.javatpoint.com/white-box-testing-vs-black-box-testing>

<https://www.geeksforgeeks.org/automation-tools-for-testing-android-applications/>

What is debug app in Android?

Debugging **allows you to go through each line of code, evaluating your app's variables, methods and how well your code is working**. It's easier to find small mistake in large pieces of code. In this article we will go through basic tips and tricks on debugging an Android app

**Start debugging**

1. Set breakpoints in your app's code.
2. In the toolbar, select a device to debug your app on from the target device menu. ...
3. In the toolbar, click Debug . ...
4. If the Debug window isn't open, select View > Tool Windows > Debug, or click Debug in the tool window bar.
5. Click the Debugger tab,

**How to Debug Database in Android?**

Using Android Debug Database Library in your application  
  
That's all, just start the application, you will see in the logcat an entry like follows : **D/DebugDB: Open http://XXX.XXX.X.XXX:8080 in your browser**. You can also always get the debug address url from your code by calling the method DebugDB. getAddressLog();

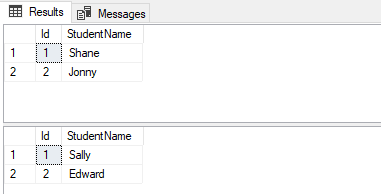
The Android Debug Database library is a useful tool for troubleshooting databases and shared preferences in Android apps. we would be looking forward to using this library and get our hand on it, so continue reading, and indulge. First thing’s first,

### What’s Exactly an Android Debug Database?

Well, the answer to that is simple, In a very easy way, Android Debug Database **allows you to examine databases and shared settings straight in your browser.**

Not only this, but the ADD also enables you to look at the structure of your database in a debuggable format, so that you could take action on your data, in a much more granular and sophisticated manner.

*All these features work seamlessly, even****without***[***rooting your Android Device***](https://www.geeksforgeeks.org/what-is-android-rooting/)*! Isn’t that great?*



In case you missed the address log in [logcat](https://www.geeksforgeeks.org/logcat-window-in-android-studio/)to get the address with [Toast](https://www.geeksforgeeks.org/android-what-is-toast-and-how-to-use-it-with-examples/). Because this library is auto-initialized, if you wish to retrieve the address log, add the following function and call it using reflection (we have to do this to avoid build errors in release builds because this library will not be included). Just simply pop out a toast by using this:

**public static void showingDebug(Context context) {**

**if (BuildConfig.DEBUG) {**

**try {**

**Class<?> debugDB = Class.forName("com.GeeksforGeeks.DebugDB");**

**Method getAddressLog = debugDB.getMethod("getAddressLog");**

**Object value = getAddressLog.invoke(null);**

**Toast.makeText(context, (String) value, Toast.LENGTH\_LONG).show();**

**} catch (Exception ignore) {**

**// A simple catch statement!**

**}**

**}**

**}**



public static void applyCustomFileDBHere(Context c)

{

if (BuildConfig.DEBUG) {

try {

Class<?> GeeksforGeeksDB = Class.forName("com.GeeksforGeeks.GeeksforGeeksDB");

Class[] argTypes = new Class[]{HashMap.class};

Method setCustomDatabaseFiles = GeeksforGeeksDB.getMethod("setCustomDatabaseFiles", argTypes);

HashMap<String, Pair<File, String>> customDatabaseFiles = new HashMap<>();

// set your custom database files

customDatabaseFiles.put(GfGDBHelper.DATABASE\_NAME,

new Pair<>(new File(c.getFilesDir() + "/" + GfGDBHelper.DIR\_NAME +

"/" + GfGDBHelper.DATABASE\_NAME), ""));

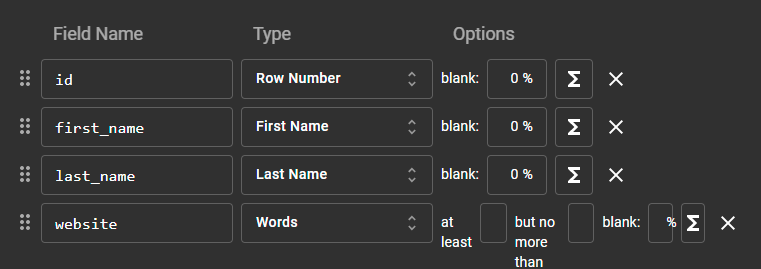
setCustomDatabaseFiles.invoke(null, customDatabaseFiles);

} catch (Exception ignore) {

}

}

}



# How Does Android App Work?

# Developing an [android](https://www.geeksforgeeks.org/kotlin-android-tutorial/) application involves several processes that happen in a sequential manner. After writing the source code files, when developers click the **Run button** on the [Android studio](https://www.geeksforgeeks.org/guide-to-install-and-set-up-android-studio/), plenty of operations and process starts at the backend. Every operation happening in the background is a crucial step and are interdependent. The IDE builds all the application files, make them device compatible in order to deploy, and assure that the application runs successfully on the device. This article broadly explains each and every critical step involved in the journey of an android app, from the IDE files to a working device application.

## Step 1: Building the APK File

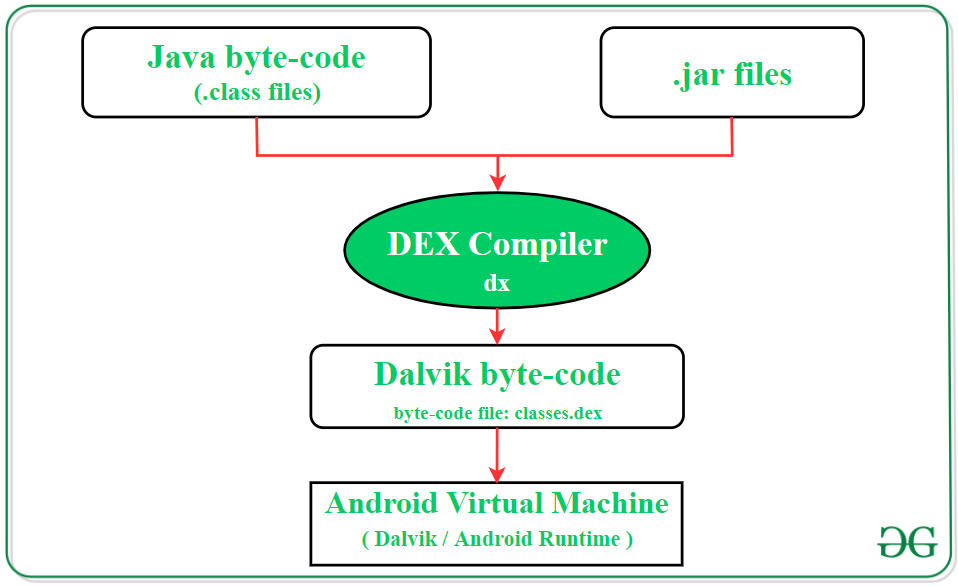
### 1. Code Compilation

The android application source files are written in either [**Java**](https://www.geeksforgeeks.org/java/)**(\*.java files)** or **[Kotlin](https://www.geeksforgeeks.org/kotlin-programming-language/)(\*.kt files)**programming languages. Syntax of writing the code in these 2 languages are different but their compilation process is almost the same. Both programming languages generate code that can be compiled to **Java byte-code** which is executable on [JVM(Java Virtual Machine)](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/). In an android environment, the process begins with the compilation of Java/Kotlin source code into the [**Java class file**](https://www.geeksforgeeks.org/java-class-file/). The class files have the extension **\*.class** and it contains the java byte-code(represents Java assembly). This compilation task is carried out by the **javac** and **kotlinc** compilers for the Java and Kotlin language code respectively.

# Lightbox

### 2. Conversion into Dalvik bytecodes

The generated Java class(\*.class) file in the previous step is executable on Java Virtual Machine(JVM) as it contains the standard **Oracle JVM Java byte-codes**. However, this code format is not suitable for Android devices and thus Android has its own unique byte-code format known as **Dalvik byte-code**. **Dex compiler** translates the Java byte-codeinto the Dalvik byte-code that are machine-code instructions for a theoretical processor. During the compilation process, **dx command**ties up all the **.class files** as well as **.jar files** together and creates a single **classes.dex file**thatis written in Dalvik byte-code format. This file is now executable on the virtual machine in the Android operating system known as **Android Runtime**(or **[Dalvik Virtual Machine(DVM)](https://www.geeksforgeeks.org/what-is-dvmdalvik-virtual-machine/)**for android version older than Kitkat(4.4)).



***Java regular code:***

*public int addTwoNumbers(int a, int b)*

*{*

*return a+ b;*

*}*

***Equivalent Java byte-code:***

*public int addTwoNumbers(int, int);*

*Code:*

*0: iload\_1*

*1: iload\_2*

*2: iadd*

*3: ireturn*

***Equivalent Dalvik byte-code:***

*.method public addTwoNumbers(II)I*

*.registers 4*

*.param p1, “a”    # I*

*.param p2, “b”    # I*

*.line 6*

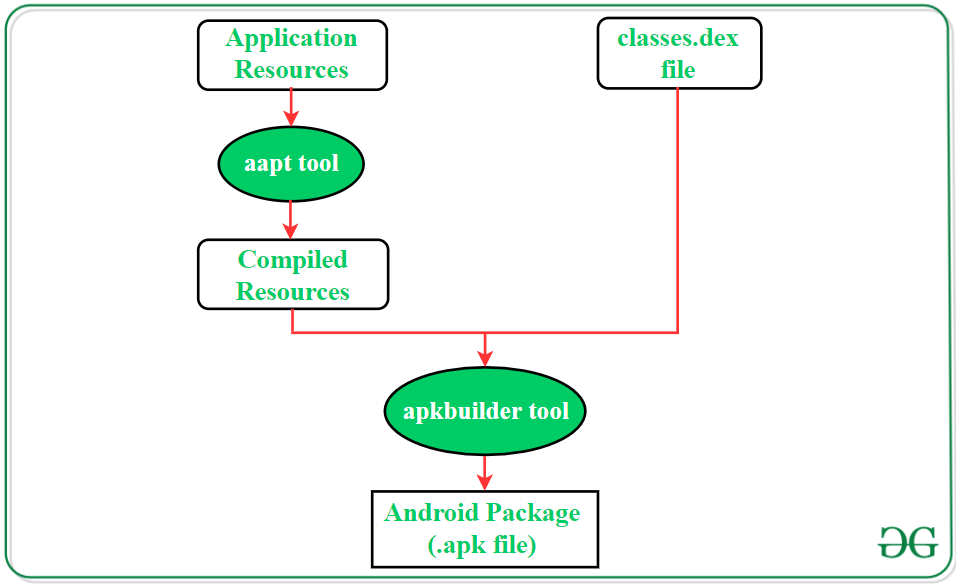
*add-int v0, p1, p2*

*return v0*

*.end method*

### 3. Generating .apk file

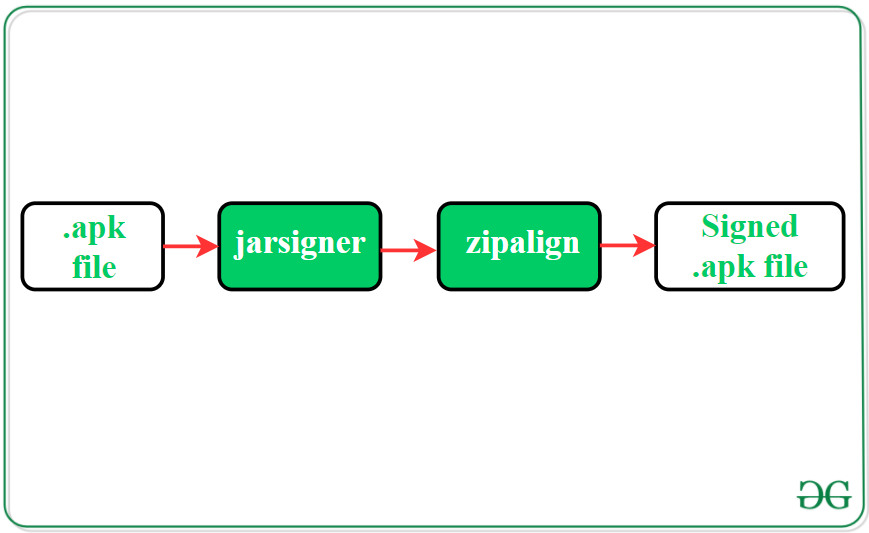
[Resource files](https://www.geeksforgeeks.org/android-res-values-folder/) of the android application like images, fonts, XML layouts, etc. are transformed into a single compiled resource unit by the **Android Asset Packaging Tool(aapt)**. The aapt tool is also responsible for creating the R.java file of an android application. Further, the compiled resource unit along with the **classes.dex** file is compressed by the **apkbuilder tool** and a zip-like file is created that is termed as **Android Package(.apk file)**. The generated .apk file contains all necessary data to run the Android application.



### 4. App Distribution

The .apk file generated in the previous step is a ready-to-use application package and developers can use this file for the purpose of app distribution. However, to distribute and [publish the application through **Google Play Store**](https://www.geeksforgeeks.org/how-to-publish-your-android-app-on-google-play-store/), developers need to sign it. Android applications are required to be digitally signed with a certificate so that they can be installed by the users. The certificate is self-signed, and the Android uses it to identify the author of the application. The app developer/author holds the private key of the certificate and these all details are stored as an additional file in the android package(.apk file).

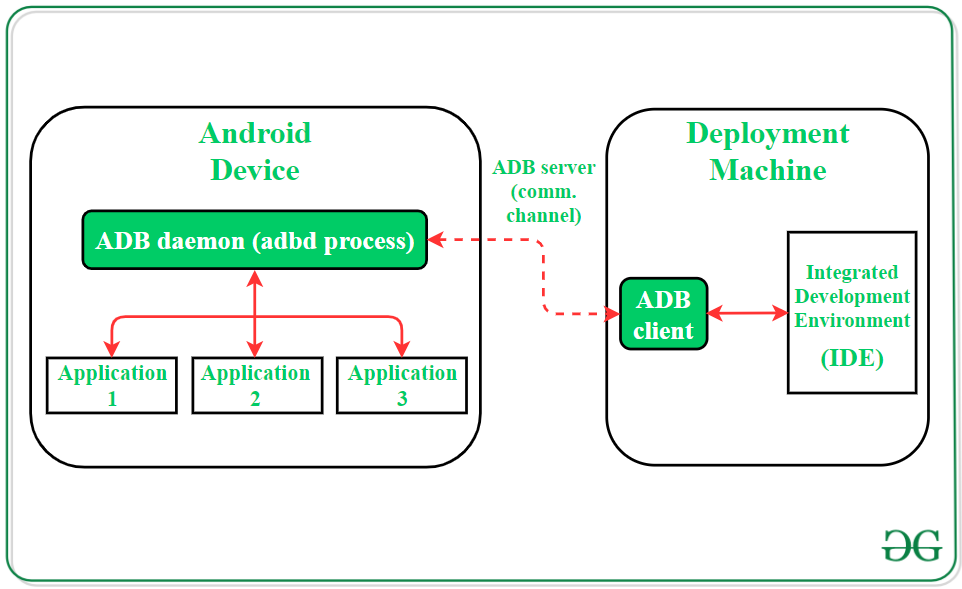
Oracle Java Development Kit(JDK) provides the **jarsigner tool**to sign the .jar files and .apk files. Further, the compressed parts of the [signed .apk file](https://www.geeksforgeeks.org/how-to-generate-signed-apk-in-android-studio/) are required to line up on byte-boundaries in such a manner so that Android OS can read them without uncompressing the file. The byte alignment of files is assured by running the signed .apk file through the **zipalign tool**.



## Step 2: Deploy the Application

### 1. Establish the ADB Server

Android Debug Bridge(ADB) deploys an application to Android devices. It is a command-line tool that acts as an interface and facilitates developers to communicate with an android device. To start the deployment, the **ADB client** will first check whether the **ADB server process** is already running on the device. If there isn’t, the server process starts with the **ADB command**. The ADB server starts and binds with local **TCP port 5037**. All communication and commands are transferred from the ADB server to ADB clients using port 5037. Further, the server sets up a connection with all running devices. It scans all the ports and when the server detects an **ADB daemon**(adbd: a background process on emulator or device instance), it set up a connection to that port. The adbd process that matches on the android device can communicate with applications, debug them, and collect their log output.



### 2. Transfer .apk file to the Device

The ADB command transfers the .apk file into the local file system of the target Android device. The app location in the device file system is defined by its package name. For example, if the application package is *com.example.sampleapp*, then its .apk file will be located in the path */data/app/com.example.sampleapp*.

## Step 3: Run the Application

### 1. App launch request

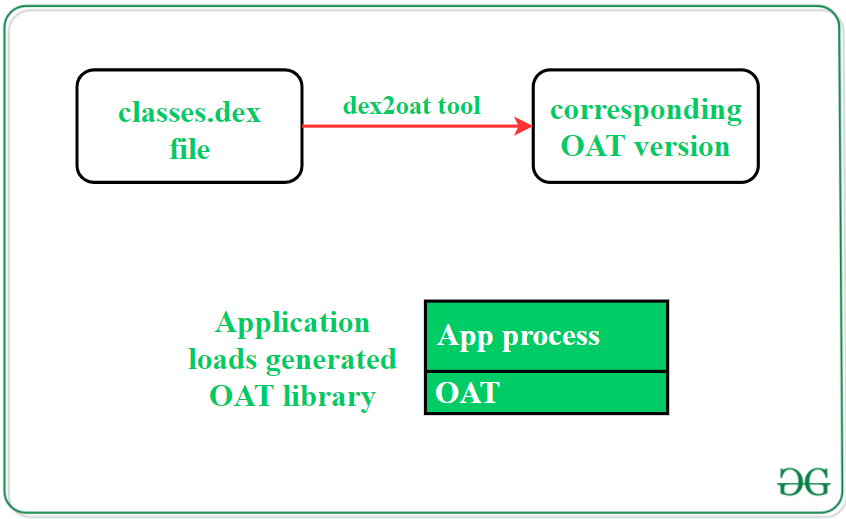
**Zygote process** is the parent to all Android apps and it launches an application when a user makes the request to do so. The zygote is a special kind of Android OS process which enables code sharing between different instances that run across Android virtual devices(Dalvik/Android Runtime). Those resources, classes, and code libraries which possibly required by any application at runtime are preloaded in the memory space of the zygote process. Whenever the process gets a request to launch a new application, it forks itself(creates a copy) using the fork system call(android is a Linux system) and starts the new app. The preloaded libraries and resources are the reason for efficient and fast app launch in android.

### 2. Conversion of the .dex code to native OAT format

When a new application is installed, the Android optimize the app data and generates a corresponding OAT file. This file is created by the Android OS to accelerate the application loading time. The process to generate the OAT file starts with the extraction of **classes.dex** file present inside the **.apk** file of the application. The classes.dex file is placed in a separate directory and Android compiles the Dalvik byte-code with **ahead-of-time(AOT, also**abbreviated**as OAT)** compilation into native machine code. Android system uses this native OAT file to enhance the user experience by loading the application quickly and smoothly.

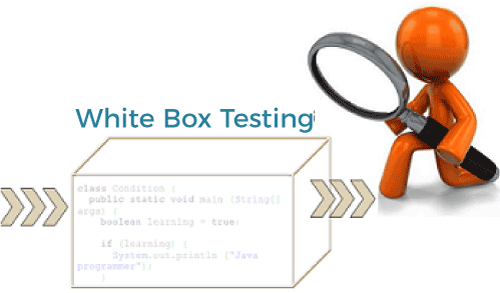
Before AOT came into the picture, **dexopt tool**is used to convert the .dex files into .odex file(optimized DEX) that holds the optimized byte-code. With the introduction of AOT in Android, **dex2oat tool**converts and optimize the .dex file into an OAT file format that holds machine code written in **ELF format(Executable and Linkable Format).** This native library is then mapped into the memory of the application process. OAT files are generally saved in the Android device in the directory:

*/data/dalvik-cache/*



## White Box testing

The term 'white box' is used because of the internal perspective of the system. The **clear box or white box, or transparent box** name denotes the ability to see through the software's outer shell into its inner workings.



It is performed by Developers, and then the software will be sent to the testing team, where they perform black-box testing. The main objective of white-box testing is to test the application's infrastructure. It is done at lower levels, as it includes unit testing and integration testing. It requires programming knowledge, as it majorly focuses on code structure, paths, conditions, and branches of a program or software. The primary goal of white-box testing is to focus on the flow of inputs and outputs through the software and strengthening the security of the software.

It is also known as structural testing, clear box testing, code-based testing, and transparent testing. It is well suitable and recommended for algorithm testing.

To read more about white box testing, you can refer to the following link – [White box testing](https://www.javatpoint.com/white-box-testing).

## Black Box testing

The primary source of black-box testing is a specification of requirements that are stated by the customer. It is another type of manual testing. It is a software testing technique that examines the functionality of the software without knowing its internal structure or coding. It does not require programming knowledge of the software. All test cases are designed by considering the input and output of a particular function. In this testing, the test engineer analyzes the software against requirements, identifies the defects or bugs, and sends it back to the development team.

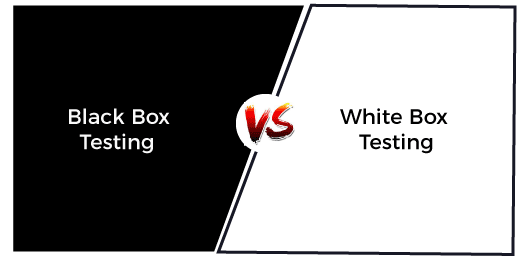


In this method, the tester selects a function and gives input value to examine its functionality, and checks whether the function is giving the expected output or not. If the function produces the correct output, then it is passed in testing, otherwise failed.

Black box testing is less exhaustive than White Box and Grey Box testing methods. It is the least time-consuming process among all the testing processes. The main objective of implementing black box testing is to specify the business needs or the customer's requirements.

In other words, we can say that black box testing is a process of checking the functionality of an application as per the customer's requirement. Mainly, there are three types of black-box testing**: functional testing, Non-Functional testing,**and **Regression testing**. Its main objective is to specify the business needs or the customer's requirements.

## White box testing v/s Black box testing



|  |  |  |  |
| --- | --- | --- | --- |
| **S.no.** | **On the basis of** | **Black Box testing** | **White Box testing** |
| **1.** | **Basic** | It is a software testing technique that examines the functionality of software without knowing its internal structure or coding. | In white-box testing, the internal structure of the software is known to the tester. |
| **2.** | **Also known as** | Black Box Testing is also known as functional testing, data-driven testing, and closed-box testing. | It is also known as structural testing, clear box testing, code-based testing, and transparent testing. |
| **3.** | **Programming knowledge** | In black-box testing, there is less programming knowledge is required. | In white-box testing, there is a requirement of programming knowledge. |
| **4.** | **Algorithm testing** | It is not well suitable for algorithm testing. | It is well suitable and recommended for algorithm testing. |
| **5.** | **Usage** | It is done at higher levels of testing that are system testing and acceptance testing. | It is done at lower levels of testing that are unit testing and integration testing. |
| **6.** | **Automation** | It is hard to automate black-box testing due to the dependency of testers and programmers on each other. | It is easy to automate the white box testing. |
| **7.** | **Tested by** | It is mainly performed by the software testers. | It is mainly performed by developers. |
| **8.** | **Time-consuming** | It is less time-consuming. In Black box testing, time consumption depends upon the availability of the functional specifications. | It is more time-consuming. It takes a long time to design test cases due to lengthy code. |
| **9.** | **Base of testing** | The base of this testing is external expectations. | The base of this testing is coding which is responsible for internal working. |
| **10.** | **Exhaustive** | It is less exhaustive than White Box testing. | It is more exhaustive than Black Box testing. |
| **11.** | **Implementation knowledge** | In black-box testing, there is no implementation knowledge is required. | In white-box testing, there is a requirement of implementation knowledge. |
| **12.** | **Aim** | The main objective of implementing black box testing is to specify the business needs or the customer's requirements. | Its main objective is to check the code quality. |
| **13.** | **Defect detection** | In black-box testing, defects are identified once the code is ready. | Whereas, in white box testing, there is a possibility of early detection of defects. |
| **14.** | **Testing method** | It can be performed by trial and error technique. | It can test data domain and data boundaries in a better way. |
| **15.** | **Types** | Mainly, there are three types of black-box testing**: functional testing, Non-Functional testing,**and **Regression testing**. | The types of white box testing are – **Path testing, Loop testing,**and **Condition testing**. |
| **16.** | **Errors** | It does not find the errors related to the code. | In white-box testing, there is the detection of hidden errors. It also helps to optimize the code. |

## Conclusion

So, both white box testing and black box testing are required for the successful delivery of software. But 100% testing is not possible with both cases. Tester is majorly responsible for finding the maximum defects to improve the application's efficiency. Both black box testing and white box testing are done to certify that an application is working as expected.

Hence, it is necessary to understand both testing techniques. It will also be helpful to learn the difference between both terms to effectively pick up the right option.

# Automation Tools for Testing Android Applications

Testing mobile apps is a critical component of the software development cycle. However, it takes quite some time and resources and can be a challenge when there is a need to meet tight deadlines. Also, most users want fast services and may not have the patience to wait for a developer to sort out defects in a piece of software when they can easily go to a competitor.   
For this reason, software companies need to have efficient, fast and reliable testing tools and practices for all their apps. While manual testing and tools have some successes, they may be slow and sometimes ineffective when there are many apps to test. As such, developers need to prioritize automation.   
Adding automation is, however, not just picking any tool from the saturated market. Some tools are better than others. To pick the best, there needs to be an understating of the app, what to test, what can be automated, and many other issues.

### Best Android App Testing Automation Tools

Usually, the [mobile app testing](https://www.indiumsoftware.com/mobile-apps-testing-services/) automation tools are available in both free and paid versions. Software companies with limited budgets or resources can consider using open source tools. Although free, the tools have almost all the features to automate most of the testing tasks. However, they may have a few limitations which may not have a big impact on the testing processes.

To help you narrow down on the right tool, here are the best mobile automation tools for Android apps.

### 1. [Appium](http://appium.io/" \t "_blank)

This is an open source automation tool that supports a wide range of Android operating system versions starting from 2.3 upwards. It has a UI Automator and support for a wide range of programming languages. For example, it works on apps written in [Java](https://www.geeksforgeeks.org/java/), [Ruby](https://www.geeksforgeeks.org/ruby-programming-language/), [PHP](https://www.geeksforgeeks.org/php/), Node, [Python](https://www.geeksforgeeks.org/python-programming-language/), [C#](https://www.geeksforgeeks.org/csharp-programming-language/) and others as well as the ones in the WebDriver library.   
The Appium is a cross-platform automation tool and has the ability to easily test multiple devices in parallel. In addition, there is great support and advice from the large open-source community. This tool is available from GitHub and is easy to install and low maintenance tool. Since it has a standard robotization APIs, it does not require recompiling the application.   
The tool has a few limitations such as lack of detailed reports and being a bit slow as the tests rely on a remote web driver.

### 2. [Robotium](https://en.wikipedia.org/wiki/Robotium" \t "_blank)

This is an open-source automation tool that supports native and hybrid mobile apps running Android versions from 1.6 upwards. It provides an easy and quick way to write test cases and scenarios without requiring high levels of coding. Other benefits include support for several languages. Also, it is capable of automatically running multiple Android routines. The tool is ideal for small software companies with limited budgets.

The main benefits include:

* Robust tests
* Supports testing native and hybrid mobile apps
* Faster testing speeds
* Ability to perform simultaneous tests
* Simulate user behaviors such as clicking, touching, typing, etc.

Although powerful, it lacks the Record and Play functions and does not have screenshots. Also, it can take long before getting the tests to run.

### 3. [Kobiton](https://kobiton.com/" \t "_blank)

The Kobiton automation tool has both free and paid versions. Either of them gives the testers the ability to check the real devices. It also supports automation of actions that manual testing would require. This includes orientation, speaker and camera control, GPS simulations, multi-touch gestures, connection management and more.   
The Kobiton tool will also capture all the actions by the tester hence making it easier and quicker to identify and resolve issues.

Benefits include: 

* Ability to test using real, cloud-based mobile devices and a variety of configurations
* Centralized logs that make it easier to check the testing logs and history
* Easy to install, simple user interface
* Free trial for limited time
* Access to over 350 real devices on the cloud
* Parallel tests
* Detailed results, logs, screen captures video, as well as system metrics such as the battery, and memory performance.

Although it is free for the basic features, users can upgrade to other versions with additional functionalities at a monthly fee of between *$50 and $300*.

### 4. [Squish For Android](https://www.froglogic.com/squish/)

Squish for Android is a premium automation tool that supports native, web and hybrid mobile apps. The automation tool runs on both real and simulated devices and does not require rooting the device under test. The easy to use tool allows testers to get started with the automated GUI quickly and easily.

The automation tool: 

* Provides support for the automation of both standard and complex gestures such as touch, tap, swipe, multi-touch, etc.
* Tests devices in the cloud
* Provides IoT testing
* End-to-end testing
* Testing web and HTML components embedded in the app

This is a paid automation tool but has a trial period so as to allow the companies to test and see if it can deliver.

### 5. KMAX

KMAX is one of the best automation tools for testing how mobile apps behave under different network conditions. In particular, it establishes the effect of poor as well as normal connectivity.   
The tool provides testers with several pre-defined network scenarios based on common connectivity solutions. This includes 3G, 4G, LTE, and others, and allows the testers to emulate a wide range of network conditions. For example, the tool enables the teams to establish the effect of different levels of packet loss, delays, or corruption as well as other adverse conditions such as bandwidth issues, jitter and more. In addition, testers can use the KMAX tool to recreate certain scenarios and especially those that had issues.

Other benefits include 

* Over 22 predefined network scenarios
* Customizable, controlled and repeatable tests
* Use of real or simulated mobile network traffic
* The premium tool with flexible payment plans based on requirements

### 6. [Experitest](https://experitest.com/" \t "_blank)

The Experitest is a powerful automation platform that allows software companies to create and execute a wide range of automated tests in parallel. It has the ability to run tests on over 1000 real Android devices in the cloud at the same time. The tool provides real-time interaction, visual test reports, and analytics. In addition, it provides the ability to debug the apps in real-time.

The tool enables the software developers to accelerate the testing processes on native, web and hybrid mobile apps. It gives them the ability to detect bugs easily and quickly on a wide range of apps   
Experitest is a paid tool but has a trial period.

### 7. Moneytalk

Moneytalk is an open-source tool that supports automation of a wide range of basic to complex data-driven tests. It automates interactive functional tests for the native, web and hybrid Android apps through simple and understandable test scripts which it can also record and playback.   
The tool, which does not require high levels of scripting or programming skills, supports emulators, tethered and network connected devices. It also generates [HTML](https://www.geeksforgeeks.org/html-tutorials/)and XML reports as well as screenshots at failure points.   
Considerations when selecting a mobile testing automation tool   
Since there are so many automation tools, it is usually difficult choosing the one that will meet the project’s testing needs may be a challenge. Another problem that there is no one tool that provides a comprehensive solution and you may need to make a trade-off or combine several tools.

To get reliable results at the fair cost and time, it is important to

* Understand the app under test
* Determine what requires testing
* Determine all the requirements to perform the testing including hardware, software-network, and other resources
* Research for automation tools that meets the testing requirements
* If possible, try two or three of the best tools to see what works best for your budget.
* Choose the best tool that meets your requirements and budget

When there is a limited budget for automation tools, a software development company should consider open source products. Most of the best free tools have the ability to automate a wide range of testing processes. Unless looking for advanced features and support, the small development companies should consider the free options otherwise it may not be justified to purchase expensive tools to test only a few apps.

### Conclusion

Automating mobile app testing process enables development companies to save on costs and time, eliminate human errors and produce better quality software. However, its success requires an understanding of the application, proper research, planning and selecting the right tool.   
Usually, there is a wide range of automation tool for testing Android apps. These range from free open source products to premium automation tools. While the expensive automation tools provide advanced features, they may be out of reach for small companies. For those with limited or no budgets for automation, they have a choice of using open source tools which provide a good level of automation.   
Whether paid or free, the software companies need to pay attention to the capabilities and limitations of each tool and only choose what fits their requirements and budget.