

In the beginning, there was Make as the only build tool available. Later, it was improved with GNU Make.

JVM ecosystem is dominated with three build tools:

1. Ant with Ivy
2. Maven
3. Gradle

Ant

Ant was the first build tool released in 2000 and was easy to learn. Build scripts format was XML.

Drawback of ANT:

1. XML being hierarchical in nature, is not a good fit for procedural programming approach.
2. XML tends to *become unmanageably big* when used with all but very small projects

Maven

Maven was released in 2004. It has improved few of the problem of ANT.

Maven continues using XML as the format to write build script, however, structure is diametrically different.

The most important addition was the ability to download dependencies over the network(later on adopted by **Ant through Ivy**)

Drawback of Maven:

1. Dependencies management does not handle conflicts well between different versions of the same library.
2. Customization of targets (goals) is hard.

Gradle

Gradle was released in 2012. Google adopted Gradle as the default build tool for the Android OS.

Gradle does not use XML. Instead, it had its own DSL based on [Groovy](#) (one of JVM languages).

As a result, Gradle build scripts tend to be much shorter and clearer than those written for Ant or Maven.

Initially, Gradle used [Apache Ivy](#) for its dependency management. Later, it moved to its own native dependency resolution engine.

Build script samples:

```
<ivy-module version="2.0">
  <info organisation="org.apache" module="java-build-tools"/>
  <dependencies>
    <dependency org="junit" name="junit" rev="4.11"/>
    <dependency org="org.hamcrest" name="hamcrest-all" rev="1.3"/>
  </dependencies>
</ivy-module>
```

Ant Build Script Sample

```

<project xmlns:ivy="antlib:org.apache.ivy.ant" name="java-build-tool"

  <property name="src.dir" value="src"/>
  <property name="build.dir" value="build"/>
  <property name="classes.dir" value="${build.dir}/classes"/>
  <property name="jar.dir" value="${build.dir}/jar"/>
  <property name="lib.dir" value="lib" />
  <path id="lib.path.id">
    <fileset dir="${lib.dir}" />
  </path>

  <target name="resolve">
    <ivy:retrieve />
  </target>

  <target name="clean">
    <delete dir="${build.dir}"/>
  </target>

  <target name="compile" depends="resolve">
    <mkdir dir="${classes.dir}"/>
    <javac srcdir="${src.dir}" destdir="${classes.dir}" classpat
  </target>

  <target name="jar" depends="compile">
    <mkdir dir="${jar.dir}"/>
    <jar destfile="${jar.dir}/${ant.project.name}.jar" basedir="
  </target>

</project>

```

Maven Build Script Sample

```

apply plugin: 'java'
apply plugin: 'checkstyle'
apply plugin: 'findbugs'
apply plugin: 'pmd'

version = '1.0'

repositories {
    mavenCentral()
}

dependencies {
    testCompile group: 'junit', name: 'junit', version: '4.11'
    testCompile group: 'org.hamcrest', name: 'hamcrest-all', version
}

```

Gradle Build Script

Maven,

is better for managing dependencies (but Ant is ok with them too, if you use **Ant+Ivy**) and build **artifacts**.
The main benefit from maven - its lifecycle.

You can just add specific actions on correct phase, which seems pretty logical:

- just launch your integration tests on integration-test phase for example. Also, there are many existing plugins, which can could almost everything.

Maven **archetype** is powerful feature, which allows you to quickly create project.

Ant,

is better for controlling of build process. Before your very first build you have to write you build.xml.

If your build process is very specific, you have to create complicated scripts.

For long-term projects support of ant-scripts could become really painful:

- Scripts become too complicated, people, who's written them, could leave project, etc.

Both of them use xml, which could become too big in big long-term projects.