JAVA BUILD TOOLS: PART 2

A DECISION MAKER'S COMPARISON OF MAVEN, GRADLE AND ANT + IVY

Who will win "Least Annoying Build Tool"?

REBELLABS

TABLE OF CONTENTS

↑

Click to go to section

CHAPTER I: INTRODUCTION

“The evolution from Make, to Ant, and then to Maven has done precious little to advance the state of Java build tools. Developers are still stuck with poorly thought-out tools that force us to violate DRY and write XML tag soup. Your team may be better served using a less popular alternative.”

- *Jess Johnson in October 2010,* http://grokcode.com/538/

Oooh, more on Java Build Tools. Hooray :-/

Believe it or not, Java developers don't consider build tools to be the most interesting topic out there. They are generally not considered the most exciting segment of *any* developer's overall utility belt.

After all, the majority of the dev world still chooses between just two build tools, Maven and Ant, the older of the two having been created nearly a generation ago. At best, programmers would prefer their build tool remain invisible and stable; at worst, we hear complaints of downloading enormous libraries, scripts failing for no reason due to some invisible rule running in the background, and general annoyances.

However, build tools *should* still be able to rock, and it's in the spirit of "Build Tools [Can] Rock!" that RebelLabs has set out to finalize our journey into the realm of Java's three most popular build tools--Maven, Gradle and Ant (along with Ivy for managing dependencies). After all, if anyone was going to try to put it a little "sexy" back into Ant, it would be us ;-)

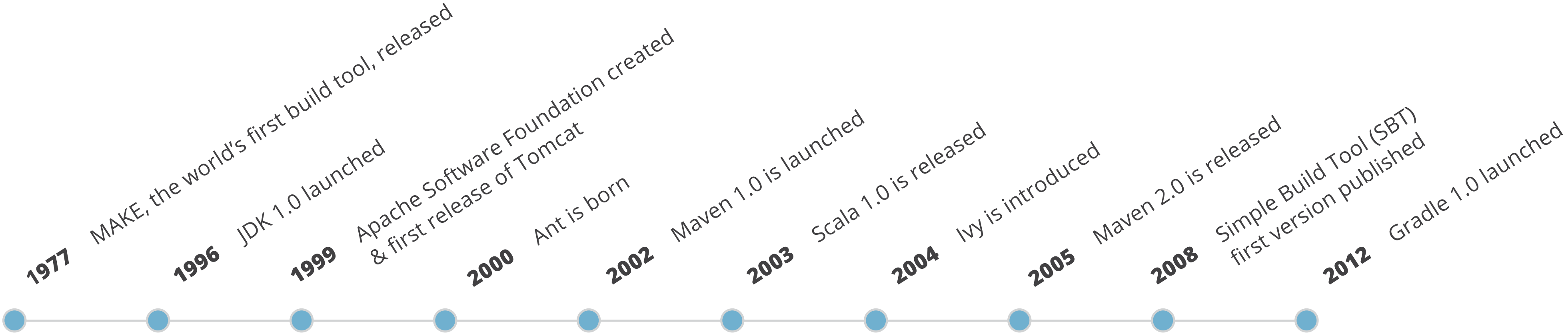
This all started in December 2013, when we published Java Build Tools Part 1. It would help to think of Parts 1 and 2 as a single publication, broken into two sections due to length and time restrictions. In the first part, we showed developers how to get started with all of the tools mentioned, we reviewed some tips and pointers on creating the build script, interacting with communities and how to use/create plugins for your overall development environment.

In Part 2, we go deeper and get out the proverbial red pen, taking each build tool and ranking them in six categories, then applying those scores to four common user profiles (i.e. use cases) in order to figure out which build tool makes the most sense for you. But now, let's review where the general Java development industry stands in terms of current and historical build tool usage.

What we've been using over time

THE EVOLUTION OF BUILD TOOLS: 1977 - 2013 (AND BEYOND)

Visual timeline


1977 MAKE, the world's first build tool, released
1996 JDK 1.0 launched
1999 Apache Software Foundation created & first release of Tomcat
2000 Ant is born
2002 Maven 1.0 is launched
2003 Scala 1.0 is released
2004 Ivy is introduced
2005 Maven 2.0 is released
2008 Simple Build Tool (SBT) first version published
2012 Gradle 1.0 launched

Here is a timeline going from 1977 to 2013, showing how things have emerged over time. You'll notice that MAKE and SBT are mentioned, but aren't appearing in the report due to lack of higher representation in the world. But they are worth looking into as well, just FYI.

Considering the timeline here, it's probably better to look at Build Tools in use since 2010, and we've seen the following trends over the last few years. Here are the self-reported statistics from three years worth of developer data.

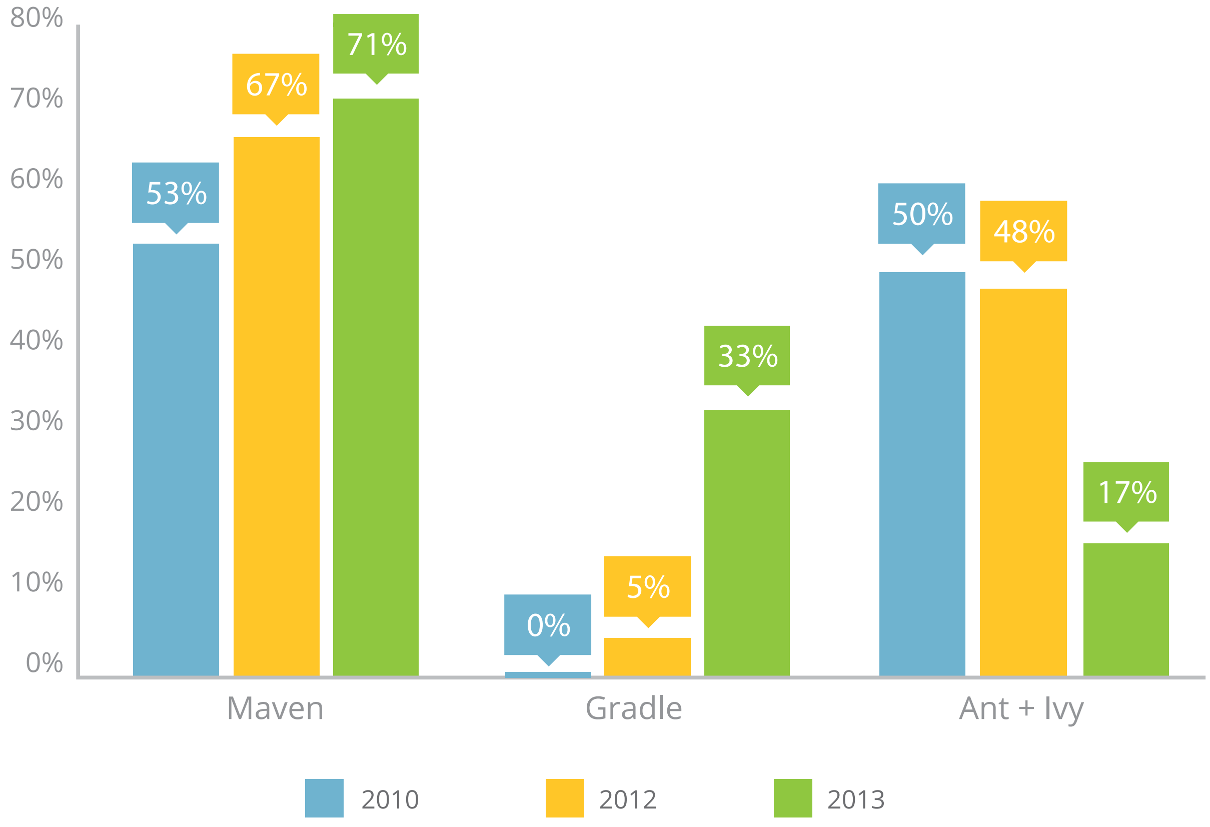
From what we can see, Maven is increasing at a steady rate, Ant is losing market share and Gradle is increasing at a more explosive pace. It would not be incorrect to suggest that people are migrating from Ant to Gradle, but it's equally likely that Ant users go first to Maven (which is also XML-based) and then proceed to Gradle.

In any case, which tool developers use is not always up to them. However, by ranking these tools against six categories for four different user profiles, we hope to shed some light on matters. In Chapter II, here is what we test and compare:

- How easy is the initial **learning curve**?
- How **fast** are different builds with each tool?
- How **complex** is it to create and maintain the build script?
- How many **plugins** exist and how simple is it to customize your own plugins?
- How good is the **community and documentation** for each tool?
- How well does each tool integrate with **developer tools**? (IDE, App Server, CI server)

From there, we take those scores and apply them across four user profiles--from hobbyist developer and OSS project creator to a developer inside of medium-sized organization and developer #12645 in a global megapower.

Build Tools Popularity - Late 2010 to Mid 2013



REBELLABS

Source: ZeroTurnaround

CHAPTER II: RANKING EACH BUILD TOOL BASED ON 6 CATEGORIES

In this chapter, we look at six areas in which an ideal build tool should really excel--from getting started and then maintaining your build process for the long term, plugins, community and documentation quality, we put each tool to the test and see how they do...

How easy is the initial learning curve?

When learning a new technology or tool, it's always nice to get up and running really quickly making use of features that are both intuitive and easy to use.

So, we'll look at how much advance knowledge is needed, how easy it is to script something up from scratch, debugging, minor customizations and problem solving. Two things: 1) the quality of the community and documentation (which is another section) plays a role here and 2) we are not considering the impact of long-term project maintainability in this section.

MAVEN

Learning Maven is not a hard thing to do and you don't need to be familiar with Maven to be able to build and package the artifact or run unit tests on some existing project. Maven's **pom.xml** file comes in to play here, and it's harder creating a new file from scratch than just adding changes to an existing one--this is where your IDE comes in, as Eclipse, IntelliJ IDEA and NetBeans will all generate some minimal working **pom.xml**, where you will have to change artifact's attributes and introduce some dependencies if you feel like it.

Of course the bigger the project becomes the bigger is its **pom.xml** file. Multi-module projects have several build files and suddenly you see that it becomes hard to trace all the links and dependencies among the **pom.xml** files. Refactoring them is a separate task: moving the configuration to parent or child *pom*, resolving version constraints and so on, these all can really drive you mad, but this isn't a problem you should encounter too often.

Custom actions during the build is another challenge. As we mentioned in the previous report, Maven does not let you describe build steps right in **pom.xml** as the build script is used only to turn on or off some plugins/ functionalities. So in order to add some custom logic to your build process, you actually have to use the **Antrun plugin** and write your logic in Ant's **build.xml** or create your own plugin!

Nonetheless, the learning curve for Maven isn't terrible and after a couple hours of reading in parallel with writing your build script, you'll do enough trial and error to get started.

Score: ●●●○○

Comments:
Creating a pom from scratch is difficult, but IDEs can do that for you.
The inability to easily customize your build process (i.e. using Ant is needed) isn't great, but getting started isn't terribly difficult using the right docs.

GRADLE

The learning curve for Gradle is partially affected by another JVM language--Groovy. With just a little understanding of Groovy, it is really easy to get it working and understand it thoroughly, especially if you have a little background with Maven as well (as most devs do). But in reality, installing Gradle and getting it to work is also quite easy even without experience with these tools at all.

So how it is possible? Gradle actually has some things that make it easily adoptable and understandable to wide audience. For example, it's based on Groovy, which is something like a simplified version of Java that is easily understandable for Java developers. Gradle also uses Maven's (or Ivy's) repository format, so that managing dependencies in Gradle does not require any extra learning time from you if you have experience with either of these tools.

The most time consuming part and what makes the learning curve a little bit more gentle is the Gradle DSL. It's a totally new thing, introduced by Gradle itself and tightly connected to Gradle internals. But on the other side, Gradle DSL makes writing Gradle build scripts quite enjoyable because of lot of open possibilities.

Score: ●●●●○

Comments:
Some experience with Maven or Ant + Ivy is useful. You have to learn a bit of Groovy, but that's just syntax, as with anything. Gradle DSL is simple, intuitive and incredibly flexible--you can even start from an empty script and build from scratch.

ANT + IVY

Starting to learn Ant and Ivy isn't the hardest task in the world, but previous experience with build tools, dependency management, and XML will go a long way. Getting a simple Ant script up and running doesn't take long; there are plenty of templates that can be found around the web, and a lot of the IDEs out there can also auto-generate a basic Ant script for you.

Unfortunately, Ivy doesn't have the same widespread IDE support as Ant, but templates do exist online. Of course, you can also follow the simple steps as described in the previous report to get started.

The hard part about Ant is often to maintain and expand your build script as your project grows in size and complexity. While Ant is pretty logically structured, it can be tricky at times to figure out which tasks and attributes to use, and how to use it! Most IDEs have full tag-completion for Ant scripts, but of course that doesn't help much if looking for a task called **compile**, you really should be looking for a task called **javac** instead.

Overall, for getting something simple up and running with both Ant and Ivy, the learning curve is fairly small. Yes, you have to get familiar with XML, and the basics tags, but once that's done, you also have the template for future scripts.

Score: ●●●○○

Comments:
Logical and simple to get started using good old XML, but understanding references to files can be painful, having to learn both Ant and Ivy syntax and maintain two files isn't awesome.

How fast are different builds with each tool?

When looking into whether there are any real substantial differences in speed across the build tools, we should consider a couple things. There are two big factors which mean that comparing build speed is somewhat hard to do... First of all a build script is only as good as the person who writes it, which will ultimately depend on their skillz within that build tool as well as their skillz in migrating that script to other build tools.

Secondly, it's hard if not impossible to find a reasonably sized project out there which doesn't already have a build environment tailored to a particular tool. For example, the majority will have a Maven build implementation and will make use of the years of expertise and performance fixes it's owners have graced it with (and of course all the fluff that people add and are later too scared to delete), so simply migrating this to another build tool might not give us the best results.

With this in mind, we selected a smaller project to test with so that it would reduce the error margins caused by the human factors discussed. We decided to use the Spring's Pet Clinic scripts that we created in the first part of this report series. (**Note**: this is currently maintained as a Maven build environment, so we just needed to create the Ant + Ivy and Gradle scripts)

Tests were all run on the same MacBook Air laptop. Spec: 8GB RAM (1600 MHz DDR3), 2GHz Intel Core i7 CPU. The HDD is a 512 GB SSD, OSX 10.8.2. Each test was run five times with both the highest and lowest test runs removed. The remaining three runs were averaged to get the results we'll show now (Omitted test runs show up in grey).

DOING A CLEAN BUILD WITHOUT TESTS

	maven	gradle	APACHE ANT ivy
Command	time mvn -Dmaven.test.skip=true clean package	time gradle clean build -x test --daemon	time ant clean war
Time - Run 1 *(seconds)*	6.458	3.302	7.414
Time - Run 2 *(seconds)*	6.112	3.215	7.466
Time - Run 3 *(seconds)*	7.042	3.29	7.361
Time - Run 4 *(seconds)*	6.266	3.622	7.359
Time - Run 5 *(seconds)*	5.218	3.433	7.222
Average (min /max omitted)	**6.279**	**3.342**	**7.378**

We're going to start by performing a clean build, then compiling our code and packaging up our war file with each build tool. Here are our results.

We can see there's not a huge amount of difference between all our build scripts here, and nobody is going to miss a second and a half, so there's little to really put between each of the build tools so far, but Maven takes the win by technical KO. Gradle suffers here from a longer initialization time which can affect it on the smaller builds. Let's move on to a more common activity now, by running an incremental build. So we omit the clean and make some code changes.

INCREMENTAL BUILD WITHOUT TESTS

	maven	gradle	<APACHE ANT> ivy
Command	time mvn -Dmaven.test.skip=true package	time gradle build -x test --daemon	time ant war
Time - Run 1 *(seconds)*	5.405	3.1	4.758
Time - Run 2 *(seconds)*	5.625	3.628	4.808
Time - Run 3 *(seconds)*	5.399	4.185	4.856
Time - Run 4 *(seconds)*	5.966	3.094	4.822
Time - Run 5 *(seconds)*	5.625	4.259	4.904
Average (min /max omitted)	**5.552**	**3.638**	**4.829**

This time Maven seems to be fairly static on it's timings compared to the previous run, but Gradle and Ant seem to have jumped ahead. We're still looking at some very close times so again there's nothing to split the pack here. Next we're going to run tests as part of our build and see how the build tools fare. The first set of results below are with a clean, compile, package and test. The second set of results omits the clean stage.

DOING A CLEAN BUILD WITH TESTS

	maven	gradle	APACHE ANT ivy
Command	time mvn clean package	time gradle clean build --daemon	time ant clean war test
Time - Run 1 *(seconds)*	13.506	11.369	13.641
Time - Run 2 *(seconds)*	13.093	14.86	13.457
Time - Run 3 *(seconds)*	12.543	10.898	13.437
Time - Run 4 *(seconds)*	13.755	13.013	13.354
Time - Run 5 *(seconds)*	14.527	15.146	13.593
Average (min /max omitted)	**13.451**	**13.081**	**13.496**

INCREMENTAL BUILD WITH TESTS

	maven	gradle	APACHE ANT ivy
Command	time mvn package	time gradle build --daemon	time ant war test
Time - Run 1 *(seconds)*	13.846	9.386	11.762
Time - Run 2 *(seconds)*	12.998	10.648	10.61
Time - Run 3 *(seconds)*	13.646	9.811	10.795
Time - Run 4 *(seconds)*	13.426	9.976	10.689
Time - Run 5 *(seconds)*	12.785	10.302	10.632
Average (min /max omitted)	**13.357**	**10.030**	**10.705**

Ant now looks like it’s stretching out a lead now, particularly over Gradle. Maven is pretty close behind. Again, there is very little between the Maven times with and without the clean, but it seems to affect Ant and Gradle a lot more, which is surprising given what it’s doing. We can work out the differences between our 4 sets of results now to work out the cost of the test phase. This is simply by subtracting the first set of results in our clean build table from the second set of results which run the test. We can do the same with the incremental build results to see how much time is spent just doing test. Should be the same right? After all, we’re running the same tests! WRONG!

THE COST OF TEST

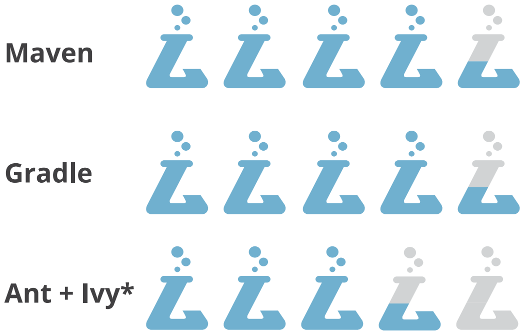
	maven	gradle	<APACHE ANT> ivy
Test time for clean build *(seconds)*	7.173	9.739	6.118
Test time for incremental build *(seconds)*	7.805	6.392	5.877
Average	**7.489**	**8.066**	**5.997**

This time Maven seems to be fairly static on it's timings compared to the previous run, but Gradle and Ant seem to have jumped ahead. We're still looking at some very close times so again there's nothing to split the pack here. Next we're going to run tests as part of our build and see how the build tools fare. The first set of results below are with a clean, compile, package and test. The second set of results omits the clean stage.

DOING A CLEAN BUILD WITH TESTS (DOWNLOAD DEPENDENCIES)

	maven	gradle	<APACHE ANT> ivy
Command	rm -rf ~/.m2/repository && time mvn clean package	rm -rf ~/.m2/repository && rm -rf ~/.gradle/caches/ && time gradle clean build --daemon	rm -rf ~/.ivy2/cache/ && time ant clean war test
Time - Run 1 *(seconds)*	41.393	35.412	136
Time - Run 2 *(seconds)*	37.418	33.402	133
Time - Run 3 *(seconds)*	36.797	30.548	137
Time - Run 4 *(seconds)*	42.656	30.336	141
Time - Run 5 *(seconds)*	39.637	35.369	129
Average (min /max omitted)	**39.483**	**33.106**	**135.333**

Holy dependencies Batman! At last, something we can really go to town on! Ivy really struggles with it's dependency downloads, in terms of speed. Maven and Gradle download dependencies one at a time, when they locate them. Ivy, however, has a two phase approach. It will try to find all it's dependencies first and then it will start to download them all. This process takes a long time, but once all the dependencies have been pulled down, Ant takes over and zips through the build and test as quick we've seen above. It's a real shame it is exposed so much during the dependency download phase.



**Average taken of Ant (5) and Ivy (2)*

Comments: Maven and Gradle are close enough to Ant for it's build, and test differences not to matter so much. Ivy loses points here for it's shocking lack of speed on dependency downloads. Nobody gets 5 here, as there's always room for improvement with build speed!

How complex is it to create and maintain the build script?

There are many aspects that can add to the complexity of a build script, particularly from the point of view of another person in your team, new to the script you're writing. When looking into this area, we also need to look long-term as well--for example, how easy is it for a new dev to read/ understand your script a year after you write it? Is your build script complex or verbose, or both? (yikes) How intuitive is the build script flow--is it easy to follow? Is your build script explicitly or implicitly ordered?

MAVEN

When reading Maven build scripts, there are some conventions that you should be aware of--this is sometimes known as "The Maven Way". Mostly, this refers to the **super pom**, which you can see here and contains some relatively non-obvious (i.e. invisible) inheritance and aggregation rules, giving Maven an implicit order of lifecycles. Thankfully, you don't have to know all of these rules by heart, and the XML in Maven build scripts is quite self describable. Really, you delve into the details only if you really need to.

What makes Maven much easier, is that all Maven build scripts are practically the same. Once you learn the pattern, all **pom.xml** files will be familiar to you. There are very few possibilities to add some custom logic in it, which is good in the sense that consistency and conformity are good--however, it makes Maven inflexible as well.

Score: ● ◐ ○ ○ ○

Comments:
Not as verbose as Ant, but complexities exist in the hidden conventions. Complexity rises a lot when dealing with multi-module projects. Implicit order of lifecycles makes things less flexible.

GRADLE

The readability of Gradle build scripts is quite simple. It depends who wrote the script and how, but in general, while taking a look at these scripts, it can roughly seen what is the script doing, even without much Groovy experience. The script is declarative and imperative at the same time, meaning that it is just possible to declare things (like source sets) and also write custom logic at the same time. Here is a build script example: https://gist.github.com/anonymous/8397963

Like Maven, Gradle tries to also perform operations implicitly without any need to describe, how to do stuff and when. But at the same time, it is possible to write and override everything:

```
sourceSets {
  test {
    resources {
      srcDir 'src/test/java'
    }
  }
}
```

As Gradle works with Groovy, it is also possible to use all Groovy's possibilities here, like closures:

```
repositories {
    println "in a closure"
       println project.buildDir
}
```

Score: ●●●●◐

Comments:
Easy to read and maintain, Gradle scripts are short with little boilerplate. However using convention over configuration for lifecycle order could lead to confusion, but at least you can change them, unlike Maven.

ANT + IVY

The complexity of Ant build scripts varies a lot, depending on what you're trying to accomplish. Especially when dealing with many many targets, the dependency graph between them can easily be unclear, but still enjoy the benefit of the dependencies being explicitly stated in the script!

The fact that Ant scripts use XML gives it a certain level of boilerplate code that cannot be avoided, thus the minimum (readable) script that is able to compile anything at all clocks in at around 20 lines. Very complex scripts though, can easily run into the thousands of lines. For instance, the harness used by NetBeans hovers close to the 2000 lines. A useful feature to counteract this is the flexibility to split an Ant script into multiple files and then use import-statements to bind them back together. This also makes it easier to reuse parts of a script for future scripts. Likewise, properties can also be loaded from an external file, thus further reducing the length of your main build script.

A basic Ivy file should get you far, though the complexity rises when you start making customizations; like adding multiple configurations and dependencies between them. Getting these mappings to work as intended can be a bit of a trial-and-error task the first few times, trying various different combinations and then it finally works, and looking at the mapping string afterwards, you have no idea why.

Score: ●●●○○

Comments:
Lots of verbose, boilerplate script, but you get used to ignoring it (at least nothing is hidden), targets that depend on other targets can create a very confusing flow. Users can be explicit with Ant and make customizations, although adding Ivy to the mix increases complexity.

How many plugins exist and how simple is it to customize your own plugins?

In the first report, we listed some of our favorite plugins for each build tool, but when you get down to it, this is really a two-part comparison. Firstly how many plugins exist out there for you to plagiarise^H^H^H^H^H^H^H^H^H^H reuse? It's important you're not just reinventing the wheel for every project, so what have others already done and how easy are they to find and integrate? When a plugin you need doesn't exist, how easy is it to create one from scratch?

MAVEN

Sometimes Maven is called "plugin execution framework", because if you want to do something that Maven currently can't do, you'll need to find a plugin or write one yourself. There is simply no other alternative. You are not allowed to write any code in the build script, like with Gradle for example. This makes the amount and quality of plugins a cornerstone element of the build tool, since you can't really do anything without them!

There are 2 main sites to look for plugins: http://maven.apache.org/plugins/ and http://mojo.codehaus.org/plugins.html. There are easily 200+ plugins and there are more to look for on Google Code. Maven has been here for a long time and thus has a lot of plugins written for different kinds of tasks. But if you haven't succeeded in finding one you need, it is really simple to write your own. What you have to do is to create a Maven project and follow this guide.

Score: ● ● ● ● ○

Comments:
Hundreds of plugins exist for Maven so a really good repository in use already, but it's easy to create a new plugin and adding them into your build script is also simple. Can't write code directly in the build script.

GRADLE

Gradle's architecture is actually plugin-based, meaning that all functionality in Gradle is actually a plugin. We can divide Gradle plugins into two categories:

- **Key plugins** - developed by the Gradleware team, these are plugins that are most important for Gradle and probably needed by most users (i.e. plugins for Java, Maven, Eclipse, Project Reports, etc.)
- **Community plugins** - developed by the community and some examples are GradleFX, Google App Engine, JavaScript plugin, Android plugin.

Developing custom plugins with Gradle is made quite easy and well-documented. After that, there are three different options for adding a plugin to Gradle:

- **Inside the build script** - you can write it in place and later copy & paste it to other scripts or just keep it there;
- **buildSrc directory** - plugins can be written as Groovy files and placed to a separate project. Still they can be very close to the project's source, so everyone can find them easily;
- **Separate project -** like with Ant or Maven, plugin can be a different project and maintained by other developers.

Another cool thing is that plugins can be written in any JVM language, but the easiest and most preferable way is to write them is in Groovy. However, since Gradle is the relative newcomer (released in 2012), there is a lack of many established plugins around to choose from.

Score: ●●●○○

Comments:
Excellent flexibility and ease of writing your own plugins (both as a new project to share across teams as well as inside your build scripts), however the library of existing plugins is meager--perhaps since it's so easy to write your own plugins, users might do that rather than creating and hosting plugins in the community.

ANT + IVY

As also mentioned in the previous report, Ant has been around for a while and has so far amassed quite a few plugins in its lifespan; both simple and complex, and the chances are great that what you're looking for already exists out there in some form or another.

To find existing plugins, there are the Apache Ant Libraries, which is basically a sandbox for plugin ideas under the Apache Ant umbrella, with the implementations ranging from the experimental stage to long-since-completed. Also found on the Apache Ant Project site is the External Tools and Tasks section, which contains a wide variety of external plugins for Ant.

If you're looking for Ant integration with a specific application or tool, it would also be advisable to look at that application's vendor's site. For instance, for many application servers official Ant plugins exist to manage and deploy your application to the application server, either bundled with the application server, or downloadable from the vendor's site.

If the above sites don't contain the plugin you're looking for, and Google doesn't hold the answer to your query either, then perhaps it's time to start looking into implementing your idea yourself? To get started writing your own Ant tasks, a simple to follow tutorial can be found in the Ant manual.

Score: ●●●○○

Comments:
Due to the longevity of Ant, there are probably as many plugins as for Maven. Creating your own plugin is also as simple as Maven, but more specifications in the script itself are needed to make it work. Can't write code directly in the build script.

How good is the community and documentation for each tool?

A very important part of any tool's adoption is the documentation, as it tells you how you should go about reaching your goals with a tool. The community is just as important, as it's built up of people who have walked the same path as the one you're on, and will give you practical advice and experience when the docs are wrong, inaccurate or simply do not apply to your environment.

In this section we rank the community, looking at overall voice and activity in forum(s) to determine not only the energy by general size of the community. When it comes to documentation, we're referring to the quality and consistency of official docs, wikis and external resources. We'll be taking the average of the two scores.

MAVEN

Many are not pleased with Maven's documentation; when we have asked some of the developers on the JRebel team at ZeroTurnaround, they told us that the docs were poor, rough and sometimes lacked examples and details. However, the documentation isn't really as bad as all that, and when going into it for this report, it was relatively easy to find the material we needed. Available to developers are some introductory guides on various topics, such as multi-module projects, configurations, extensions etc.

Some other organisations try to come up with their own documentation view, such as Sonatype's Maven reference. And of course you can ask your elder and more experienced colleagues on the topic. But as good as the documentation is, it has taken over for a less-active community.

There are also many good answers on Q&A sites, informative blog posts and thick books on the topic, but they are dated and the conversation doesn't seem to be very ripe these days.

Score: ●●●○○

Comments:
Documentation is relatively good (4), but the community, forum and activity is past its prime--the ghosts of past community interaction still live in the interwebz, perhaps waiting to be resurrected (2).

GRADLE

As mentioned in the previous report, documentation for getting started with Gradle is pure awesomeness. The format of Gradle documentation is somewhat similar to Spring Framework's documentation, where there is a single page HTML and PDF version available and everything is actually illustrated with working code samples. In terms of web design, Gradle docs look fresh and modern, and give you a good feeling when reading them.

Gradle also has a free book available which can be downloaded in PDF or HTML format (you can order the paperback version as well). Regarding online resources and documentation, obviously, everything starts at http://www.gradle.org/. Documentation is available on the same site at http://www.gradle.org/documentation. Basically, Gradle's website leads you to all good resources for learning this tool.

One big difference in the community side of Gradle is that it is organized and effective--since Gradle is developed by Gradleware, which is controlled by the creator of Gradle, you get almost professional-grade community and commercial support. Yet, the tool is open source and everyone can contribute to its development. Unlike the other tools, the Gradle community and the documentation work together very well.

Score: ●●●●●

Comments:
A lot of active users, wikis and excellent documentation--all backed up by Gradleware, an actual company that supports the tool's ecosystem. You can even email someone who will write back to you! Both sections get a 5.

ANT + IVY

Looking through the Ant manual you'll find information about all the individual tasks and attributes, as well information and tutorials to get you started. On the Apache Ant Project site you'll also find a list of external Ant resources, such as books, forum and wikis. The entries in the manual are good; looking at the information for the various tasks usually contains description, parameters, and examples. The overall look of the Ant manual might leave something to be desired (Ant is nearly 15 years old after all!), and a search feature in the online version is at times missed, especially when looking for that specific feature, but can't remember the name of the task. Overall, the information expected are present and delivered in a no-fuss kind of way.

Looking at the online manual of Ivy you get a different impression. While the look and feel of the online manual is more visually appealing than the Ant manual, and you have search available to you, the content feels lacking when veering outside the most commonly-used tags. A good place to start though when using Ivy are the Introduction and Tutorial sections. Here, you'll find descriptions and introductions to the main concepts and terminology used by Ivy, a lot of it rather essential knowledge when working with Ivy.

While sadly neither Ant nor Ivy enjoys the benefits of big and vocal communities, the standard hunting grounds for quick answers like StackOverflow still have knowledgeable people willing to answer your questions.

Score: ●●○○○

Comments:
Many experts who have used it for many years, but the community (1) activity is pretty much dead. Documentation (3) is outdated looking but sufficient.

How well does each tool integrate with developer tools? (IDE, App Server, CI server)

Development environments vary from developer to developer, and can be extremely different, so it's important that build tools support and integrate with the various other tools and products which developers interact with on a daily basis, including IDEs, Application Servers and Continuous Integration Servers. We're going to go through each of these one by one to look at the support which they provide the build tools.

IDEs

IDE support is one of the most crucial integrations for a build tool, and from the table below, we can see how that level of importance translates to support. Luckily, all three major IDEs are well on the path to full feature coverage in all categories, which makes things highly convenient, if not a very interesting table. All tools score top rank for this category, which is to say that none of them do.

Tasks	maven			gradle			<APACHE ANT> ivy		
IDEs	Eclipse	IDEA	NetBeans	Eclipse	IDEA	NetBeans	Eclipse	IDEA	NetBeans
Import a project build structure into IDE	yes	yes	yes	yes	yes	yes	yes	yes	yes
Submit a build from IDE	yes	yes	yes	yes	yes	yes	yes	yes	yes
Dependency management in IDE	yes	yes	yes	yes	yes	yes	yes	yes	yes
Automatic download of dependencies	yes	yes	yes	yes	yes	yes	yes	yes	yes
Support/Wizard to create build scripts	yes	yes	yes	yes	yes	yes	yes	yes	yes
Enablement	plugin	built-in	built-in	plugin	built-in	plugin	Ant: built-in Ivy: plugin	Ant: built-in Ivy: plugin	Ant: built-in Ivy: plugin

APP SERVERS

Many projects produce applications that are deployed into a container at runtime. Build tools offer integration, via plugins, to some/many of the major application servers. This section will dig deeper into the breadth of application server support and the richness of what is offered by each build tool.

With 12-14 years of background, Maven and Ant + Ivy enjoy full support for Tomcat, JBoss WildFly, GlassFish, Jetty, WebSphere (including Liberty Profile) and WebLogic. However, as Gradle is still quite new, it does not have such a wide variety of application server integrations as Maven or Ant have.

Gradle has official plugin for Jetty (but still, most web apps can be tested on it as well).

There is also the plugin for Tomcat, but not developed by the Gradleware team--it's quite well documented and can be found from GitHub: https://github.com/bmuschko/gradle-tomcat-plugin. But as plugin development for Gradle is relatively easy and straightforward, then integrations for other app servers is possible (but oh well, who wants to maintain them?).

APP SERVERS

Tasks	maven						gradle						<APACHE ANT> ivy					
Server >	TC	JB	GF	J	WS	WL	TC	JB	GF	J	WS	WL	TC	JB	GF	J	WS	WL
Start/ Stop Servers	yes	yes	yes	yes	yes	yes	yes	no	no	yes	no	no	yes	yes	yes	yes	yes	yes
Deploy an app	yes	yes	yes	yes	yes	yes	yes	no	no	yes	no	no	yes	yes	yes	yes	yes	yes

TC =Tomcat JB = JBoss GF = GlassFish J = Jetty WS = WebSphere (inc. Liberty Profile) WL = WebLogic

CI SERVERS

Tasks	maven			gradle			<APACHE ANT> ivy		
	Jenkins/ Hudson	Bamboo	Team City	Jenkins/ Hudson	Bamboo	Team City	Jenkins/ Hudson	Bamboo	Team City
Run build scripts	yes	yes	yes	yes	yes	yes	yes	yes	yes
multimodule project detection/incremental builds	yes	yes	yes	yes	yes	yes	yes	no	no
automatic JUnit test results publishing	yes	yes	yes	no*	no*	yes	no*	no*	no*
automatic archiving/ publishing of artifacts	yes	yes	yes	no**	yes	yes	no**	no**	no**
detection of new builds of dependencies	yes	yes	yes	no	no	yes	yes	no	no
Enablement	built-in	plugin	built-in	plugin	plugin	built-in	Ant : built-in Ivy : plugin	plugin	built-in

**CI needs to be told where junit report files are located*

***CI needs to be told where are the newly built artifacts*

CI SERVERS

Anyone who's anyone uses some form of CI server in their build environment, so it's important... wait what? You don't? Are you joking? Anyway, it's really important to achieve really good integration between your build tool and your CI server, whichever combination you choose. Oh, before we go on, when we say **automatic**, we think *User does nothing, tool does everything*...and *not User has to set it up first*.

We tested each tool again three (or four if you like) CI servers that we tested against--Jenkins (Hudson), Bamboo from Atlassian, and TeamCity from JetBrains. Maven has the best out-of-the-box support for tools, and Jenkins even has a special **Maven job**, where it configures running tests and publishing tests' results, artifact archiving and some more properties for you.

Gradle lacks in some categories for CI server tasks, but by using the **Gradle wrapper**, you can use Gradle in any CI environment, and even set it to install Gradle by itself in case it is missing.

Ant is widely supported by CI servers, but since **Ant** can be invoked by a single CLI command, plugin support in the CI isn't necessary to get things rolling, if it can execute commands directly. Getting Ivy, or other Ant plugins for that matter, to work with the CI should also not be an issue.

MAVEN: ●●●●●

Comments: Full support for each tool and every category. There IS a reason that the majority of developers use Maven, after all.

GRADLE: ●●●○○

Comments: Lacking in the App Server and CI Server categories, due mainly to newness.

ANT + IVY: ●●●●○

Comments: Full support for IDEs and App Servers, lacking in CI Server categories but still providing wider support than Gradle.

So, that covers those six categories--now it's time to look at the results! First, we'll take a look at the raw numbers to see which tool came out on top, and then we'll apply the scores to the priorities for each feature section that we suggest for each of the four user profiles. Then--at last--we'll expect to have a winner!

CHAPTER III:
THE RESULTS OF COMPARING MAVEN, GRADLE AND ANT + IVY

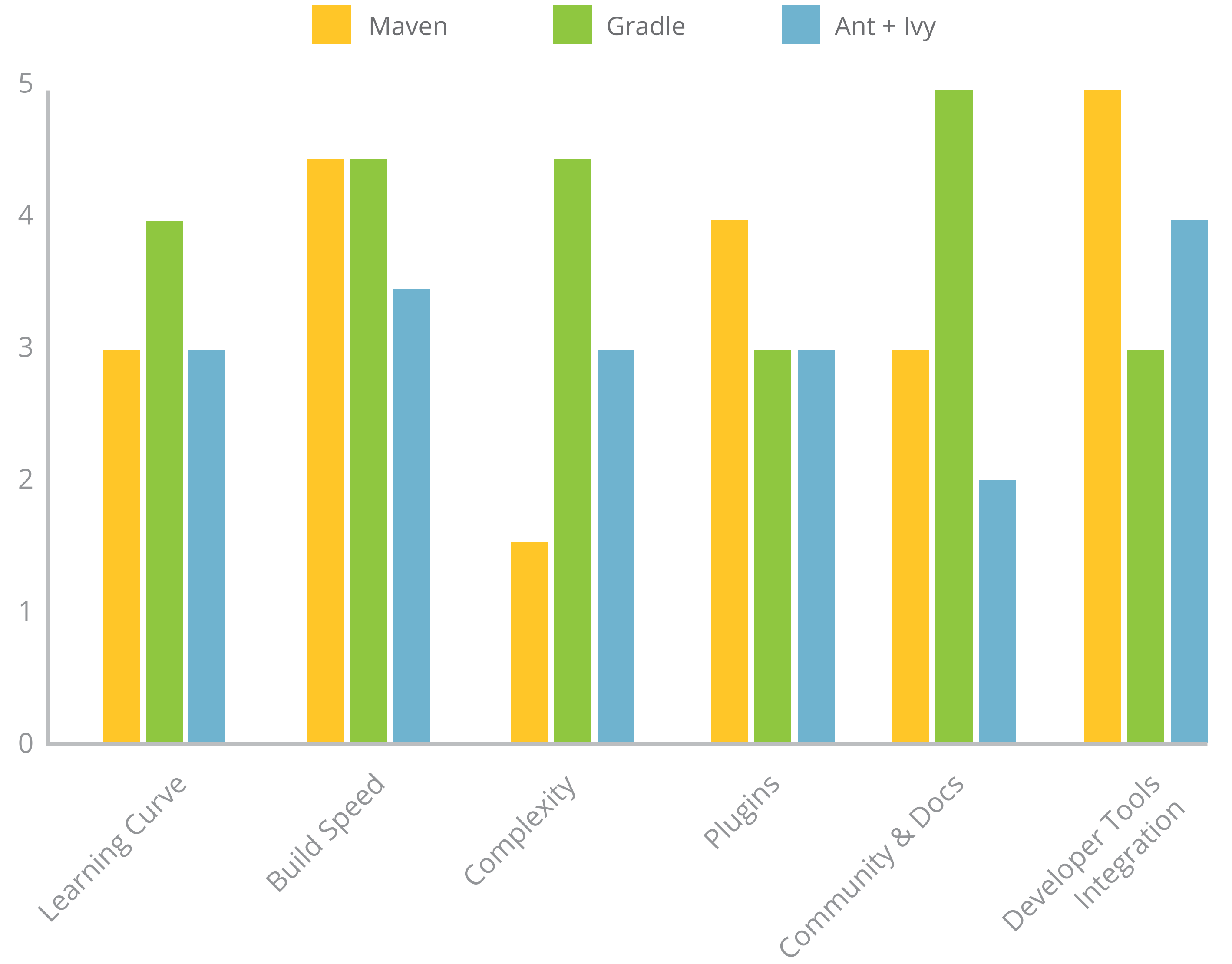
This is what you've been waiting for, right? Which one is going to take 1st place among build tools? Well, not to spoil anything, but Ivy definitely isn't taking home any awards ;-)

The Results of comparing Maven, Gradle and Ant + Ivy

Once again, as we've done with other reports of this kind, we're not just going to score categories and add them up assuming each category is just as important as the last and put them in a table like this:

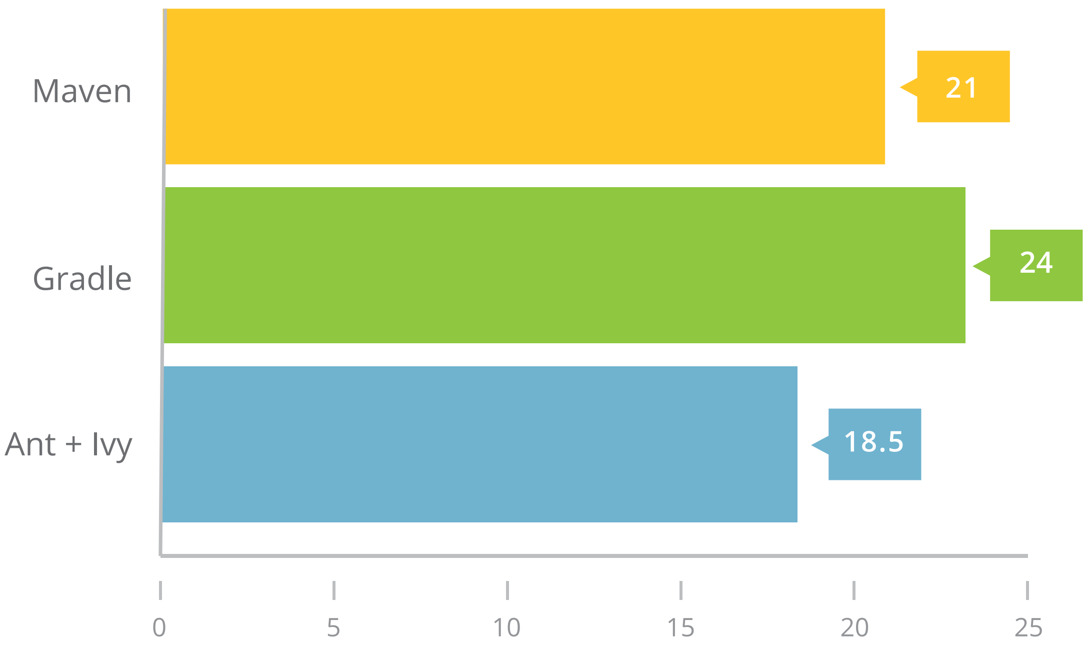
	maven	gradle	<APACHE ANT> ivy
Learning Curve	3	4	3
Build Speed	4.5	4.5	3.5
Complexity	1.5	4.5	3
Plugins	4	3	3
Community & Docs	3	5	2
Developer Tools Integration	5	3	4
Total	**21**	**24**	**18.5**

Score Breakdown by Category



REBELLABS

Grand Totals



And then sweepingly announce that Gradle is the greatest build tool EVAR and everyone should use it. That would be irresponsible and wrong, so we won't do that (seriously though, consider it!).

Instead we're going to look at a number of use cases, profiling typical developers in the industry today and weighing category scores for areas which are most important to each particular developer. Each use case will mark categories with a High, Medium or Low rating which adds a score multiplier for that category of 2x, 1x and 0.5x respectively. The developer profiles for each of our use cases this section are real live developers, true story. Only names, faces, job titles and bios may have been altered! ;)

Without further ado, let's check out our use case list:

- **Hobby developer** using GitHub as their repository
- **Open Source** project developer based in Eclipse, contributed to by many worldwide
- **Medium-sized business**, working in two countries. Traditional working, yah, we use waterfall.
- **Global, enterprise business**. Almost a hundred thousand employees, split across almost as many countries!

Hobbyist Developer

DEVELOPER PROFILE - MICHAEL

Michael codes in his day job, but his passion is at home. He can't wait to finish at 5:30 to get home and play with his pet projects on GitHub. He loves contributing to bleeding edge projects on github and has created many which others contribute to. His favourite languages are Java, Groovy and Ceylon. He loves to experiment with new code and apps and loves being one of the first beta testers. Nothing in technology scares him. Michael is a Capricorn who also enjoys small dogs. He owns 5 chihuahuas and rarely does exercise.

BUILD TOOL PRIORITIES

As a hobbyist developer, Michael is always creating new projects so it's really important to him that there isn't a high barrier to entry to get projects created from scratch. None of his pets can assist him if he were to hit a bug or problem in his build, so it's crucial that a community is there to support him in addition to good documentation. Michaels projects never grow too large, so build speed has never really been so important to him, nor has worries that if the project becomes too large, it will become unmanageable. With all this in mind, let's see how the scores are affected.

WINNER - GRADLE

For hobby projects, having a short learning curve and access to great documentation with the ability to interact deeply with the tool's community are the most important features. With consistently updated, attractive documentation and an active community, Gradle wins for this user profile.

PRIORITY	CATEGORY	maven	gradle	APACHE ANT ivy
H	Learning Curve	6	8	6
L	Build Speed	2.25	2.25	1.75
L	Complexity	0.75	2.25	1.5
M	Plugins	4	3	3
H	Community & Docs	6	10	4
M	Developer Tools Integration	5	3	4
	Grand Total	**24**	**28.5**	**20.25**

Open Source Developer

DEVELOPER PROFILE - SIGMAR

Sigmar doesn't pay for anything and has convinced both himself and his mother that Open Source Software is free. He hates 'the man' and will sooner adopt 5 chihuahuas than file a patent. He is extremely active and well known by the Apache and Eclipse foundations, as he contributes to many of their projects. One of Sigmar's greatest struggles is making Open Source projects accessible to developers all around the world, as he's keen on learning Mandarin. As a result he is keen on making the Open Source world suitable to all developers of all levels, and just like hemp shoes, it's the future.

BUILD TOOL PRIORITIES

Sigmar works with many many other developers with ranging abilities and interests. As a result it's important that all can get onboard quickly, so the learning curve must be low. As the project grows, it will be hard for others to track what's happening in build scripts, so it must be easy to maintain and low in complexity. As contributors to the Open Source project have different backgrounds, they'll likely have different styles and development environments, so support across other tools is important. Sigmar doesn't care too much for the community or docs side of things as his project has a community that he can ask questions and advice from first.

WINNER - GRADLE

The needs of an Open Source committer require that both the Learning Curve and Complexity are as low as possible. At the same time, integrated with other tools is a priority. Gradle's simplicity gives it an edge over Maven, although this is a pretty close category here. Even Ant performs reasonably well.

PRIORITY	CATEGORY	maven	gradle	APACHE ANT ivy
H	Learning Curve	6	8	6
M	Build Speed	4.5	4.5	3.5
H	Complexity	3	9	6
M	Plugins	4	3	3
L	Community & Docs	1.5	2.5	1
H	Developer Tools Integration	10	6	8
	Grand Total	**29**	**33**	**27.5**

Medium sized business, Traditional

DEVELOPER PROFILE - HANA

Hana has been a developer for over 19 years now and is convinced Java is being over engineered. As she once put it - "Java has been working just fine for years without Lambdas and Generics, why bloat the language for no reason?". Hana is a big fan and advocate of Waterfall - "I learn from the experience of those around me. My team has barely changed over 30 years and they all swear by it. I respect their views". Hana's company is a mid sized business that consider it risky to take on new challenges. Hana believes change is a swear word. She owns a 1984 DeLorean that is both beige on the inside as well as the outside.

BUILD TOOL PRIORITIES

Hana's team is quite static, so the most important thing isn't about how to get up and running with new projects, but more about the long term maintenance of those projects and what happens if something goes wrong. Complexity is a big issue as the project has been running for many years and more time and effort is spent in the servicing of the project as new features. Build speed isn't overly important at all and as everyone uses a standard company wise development environment, integration has never really mattered.

WINNER - GRADLE

Stable, medium-sized businesses that value low Complexity for maintenance in the long-term and good Community and Docs in case some wrecking-ball of a bug comes to smash their stability should find that Gradle supports their needs best, by a fairly wide margin. From there, Maven and Ant are more or less interchangeable.

PRIORITY	CATEGORY	maven	gradle	APACHE ANT ivy
M	Learning Curve	3	4	3
L	Build Speed	2.25	2.25	1.75
H	Complexity	3	9	6
M	Plugins	4	3	3
H	Community & Docs	6	10	4
L	Developer Tools Integration	2.5	1.5	2
	Grand Total	**20.75**	**29.75**	**19.75**

Enterprise Developer

DEVELOPER PROFILE - JURI

Juri is an Enterprise developer for a large multinational company. He loves saying he's cutting edge when in fact he just deals with middleware. His team is split across many countries and has different levels of expertise in each of the teams he develops with. The application Juri helps to develop is a huge multi module beast, which takes hours to build and package. The application is a Java application through and through and that's unlikely to change for a long time. Juri says the words 'mobile', 'cloud computing' and 'backburner' a lot, but doesn't really know much about them.

BUILD TOOL PRIORITIES

As an enterprise developer, projects last 10s of years and are rarely created from scratch, so build scripts follow suit. It's not about how easy it is to get started, it's more about whether we can make this 5 hour build turn into a 4 hour build! Build complexity isn't a major problem as we have teams of people who are devoted to working on and maintaining build scripts. Similarly we use their expertise rather than the community and docs purely because we can walk over to their desks to ask them. While we do have a corporate suggested development environment, we do use many app servers and have the odd hundred or so developers who want to branch away from the norm and use a different IDE, so tools integration is again important.

WINNER - MAVEN

With Build Speed and Integrations with Developer Tools considered High Priority, we now get to see the strengths of Maven come into play. Robust performance and basically integrating with everything under the sun means that Maven is a great tool to [continue to] use in large organizations with global enterprise needs.

PRIORITY	CATEGORY	maven	gradle	APACHE ANT ivy
L	Learning Curve	1.5	2	1.5
H	Build Speed	9	9	7
M	Complexity	1.5	4.5	3
M	Plugins	4	3	3
L	Community & Docs	1.5	2.5	1
H	Developer Tools Integration	10	6	8
	Grand Total	**27.5**	**27**	**23.5**

CHAPTER IV:
SUMMARY, CONCLUSION AND GOODBYE COMIC :-)

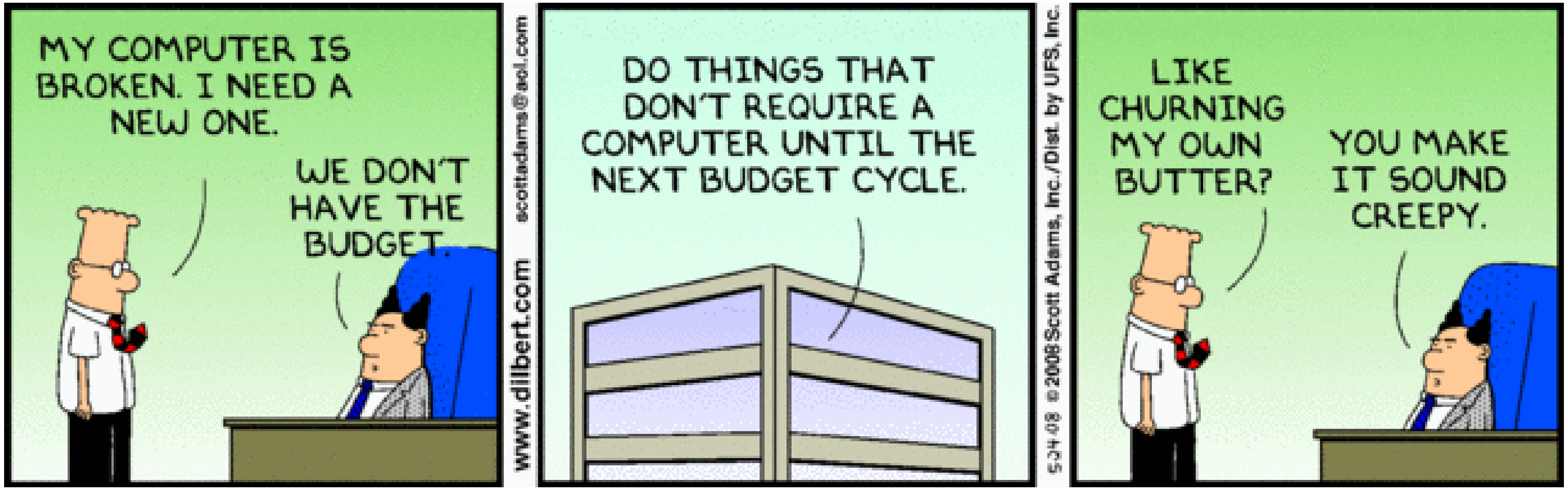
Here’s the TL;DR section for those with a short attention span or busy schedule! We give a complete summary of each chapter, plus a conclusion of our findings and scores.

Summary of findings so far

GRADLE IS OUR WINNER!

Gradle is followed reasonably closely by Maven in 2nd place and Ant + Ivy bringing up the rear in 3rd place! We cannot say that any of this is particularly surprising, considering the relative stagnation--compared to web frameworks, cloud development platforms, etc--in the build tools landscape. The newcomer, Gradle, was able to build on over a decade of experience and pitfalls and took most of those lessons to heart, it seems.

Just to remind you, here is a recap of the feature categories and user profiles that we looked at, and which tool came out on top for each one...


MY COMPUTER IS BROKEN. I NEED A NEW ONE.
WE DON'T HAVE THE BUDGET.
scottadams@aol.com
www.dilbert.com
DO THINGS THAT DON'T REQUIRE A COMPUTER UNTIL THE NEXT BUDGET CYCLE.
5-24-08 © 2008 Scott Adams, Inc./Dist. by UFS, Inc.
LIKE CHURNING MY OWN BUTTER?
YOU MAKE IT SOUND CREEPY.

Source: http://dilbert.com/strips/comic/2008-05-24/

Which build tool wins in the feature categories?

LEARNING CURVE

Gradle wins this category in light of Maven and Ant's complex set ups. Even though users need to get used to the DSL style and change to Groovy's syntax, Gradle's extreme simplicity and intuitiveness make it a winner over its competitors.

BUILD SPEED

Maven earns top points here, with speeds faster than Gradle in nearly every category. Oddly, Ant alone was faster than both Maven and Gradle in most areas, but Ivy's dependency resolution and download time was so lengthy that it ruined Ant's overall score. Too bad.

COMPLEXITY

Gradle wins here based on its extreme simplicity, readability and customizability. Maven received especially low scores mainly due to its strict conventions and invisible rules. Ant + Ivy is slightly better at letting you make all kinds of customizations, but the boilerplate and verbosity doesn't make things simpler in that sense.

PLUGINS

Maven wins this category, since it's quite simple to create a plugin yourself in case the hundreds of other plugins available don't satisfy your needs. Ant + Ivy is similar to Maven in this way, although using your own plugins is a bit more complex. With Gradle, plugin customizations are very simple (you can even write them directly into the build script and share them among your team), which is good because there are fewer existing plugins available.

COMMUNITY AND DOCS

Gradle wins here, mainly because of the positive user experience and vibrant (albeit small) community. Gradleware brings order to an otherwise fragmented tool space, with attractive, easy to use docs and professional support for users. Maven and Ant + Ivy do not compare on either level, although a long history and large community of current/past users can be a valuable resource.

DEV TOOLS INTEGRATION

Maven gets the cigar here, followed closely by Ant. When it comes to integration and feature support with IDEs, App Servers and Continuous Integration servers, Maven supports everything under the sun, with Ant + Ivy suffering slightly in the CI server category. As the newcomer, Gradle lacks in the App Server support category, and matches up with Ant + Ivy in the CI area.

As we can see, Maven won in as many categories as Gradle (three each), but the overall scores gave Gradle the advantage. Gradle never had a score below 3, whereas Maven suffered in the complexity department.

Which build tool wins for user profile (i.e. use case)?

HOBBY DEVELOPER

Gradle, with its short learning curve and availability of great documentation and an active community, makes the best option for hobby project developers.

OPEN SOURCE DEVELOPER

Similarly for those hobbyists, **Gradle** works best for open source project development where having low complexity and simple learning curve is valuable to saving time and keeping up maintenance in the future. However, Maven was not far behind.

MEDIUM-SIZED BUSINESS

Gradle will best support the needs of medium-sized businesses, which require a low level of complexity and strong community and documentation.

GLOBAL, ENTERPRISE BUSINESS

Maven is still going to benefit the largest global organizations out there, namely due to robust build speed and it’s basically ubiquitous integration points with different technologies. However, Gradle was not far behind at all, and Gradleware’s professional services are designed to support large enterprises.

Conclusion - experiment with Gradle for your next project

If we were forced to conclude with any general recommendation, it would be to **go with Gradle if you are starting a new project**.

However, there are factors personal to you that we are not able to incorporate into our judgement call--this is a guide based on feature categories and user profiles to fit common use cases, not specific ones.

Ultimately, like anything in life really, you need to decide for yourself. Gradle's simplicity and low barrier to entry make it ideal for testing and experimenting. Here is how each tool stacked up in the end...

As we've stated in previous RebelLabs articles, Maven was a game changer back in 2002, bringing "balance to the force" when it came into the build tool space with dependency management and simpler, less verbose setups and build scripts (compared to its predecessor Ant).

Large enterprises seem to have migrated from Ant + Ivy over to Maven in significant numbers in recent years. Since 2010 (or even before, based on the statistics in Chapter I), Maven has become the *de facto* build tool as it steadily gained ground over the competition.

However, such dominance has actually returned to strike Maven a few blows: having to follow the "Maven Way" with its hidden rules, strict conventions and inability to easily customize your build script (instead, relying on tons of existing plugins or writing your own to get certain things done) has exposed the soft underbelly of Maven and led to relatively large gains by Gradle since 2012.

Now, the greatest threat to Maven's dominance and old-school XML ways is Gradle's shiny DSL syntax, benefitting from flexibility with Groovy, professional support and documentation from Gradleware and a community that actually cares about the tool deeply. At the same time, Maven's fast build speed, wide-spread support for integrating with developer tools and a bazillion plugins will keep Maven in the game for a long time.

The guys who started Gradle obviously set out to do something different, and find niches in which to innovate where long-existing tools had run out of steam. Most developers that have been using Ant + Ivy or Maven for long enough will be slightly unnerved by how simple and flexible it is (i.e. starting up with an empty or single-line build script, writing plugins and customizations directly into the script).

In our experience, it seems that even hardcore Java adherents don't mind learning a bit of Groovy in order to make their build tool experience a bit more satisfying. Although, they'll probably never admit it ;-)

The last decade has seen relatively little movement in the build tools segment (people still recommend using MAKE every once in a while!), so you can rest assured that the fast growth, excellent features and success of Gradleware itself (Google's Android OS recommends building apps with it) is setting Gradle up for something promising in the future.

Oh Ant. You know, there really isn't anything wrong with Ant--especially once you're accustomed to its nuances. Seeing as how it was created back when Clinton was the US President, there is a good chance that any dev or organization that's been around a while has used it.

Ant is old school--XML, lots of boilerplate, simple functionality that, according to build speeds in most cases, is certainly good enough for modern needs (until Ivy's slow dependency checks drive you mad). These sorts of manually-configured tools with little invisible magic going on in the background are losing ground to "no-build-script-even-needed" Gradle, which does a lot of work for you but also sets a precedent for future developers who have no idea what's going on and come to rely too much on automagic.

Ant's only major liability seems to be Ivy, which does little to make to overall user experience better. However, if you want dependency management for your project, then this is the way to go. Our suggestion is to use elements of Ant that are good--simplicity, integration and Ant's extensive plugin library (including the Ivy plugin!)

Regardless of these results, we cannot with a good conscience recommend that you switch from Ant or Maven to Gradle for your projects--however, it might pay off in the future to experiment and test Gradle out a little bit, just in case the day comes (cough hack cough GlassFish cough hack) when migrating to something else becomes viable.

Build tools are still among the more reviled technologies in the Java development landscape, so if you're already using one of the tools and it's working for you, then the benefits of switching to something else are probably not very significant. You should have a very good reason to migrate, and in the end it's probably better to stick with the devil you know.

Contact Us

Twitter: @RebelLabs
Web: http://zeroturnaround.com/rebellabs
Email: labs@zeroturnaround.com

Estonia
Ülikooli 2, 4th floor
Tartu, Estonia, 51003
Phone: +372 653 6099

USA
399 Boylston Street,
Suite 300, Boston,
MA, USA, 02116
Phone: 1(857)277-1199

Czech Republic
Osadní 35 - Building B
Prague, Czech Republic 170 00
Phone: +372 740 4533

This report is brought to you by:
Juri Timošin, Sigmar Muuga, Michael Rasmussen, Simon Maple, Oliver White, Ladislava Bohacova