JAVA TOOLS AND TECHNOLOGIES LANDSCAPE 2016

SURVEY RESULTS FROM 2,040 GEEKS

Plus, we donated $1000 to Devoxx4kids!

REBELLABS by ZEROTURNAROUND

TABLE OF CONTENTS

EDITOR'S NOTE

Hurrah, it's time to present the RebelLabs Tools and Technologies Landscape report! This year we once again ran our infamous Tools and Technologies survey — asking all sorts of questions about application servers, microservices, Java versions, web frameworks, agile practices and much more. We've got some really interesting and promising results to show you which, once again, I really enjoyed mining the data for and writing up.

This is the 7th (yeah, the seventh!) productivity report we've produced. All of which start with a survey in which we examine the amazing Java community to better map out the landscape in which they work. This includes looking at the tools that they use and the direction in which the Java ecosystem is heading.

Overall, we received 2040 completed survey responses. This gave us plenty of data with which I've been able to heavily improve my Excel knowledge. We also managed to raise $1000 for Devoxx4Kids (a wonderful charity) because we reached our 2000 response target. Great job to all those who took part and answered all of our questions!

I sincerely hope you enjoy reading this report as much as I have enjoyed writing it. We hope your future decisions can be made with confidence using data instead of guesswork.

Before we start, I'd love to share some of the most significant findings from our raw survey data! A little taster, if you will. On the following page, we've created a leaderboard with some of the most popular tools and technologies used in 2016. Enjoy!

SIMON MAPLE (@sjmaple)
Head of Developer Advocacy, ZeroTurnaround
and RebelLabs witch doctor writer

Developer Productivity Report 2015: Java Performance Survey Results

Java Tools and Technologies Landscape for 2014

RebelLabs Tools and Technologies Leaderboard 2016

REBELLABS by ZEROTURNAROUND

Maven

Over **two in three (68%) devs** useMaven as their main build tool

git

Over **two in three (68%) devs** use Git as their version control

Java 8

Almost **two in three (62%) devs** use Java 8 in production

Jenkins

Three in five (60%) devs use Jenkins for CI

IntelliJ IDEA

Almost **one in two (46%) devs** useIntelliJ, the most popular IDE in the survey.

spring

Over **two in five (43%) devs** use Spring MVC

Tomcat

Over **two in five (42%) devs** useTomcat server in production

ORACLE DATABASE

Almost **two in five (39%) devs** use Oracle DB in production

Microservices

Over **one in three (34%) devs** have adopted a microservices architecture

docker

Almost **one in three (32%) devs** use Docker in production

Java EE 7

Over **three in ten (31%) devs** use Java EE 7

spring

Almost **three in ten (29%) devs** use Spring Boot

Based on the **2016 Developer Productivity Report** by **RebelLabs**

TOOLS AND TECHNOLOGIES LANDSCAPE REPORT 2016 AT A GLANCE

> *Technology is nothing. What's important is that you have a faith in people, that they're basically good and smart, and if you give them tools, they'll do wonderful things with them.*
>
> — STEVE JOBS

Tools and **Technologies**

Java, the language and the platform, owes much of its fame and longevity to the libraries, frameworks and tools which together make up its ecosystem. No other programming language has been able to match the support that a rich ecosystem like the JVM has achieved. It's extremely important to understand how the ecosystem exists today, as well as its history. This way we can extract trends and patterns to understand the direction in which we are heading. In fact, all ecosystems are living organisms. They're born, they grow and become established, live their lives and one day will be replaced by a better, more suitable technology that solves the problem better. Of course the problems we deal with are changing too. Some ecosystems survive in care homes, refusing to pass away, like FORTRAN and COBOL.

This report sets out to understand the current landscape in Java, the JVM and its ecosystem, including JVM languages, application servers, build tools, Java and Java EE adoption, microservices architectures, web frameworks, performance tooling, agile processes and much more. Our goal is to show you the raw data as well as profile which respondents are more likely to prefer certain tech over others based on their organization and other preferences. We'll also look forward, making predictions based on the data for what the landscape might look like in a couple more years.

The **Data**

The survey itself was open between March and April 2016 and profiled application types, processes, tools and more. Overall we received 2044 responses and we had to eliminate 4 based on their responses to the years of experience question. While it would be amazing to speak to someone who states they have over 10,000 years of experience, we feared the entry could well have been bogus!

Each year, when we release our survey, developers everywhere stop what they're doing (unless operating heavy machinery) to answer our questions to the best of their ability, often missing deadlines and ignoring significant others while doing so. You don't believe me, do you? Well, that's because everyone's time is very valuable and we appreciate all the effort made by our respondents in completing each survey. As a reward for reaching 2000 completed surveys, we donated $1000 to a great charity called Devoxx4Kids. They run a great initiative, one that I've personally spent many hours volunteering for at events like JavaOne.

If you completed the survey, you helped this charity get a well-deserved $1000. This goes toward buying equipment for kids to learn and use and to helping pay for events that children can attend. High-fives all round — that's a great achievement!

The goal of Devoxx4Kids is to allow children to be more creative with computers and teach them Computer Programming while having fun. To accomplish this, the Devoxx4Kids worldwide teams organize events where children can develop computer games, program robots and also have an introduction to electronics. Teachers are computer professionals that are volunteering to spend some of their free time to transmit their passion to children.

This **Report**

The report is split into 3 sections. The first is a representation of the raw answers given to the survey questions. No fluff, no pivoting, just pure data and answers! Part 2 provides a more in-depth analysis, with pivoting on the data points to understand trends. Pivoting? Wasn't that a thing in my old physics class with a see-saw, some forces and some kind of formula? Well, maybe. But in this instance, we're asking more detailed questions about our data, based on the answers given to other questions. For instance, do early adopters that believe they're better than the average person in their role actually use technologies such as J2EE and Java 1.4? Read on to find out, the results may just surprise you! Part 3 looks at the data we collected in this survey and compares it to previous reports we've created — to see trends in adoption for tools and frameworks. But for now, let's start at the beginning with Part 1.

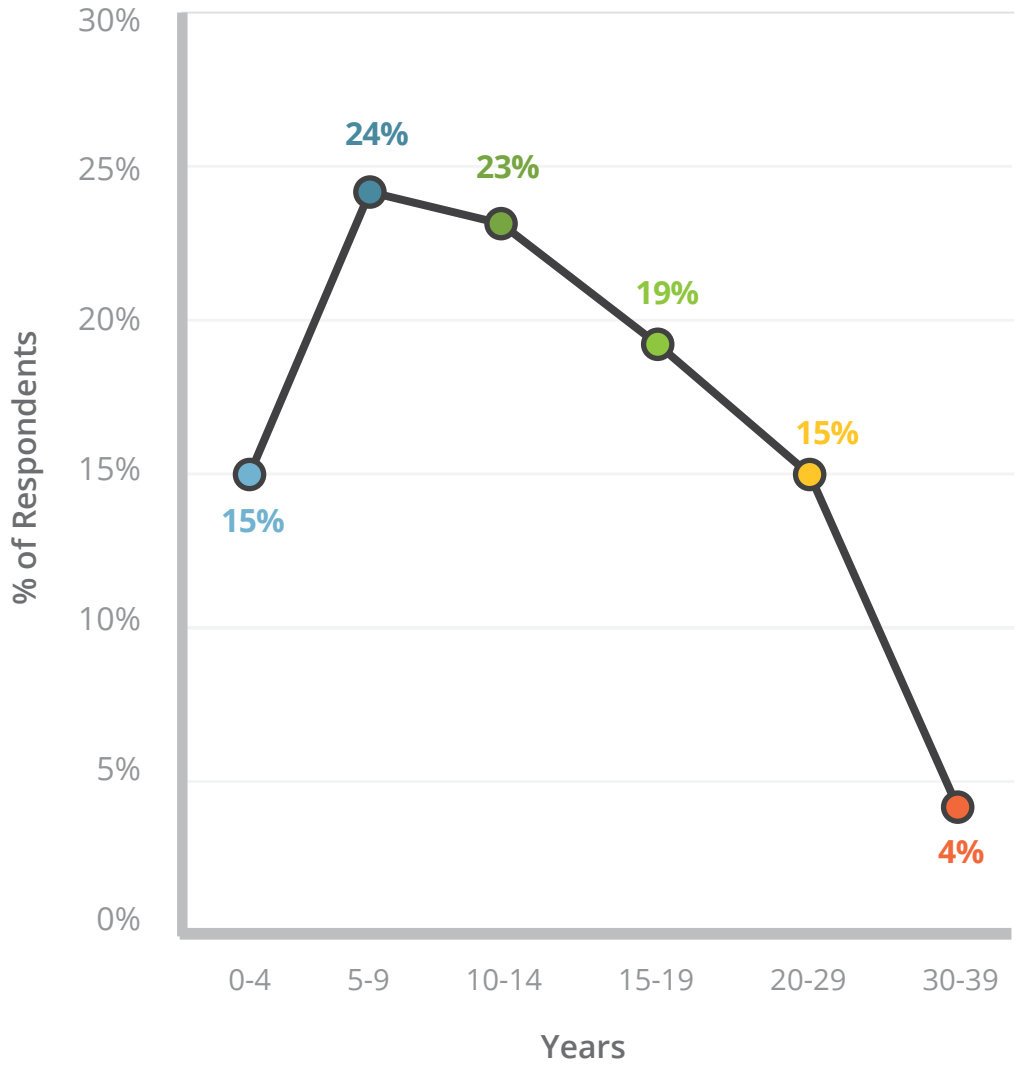
PART 1:
RAW DATA

Welcome to the real start of the report! And congratulations for getting past the fluff that precedes the actual fun part. Let's look at the core results. The raw data. In this section,our headings are the questions that were asked during the survey. There's only one question that's missing: *"What tool, technology or library are you super excited or proud about having used or planning to use in 2016?"*. Don't worry, we didn't forget it! We thought that question is better suited to Part 3's future trends section. So, let's begin.

How many years of experience do you have in software engineering?

This was a question we've not asked in past surveys. This year, we wanted to understand more about the type of person who responds to the survey as well as their technology preferences. We actually had a pretty good spread of results as you can see from *Figure 1.1*. Around half of the respondents exist in the 5-15 years of experience bracket. The overall median is 10 years of experience and the average is 12.2 years.

Figure 1.1 Years of experience in software engineering



REBELLABS by ZEROTURNAROUND

What is your **job description?**

This is a common question that we ask every year. Our respondent split was similar to previous years, with the majority being made up of Software Developers (54%), followed by Architects (18%) and Team Leads (12%) as you can see in *Figure 1.2*. The remaining split was fairly equal across consultants, management, C levels, and Dev advocates. The 'Other' category included operations and QA, among others. This was sad as it would have been great to get some of the opinions from operations teams on our microservices questions later. Overall, this spread of roles is what we would expect as the communities are heavily made up of developers and it's those communities which RebelLabs associates heavily with.

Figure 1.2 Respondents by job role

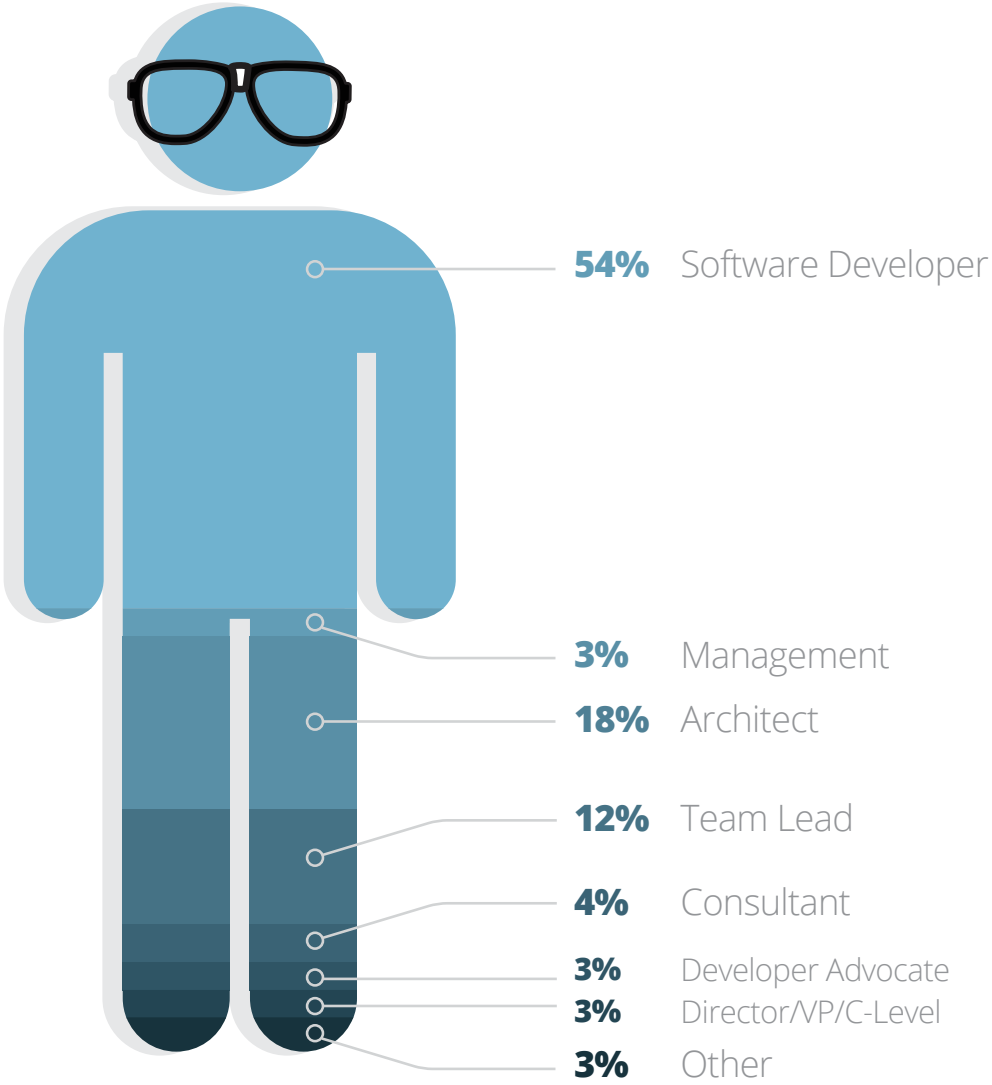
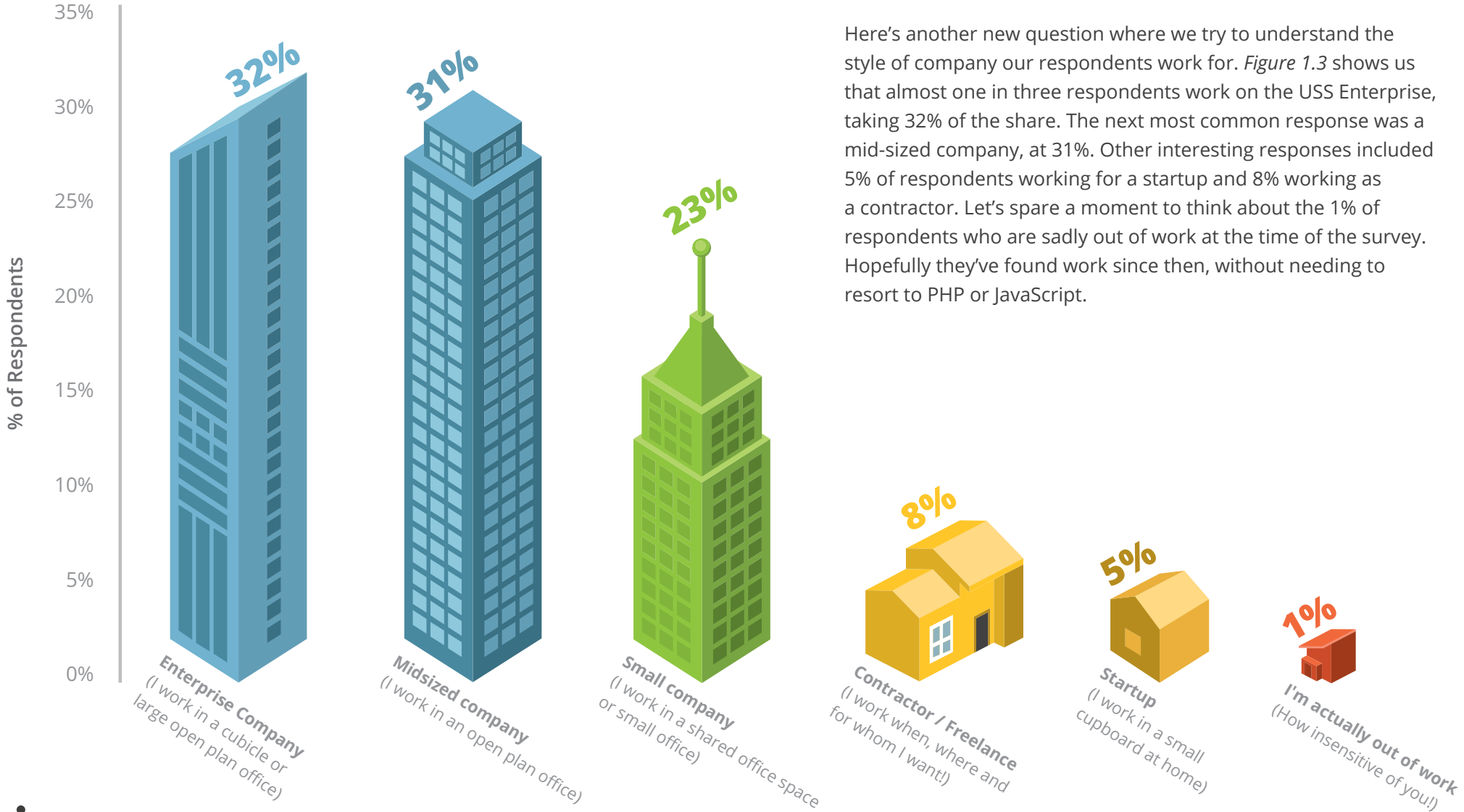


Figure 1.3 Respondent breakdown by company type



Which description best fits **the company you work for?**

Here's another new question where we try to understand the style of company our respondents work for. *Figure 1.3* shows us that almost one in three respondents work on the USS Enterprise, taking 32% of the share. The next most common response was a mid-sized company, at 31%. Other interesting responses included 5% of respondents working for a startup and 8% working as a contractor. Let's spare a moment to think about the 1% of respondents who are sadly out of work at the time of the survey. Hopefully they've found work since then, without needing to resort to PHP or JavaScript.

What kind of person would you say you are?

Many people or companies are opposed to trying something new, considering it wasteful to spend time on a technology that might not even exist in five years time. Or perhaps putting off an upgrade or version migration till they are sure all bugs/issues had been ironed out. This question was therefore designed to understand how likely someone was to try out a new tool or technology. We refer to respondents as early adopters or technology sheep based on their adoption preferences. This question will also be key in understanding which technologies or versions are considered established among respondents and which are still considered new and upcoming.

Figure 1.4 shows that the majority of people claimed they are technology sheep at 53%. A very strong proportion were early adopters with 44%. A mere 3% of people say that change is all bad and that they'd prefer to do real work rather than change. Be sure to check out Part 2 of this report to see which tools and technologies the early adopters are looking at!

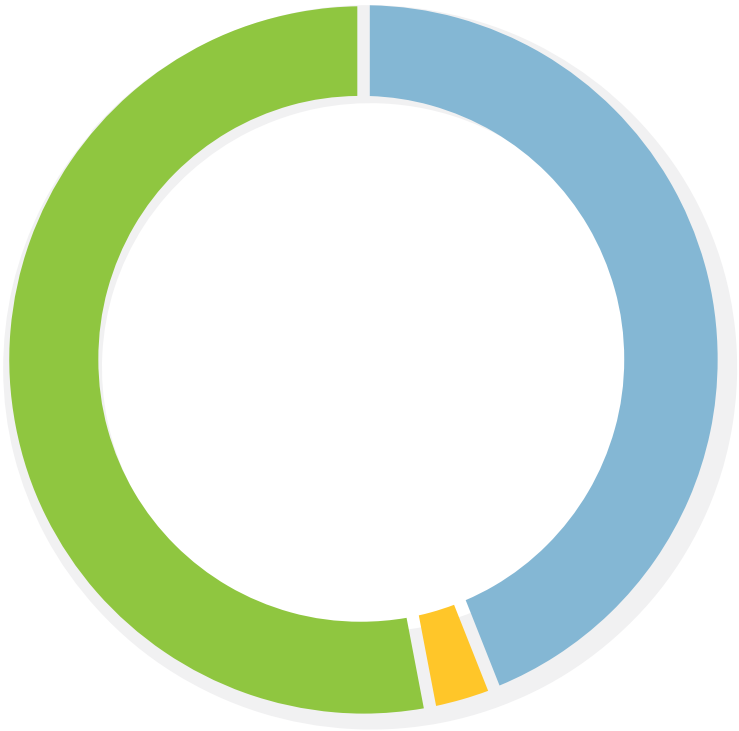


Figure 1.4 Early Adopters vs Technology Sheep

REBEL LABS by ZERO TURNAROUND

Would you say you are better than the average person in your role?

Now for my favorite question in this survey! We asked participants whether or not they thought they were better than the average person in their role. Now, all our RebelLabs readers are likely to be better than average in their role as they're trying to constantly better themselves by reading technical content. However, it's always amusing to see the results of possible overconfidence, or maybe arrogance, bubble up? Either way, we'll see just how good the 74% of people are, those who consider themselves to be better than average, when we study their tools, technologies and behaviours in Part 2. Tell me they use J2EE, please tell me they do!

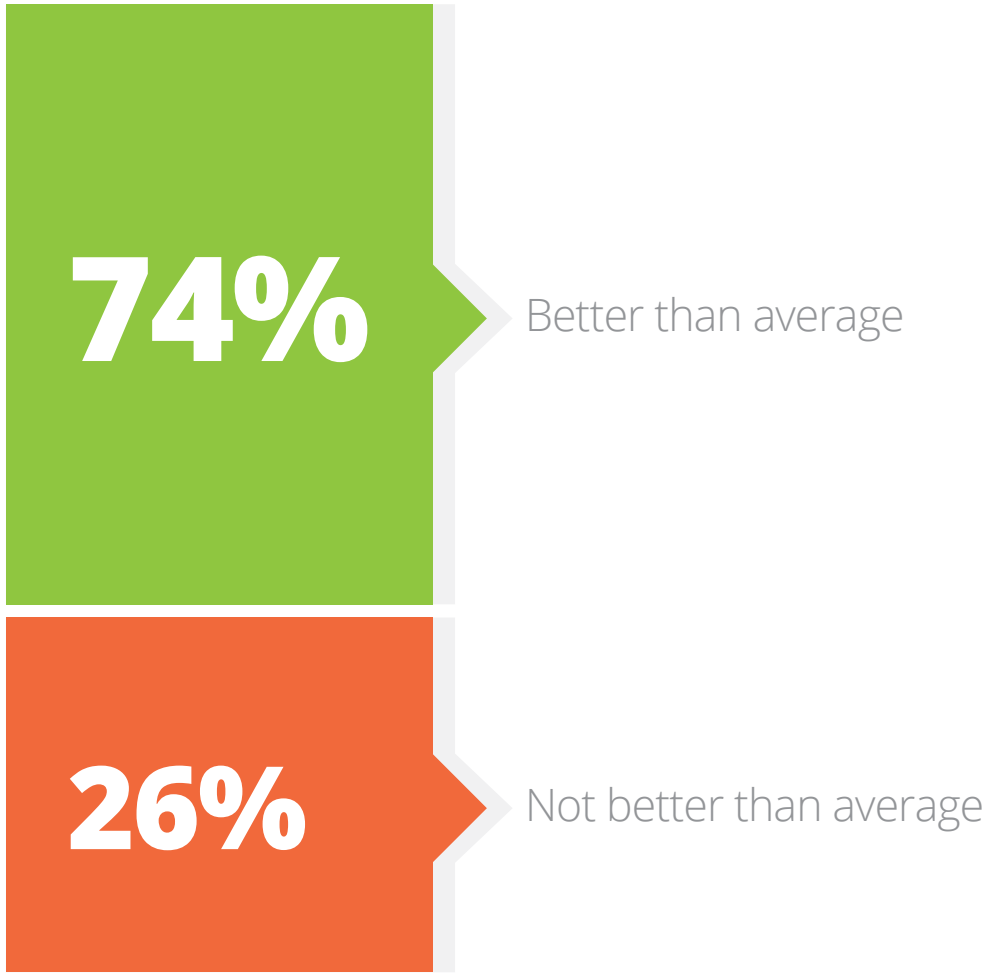


Figure 1.5 Three quarters (74%) of our of respondents think they're better than the average person in their role.

Which type of application describes the main project you work on?

We're always keen to get a breakdown of the applications that our respondents work with. As we can see from *Figure 1.6*, two thirds of respondents work on full stack web applications, as you might expect, with a further 18% working on backend code. The remaining votes are split between batch, mobile and libraries, with 6% working on desktop applications. The 'other' option contained middleware developers, tools developers, programming languages and more.

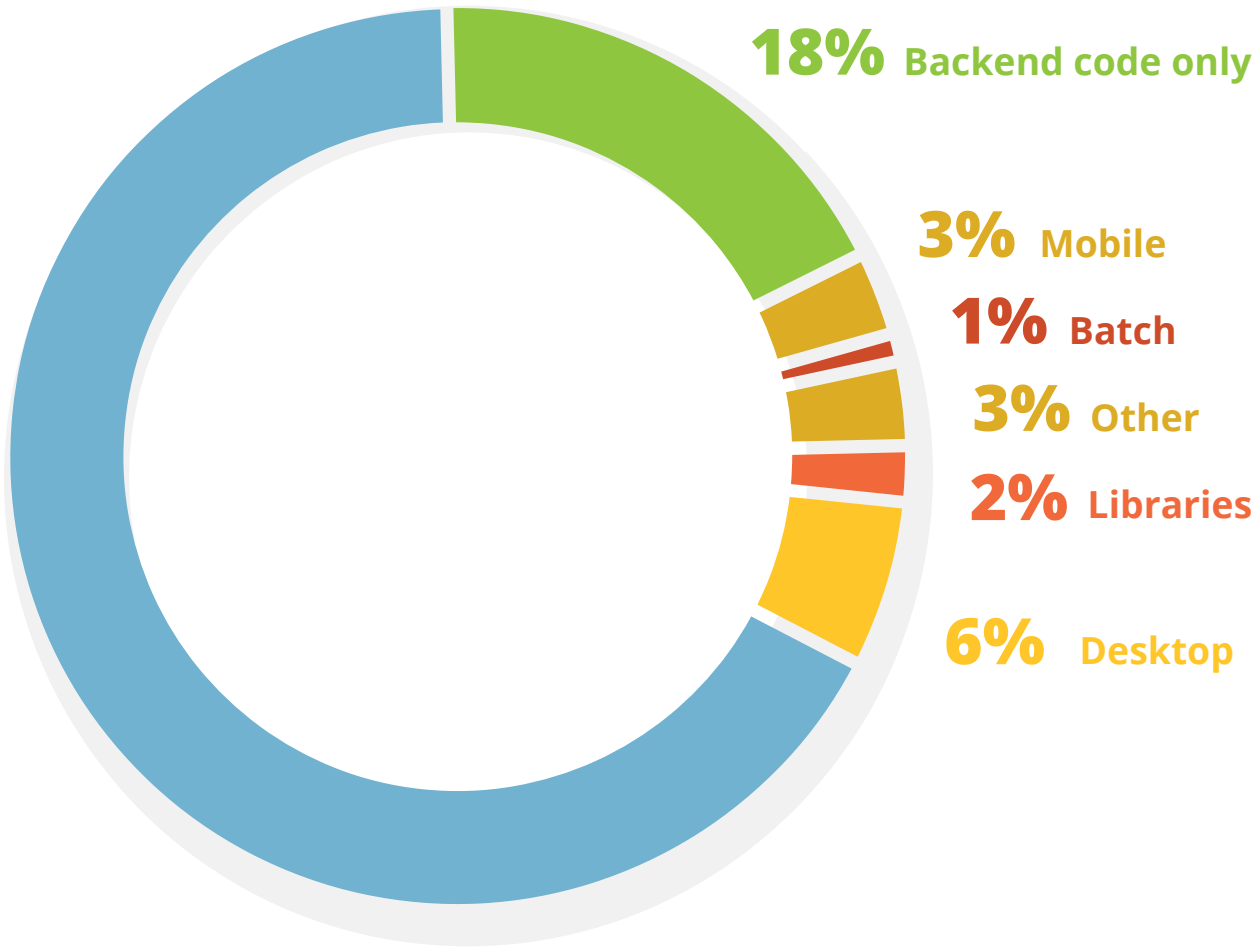


Figure 1.6 Breakdown by application types

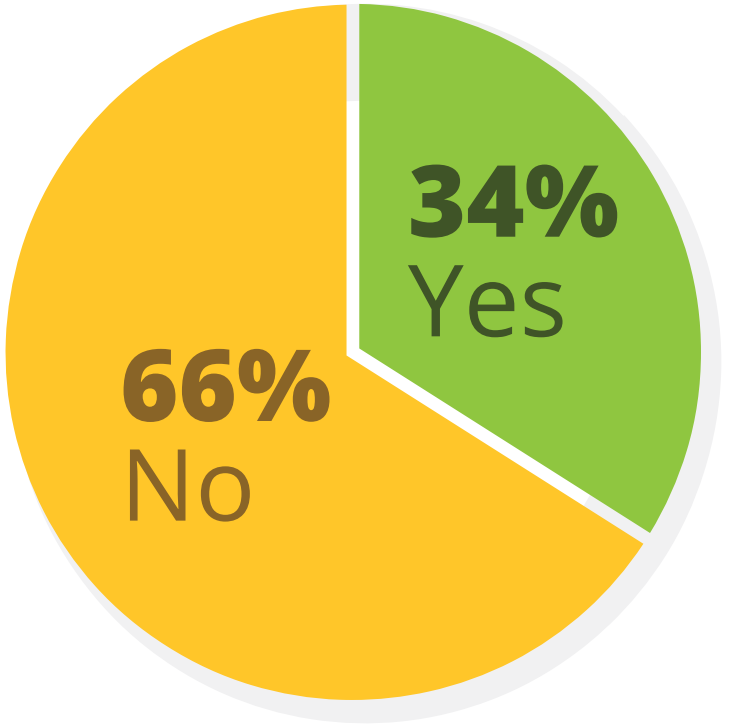
REBELLABS by ZEROTURNAROUND

Have you adopted a microservices architecture?

Microservices has been a popular buzzword in recent times, so we felt we had to include a specific question about it in this year's survey to better understand its adoption rate. This question is fairly simple on the face of it. However, I think that if we sat a few respondents next to each other, they may question whether each other has indeed implemented a microservices architecture or just adopted a couple of best practices. As we can see from *Figure 1.7*, two thirds of people (66%) have not adopted a microservices architecture whereas the remaining third (34%) have adopted a microservices architecture.

Next, we'll investigate this in more depth by asking two further questions. To those who **have not** adopted microservices, we'd like to know whether they plan to in the future. To those who **have** adopted microservices, does the new architecture make their job easier or harder?

Figure 1.7 Microservices adoption

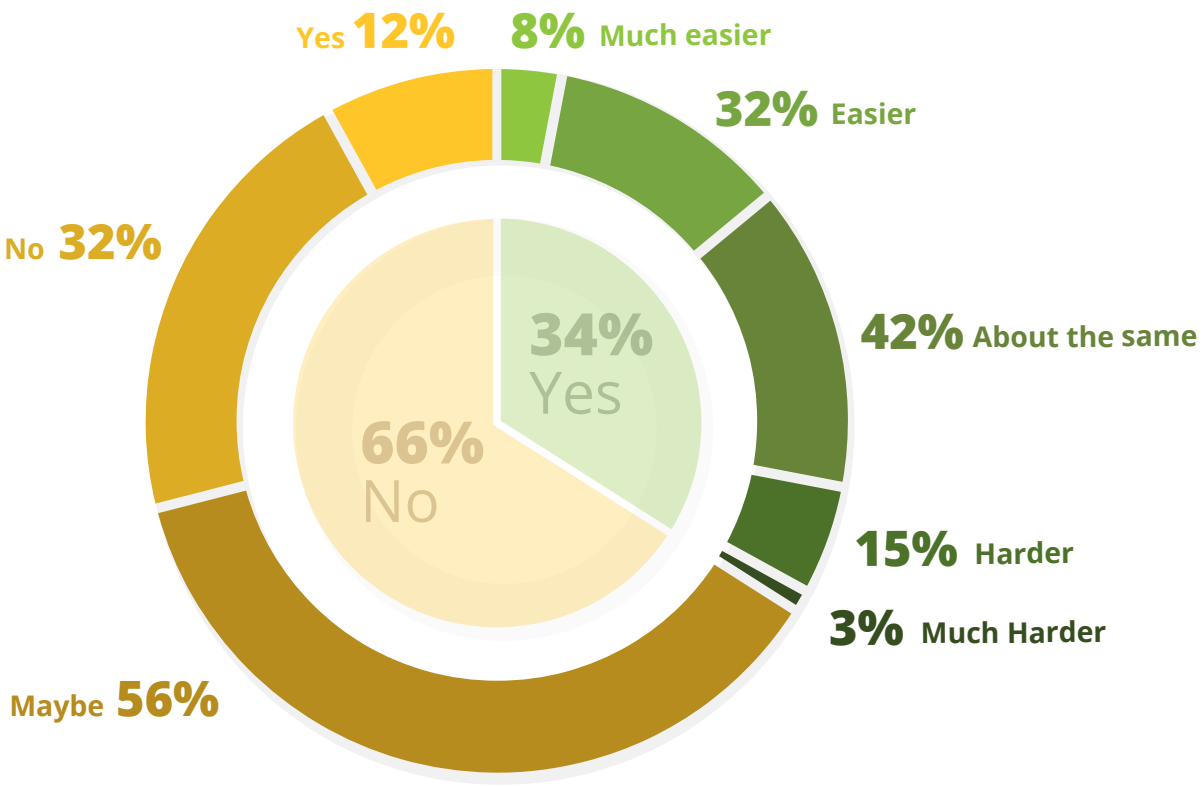


REBEL LABS by ZEROTURNAROUND

Figure 1.8 Microservices Plans and Results

[Non-Microservice Users]
Are you planning to move to a microservices architecture?

[Microservice Users]
Has moving to a microservice architecture made your job easier or harder?



REBEL LABS by ZERO TURNAROUND

Are you planning to move to a microservices architecture?

For those not using microservices, we asked if they were planning to move to that architecture. Looking at the yellow portion of the outer ring in *Figure 1.8*, we see just 12% of respondents said yes, they were planning to adopt them. Almost three times that number (32%) stated they did not want to move to a microservices architecture with 56% still making their minds up.

Has moving to a microservice architecture made your job easier or harder?

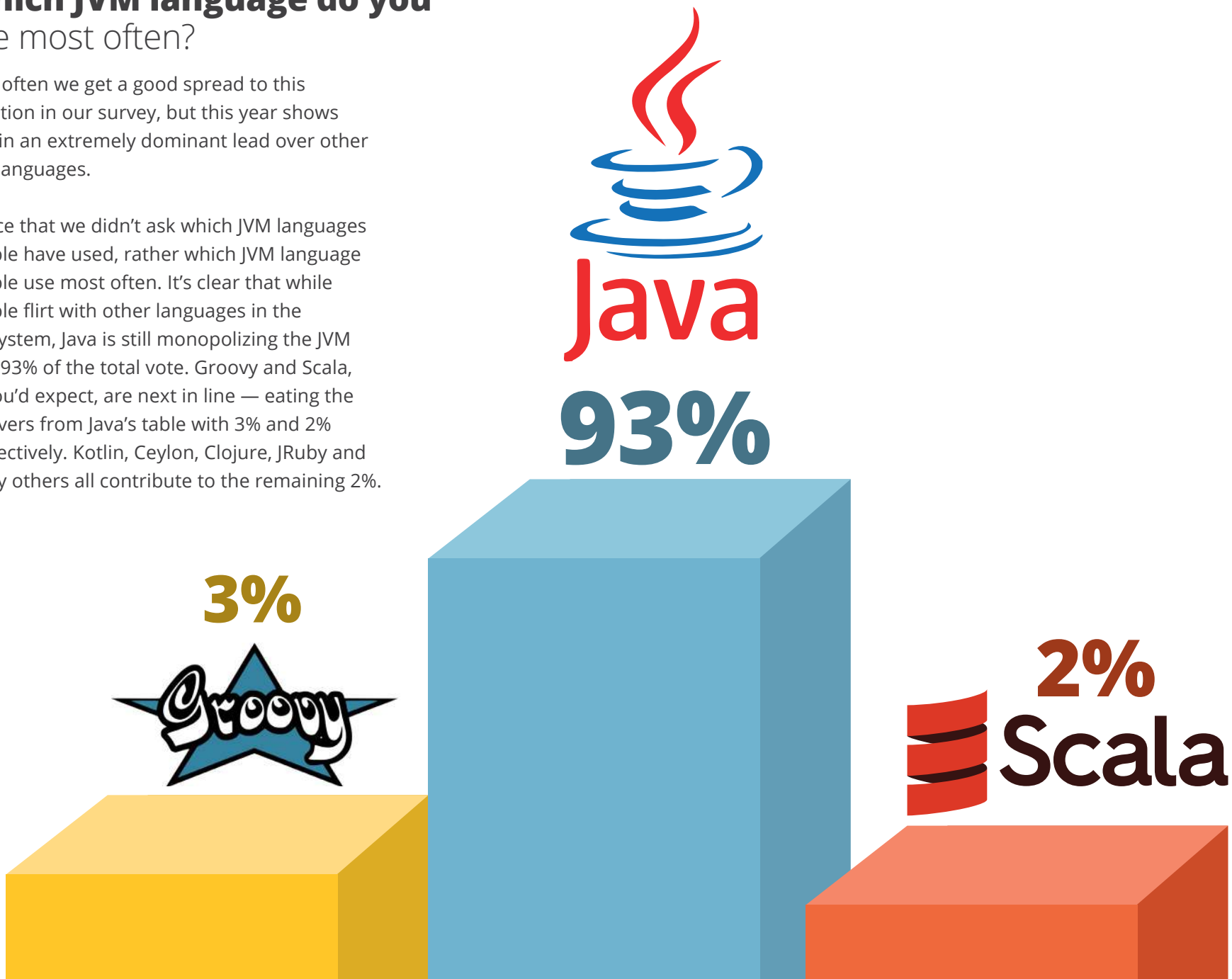
We asked those who stated they have already adopted a microservices architecture whether it made their job easier or harder. On the green portion of the outer ring in *Figure 1.8* we see that almost half of those who responded (42%) stated it made little difference to their job, with 40% saying their jobs became easier and 18% claiming their jobs were harder as a result.

Put in another way, we can say that approximately one in five people (18%) say their job is harder whereas four in five people (82%) state that their job is about the same or easier while using microservices. We could also state that three in five people state their roles are either the same or harder as a result of adopting microservices. Aren't statistics a wonderful thing! Luckily in this report, we show you all the numbers so you can make up your own minds!

Which JVM language do you use most often?

Very often we get a good spread to this question in our survey, but this year shows Java in an extremely dominant lead over other JVM languages.

Notice that we didn't ask which JVM languages people have used, rather which JVM language people use most often. It's clear that while people flirt with other languages in the ecosystem, Java is still monopolizing the JVM with 93% of the total vote. Groovy and Scala, as you'd expect, are next in line — eating the leftovers from Java's table with 3% and 2% respectively. Kotlin, Ceylon, Clojure, JRuby and many others all contribute to the remaining 2%.



Which Java version do you use to run your main app in production?

One of the most common questions people ask in the Java community concerns Java version adoption. This is mostly fueled by the exciting release of Java 8 back in March 2014 (I wanted to say recent, but it's already been over 2 years now!), combined with the aggressive removal of public updates on older versions. It's important to upgrade for a number of reasons, including performance, security and features. From a library or tooling vendor point of view, it's incredibly frustrating to support older versions of Java as it's harder to keep backward compatibility and make use of newer language features.

An important item to note at this point is that we're asking about the respondent's **main application** in **production** — rather than a proof of concept application running a service nobody really cares about. We can see from *Figure 1.9* how conclusive the results are, showing the success of the Java 8 release. 62% of respondents state that Java 8 is used to run their main production applications. 28% of respondents are on Java 7 with less than one in ten using Java 6. The 1% sitting in the other category consist of Java 5 users and some brave Java 9 power users! Needless to say, the success of Java 8 is very apparent in this one image of coffee cups. Now, it's time to grab an espresso and carry on reading! Next up is Java EE.

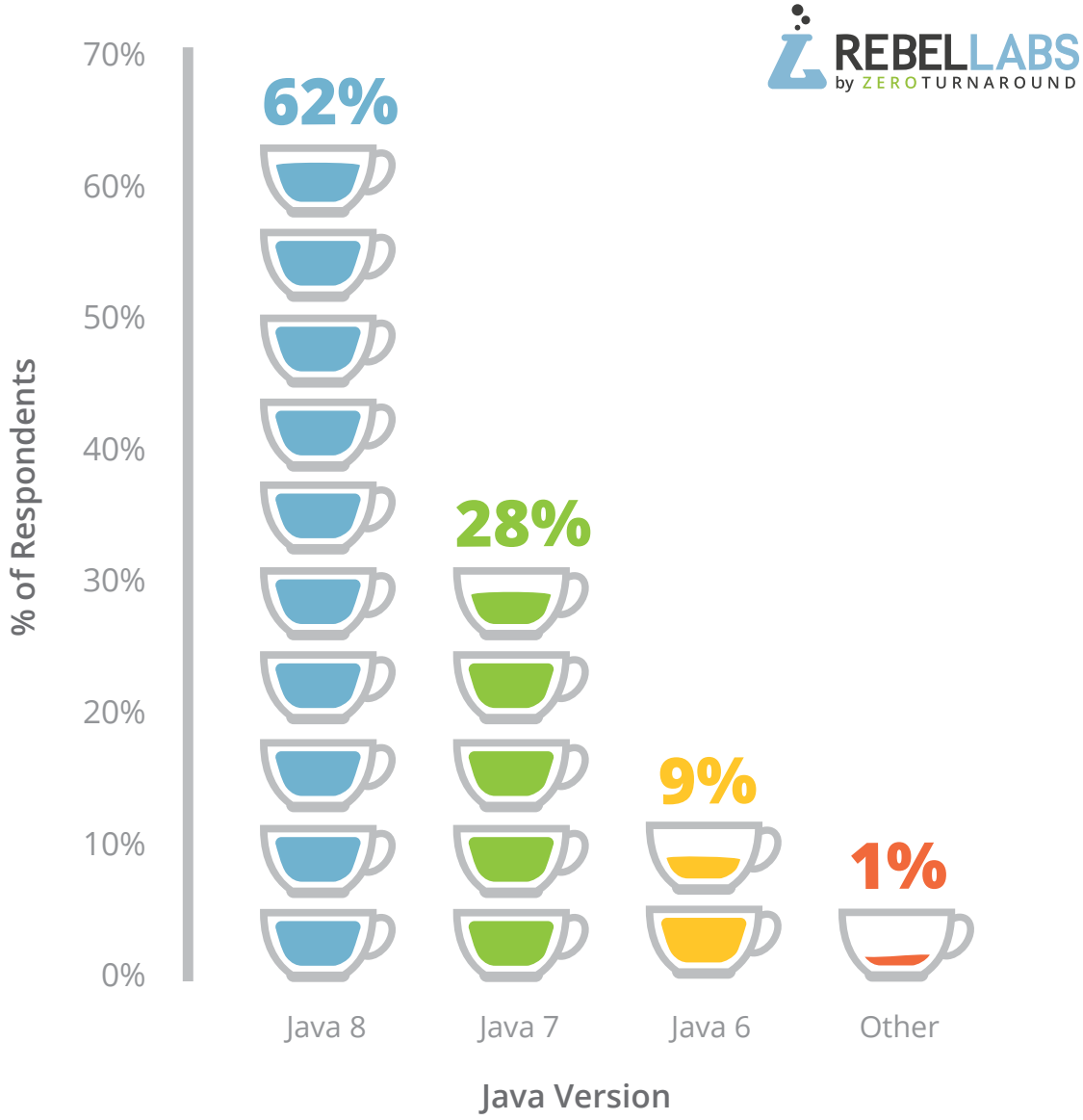
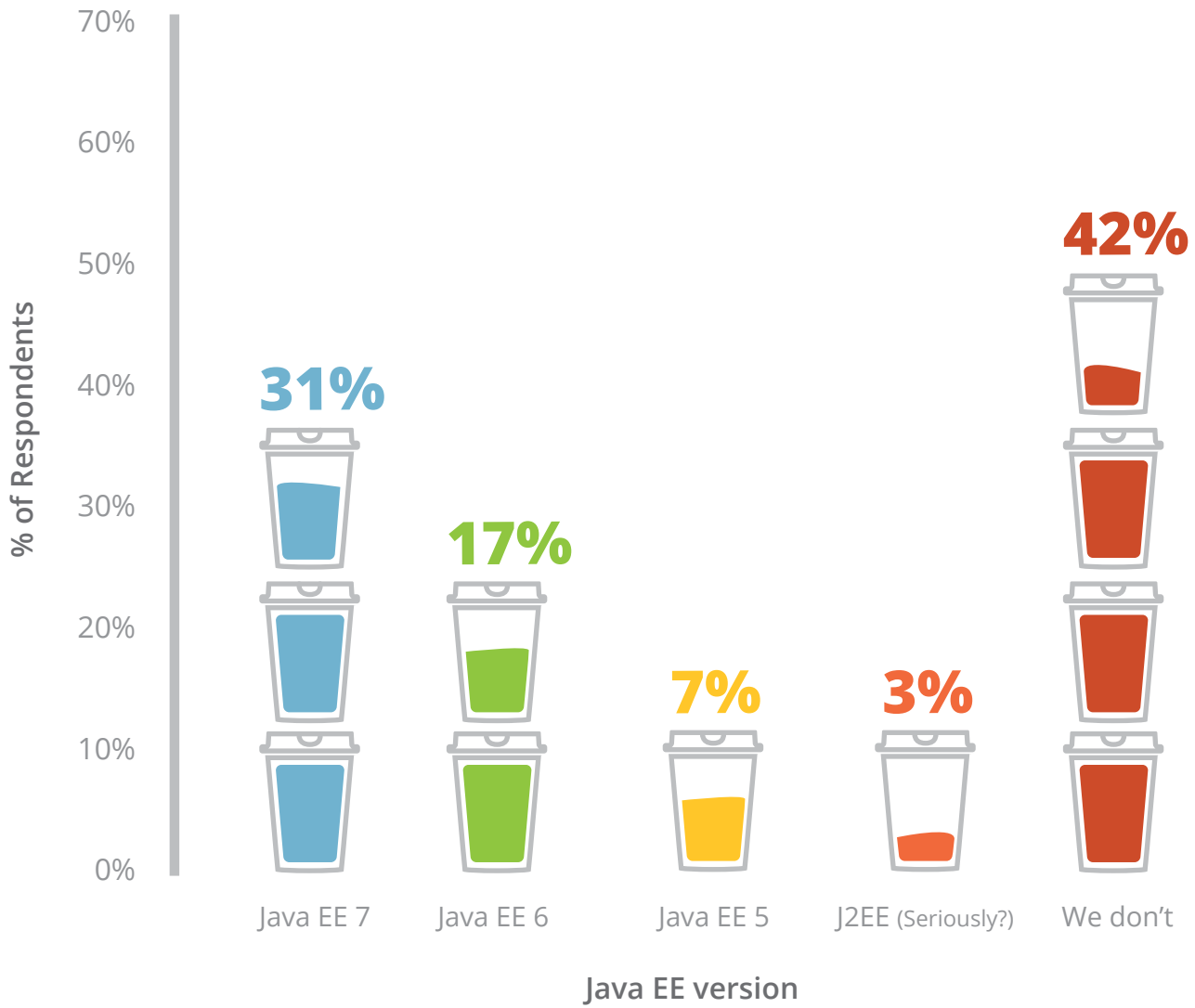


Figure 1.9 Java version adoption in production

Which version (if any) of Java EE do you tend to use most?

See, I told you it's Java EE next! Now that we've covered Java SE, let's take a look at how life looks in the enterprise world. Java EE 7 was released in June 2013 so it's no surprise it is the version which most have since migrated up to. Four out of ten respondents don't use Java EE. If we normalize the data, to only look at Java EE users, we can say 58% of them are on the current version, Java EE 7. It makes me very emotional to see almost one in ten people spending their working lives on a J2EE project, a specification that was released 13 years ago. Maybe we should have a minute's silence and finish the espresso we made in the last question.

Figure 1.10 Java EE version adoption

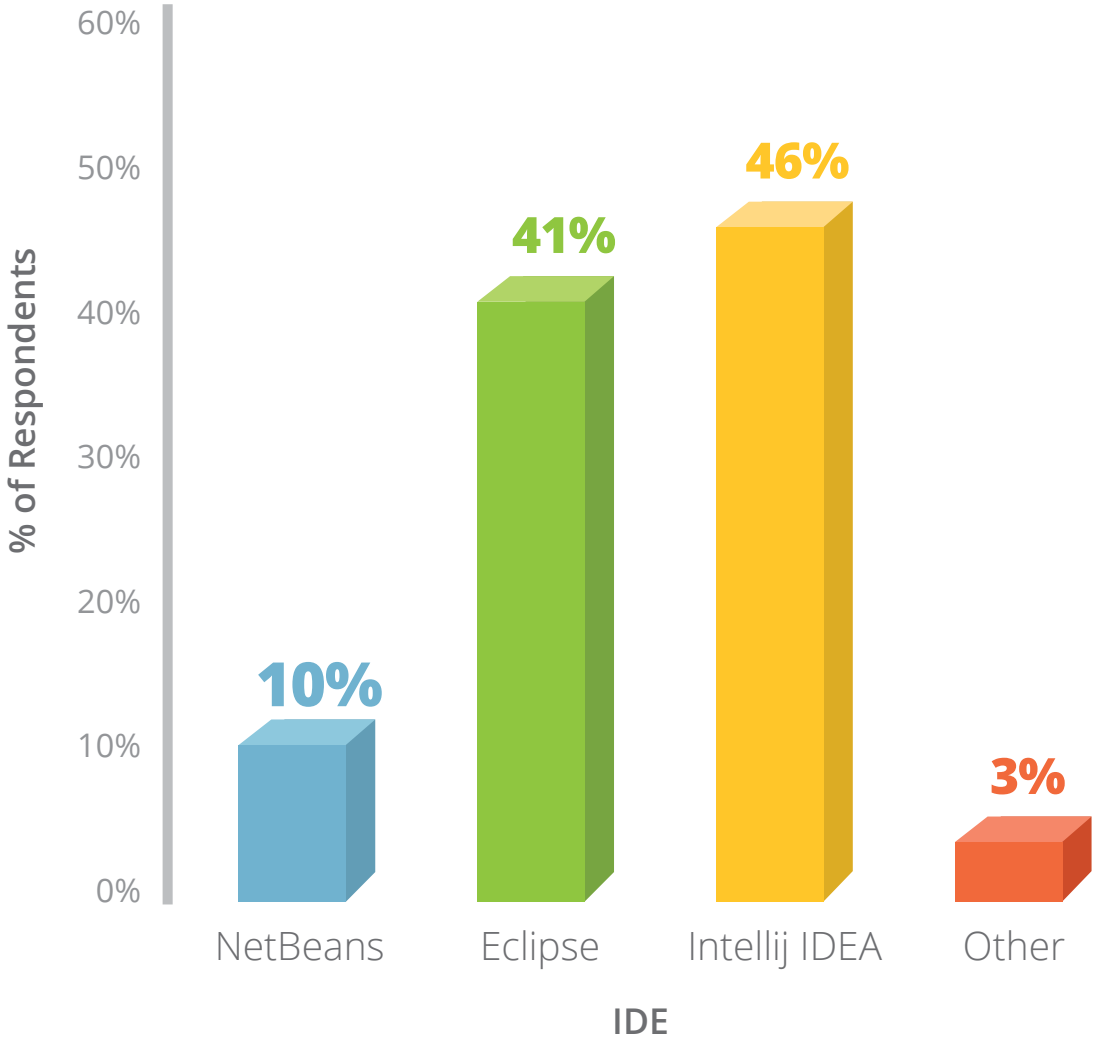


Which IDE do you use most often?

Next up is a real doozy of a question! The eternal battle between good and evil is nothing compared to the IDE flame war! This year marks a very special milestone for this question, as all previous surveys have seen Eclipse take the top spot. This year, even after combining the different Eclipse flavors, such as STS and JBoss tool suite etc, **IntelliJ is the number one IDE** among Java developers for the first time in our survey. Not only is it number one, but with 46% of the share it beats Eclipse by a clear 5% of the vote! In part 3 we'll see the trend of previous years to determine whether this result was expected or not. SPOILER: It was!

This will please JetBrains, the company behind IntelliJ IDEA, given the recent controversy over the new subscription based model that JetBrains introduced in 2015 (which is a great model by the way!). It will be interesting to see if they can hold the top spot next time we run the survey! NetBeans still has a solid 10% of the votes with 3% existing in the other category which includes hardcore devs, who use vi, vim, Emacs or a magnet, needle and steady hand, as well as JDeveloper, RAD and TextMate users (we're looking at you, @venkat_s!).

Figure 1.11 Battle of the IDEs



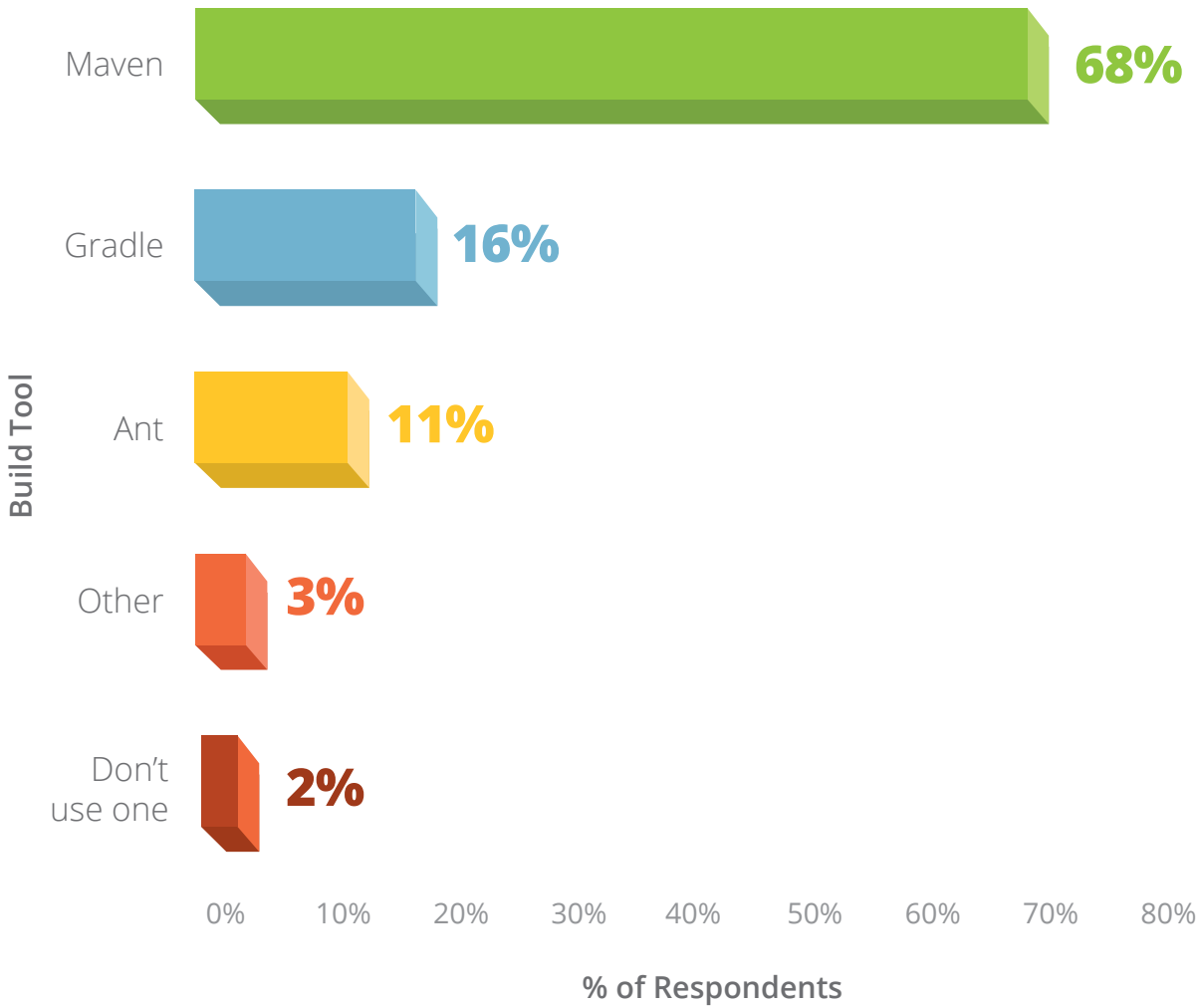
REBEL LABS by ZEROTURNAROUND

Which build tool do you use most often?

Another source of much frustration in a developer's job is their choice of build tool. Of course, unlike IDE choice, build tools are decided by the project rather than the individual developer. You wouldn't see a team of developers working on the same project using different build tools, which you quite often see with IDEs. The big question this year around build tools is whether Gradle has further eaten away at the Maven monopoly or not. Well, looking at the responses, Maven still has a very strong dominance in the build space with almost seven in ten developers (68%) downloading the internet every day using Maven. The Gradle team will be upset to see such a poor uptake of just 16% of the vote, with Ant just behind on 11%. There's always one using SBT, and that's covered in the other category along with the hardcore makefile users. #Respect.

Note that the question here wasn't multiple choice, so as we've seen with other surveys, Gradle is used by many more people, but clearly not as the primary build tool. For more information about this, be sure to check out the trends section in Part 3 of this report.

Figure 1.12 Battle of the build tools



REBELLABS by ZEROTURNAROUND

Which Application Server do you use for your main application?

Application servers — the technology we love to hate! Whether we're waiting for yet another server restart (Wat! You should be using JRebel! OK, cheap shot!), or fighting with a configuration model that could only have been dreamt up by an alien race, they often take the brunt of our bad language and frustrations. We asked a couple of questions about application server usage; one pertaining to production environments, the other about development preferences. While production environments are designed, in principle, by decision makers, a developer could always opt to use a lighter weight application server in their development environment. This of course brings other challenges, like migration issues between development and production, but for now we'll just look at what people are actually using.

We can see from *Figure 1.13* on the following page that Tomcat is once again the standout leader in application server usage with 42% of respondents using the application server in both development and production. So, almost one in two of those who state they use an application server, pick Tomcat. Everyone else takes the scraps that are left. In fact, Tomcat's biggest competitor among production applications servers is "We don't use an application server in production". That pretty much sums it up. However, I would have expected to see Tomcat's development percentage to be higher than production as I might have predicted that many using the heavyweight app servers in production would switch to Tomcat for ease of use in their local development environments.

Perhaps Jetty has taken the mantle for this as the second most popular development application server at 12%. In fact, there are 50% more respondents using Jetty in development than production. Where does this additional 50% come from? One possibility is Spring Boot users, picking Jetty as their embedded server of choice. Looking down the graph in *Figure 1.13*, we see WAS and WebLogic lose the same percentage in development. Almost half of those using WebSphere in production switch application servers in development, which should be very concerning for IBM. A quarter of those using WebLogic in production also switch to a different server in development. WildFly and JBoss EAP share similar numbers as you'd expect with a minor percentage exchange where you'd expect those developing for a JBoss EAP production environment are using WildFly in development. Not really much of a migration as they're essentially the same codebase. It's nice to see GlassFish at 4% particularly after they received the cold shoulder treatment from Oracle. Perhaps this server has remained significant due to the support of the team at Payara — great job and community support, folks!

Figure 1.13 Application Server Usage in Production and Development

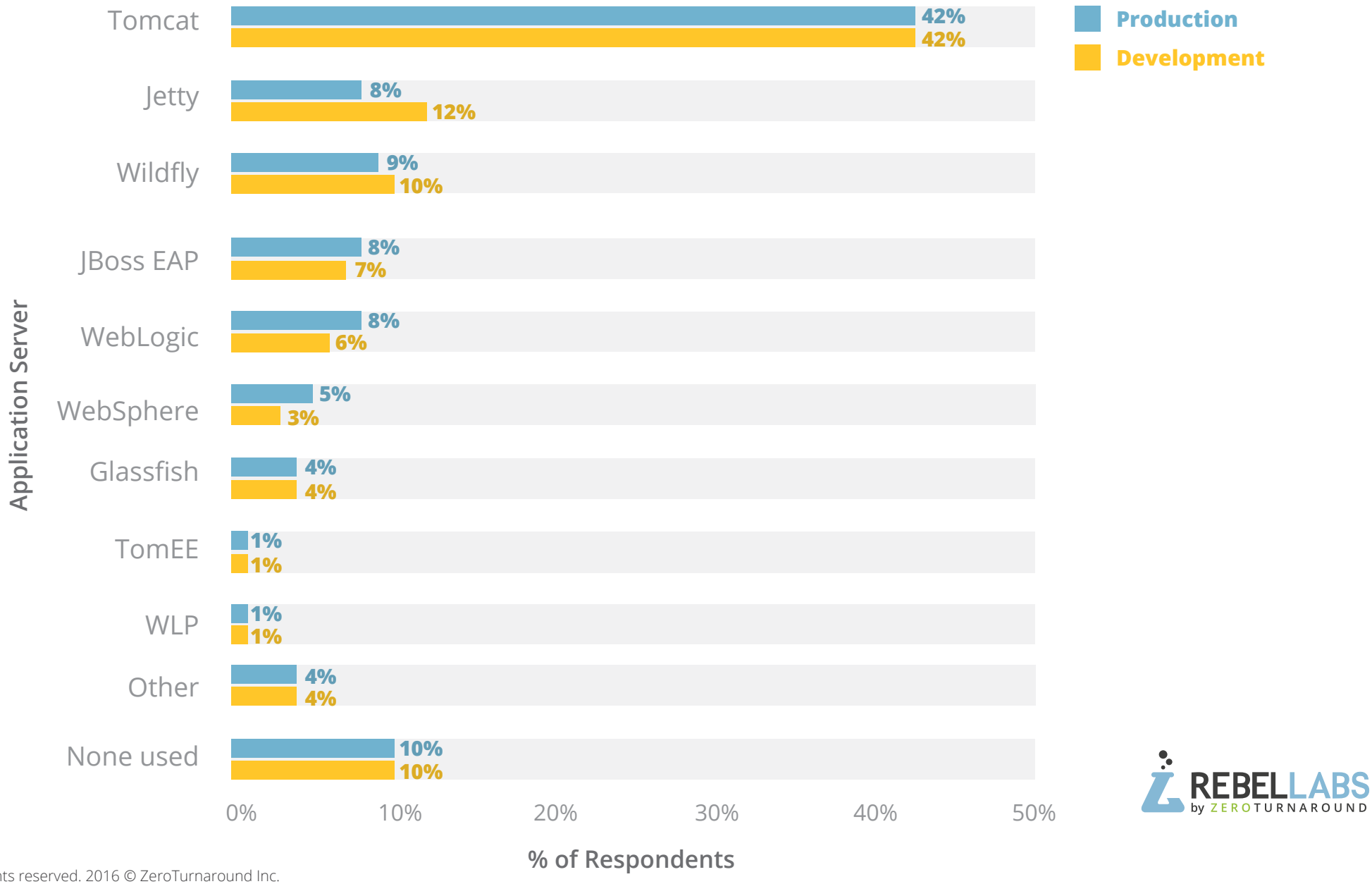
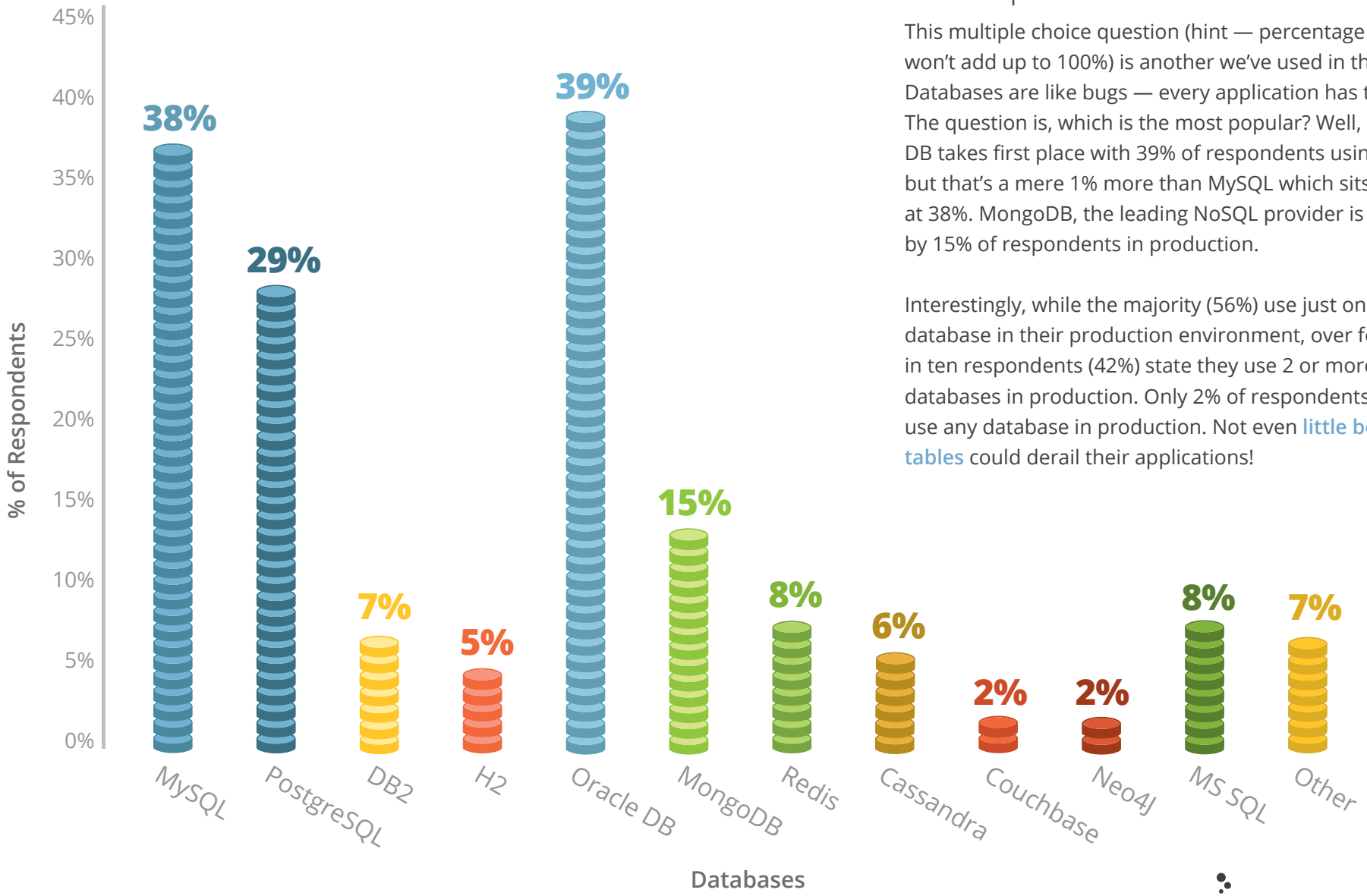


Figure 1.14 Databases used in production



Which databases do you use in production?

This multiple choice question (hint — percentage values won't add up to 100%) is another we've used in the past. Databases are like bugs — every application has them! The question is, which is the most popular? Well, Oracle DB takes first place with 39% of respondents using it, but that's a mere 1% more than MySQL which sits pretty at 38%. MongoDB, the leading NoSQL provider is used by 15% of respondents in production.

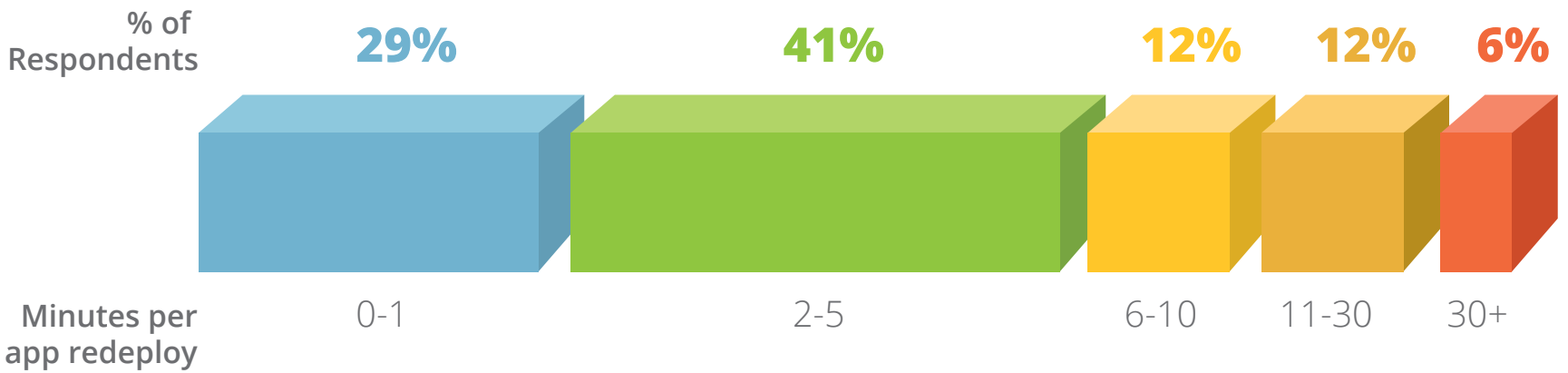
Interestingly, while the majority (56%) use just one database in their production environment, over four in ten respondents (42%) state they use 2 or more databases in production. Only 2% of respondents don't use any database in production. Not even little bobby tables could derail their applications!

How long does it take you to compile, package and redeploy your application once you've made a code change,
so that you can actually see your change at runtime? (in minutes)

As the company behind JRebel, a tool that eliminates time spent during compile, package and redeploy stages, we're always interested in understanding how long it takes for developers to go from changing code in their IDE to seeing their code changes in a live runtime. We typically ask a question about how long a redeploy takes. But a more realistic question is to ask how long it takes a developer to go from making a change in their IDE to seeing their change in their runtime. This includes compilation, build, redeploy or server restart, state creation, including logging back into your app and navigating to the page with your changed code.

We can see varying results from *Figure 1.15*, and this is because different environmental factors will have contributing factors to this overall number, including application size, application state, build process, application server and so on. The majority sit in the 2-5 minute bracket, which is unacceptable for any developer to suffer through. In fact, over 70% of respondents were 2 minutes or over, with the median being 3 minutes and the mean an incredible 8.7 minutes. 18% of developers, that's almost one in five, have to wait over 10 minutes to see their code changes in a live runtime. For ideas of how you and your team can eliminate this wasted time and stay sane, visit http://jrebel.com.

Figure 1.15 Wasted development time spent on building and redeploying



REBEL LABS by ZEROTURNAROUND

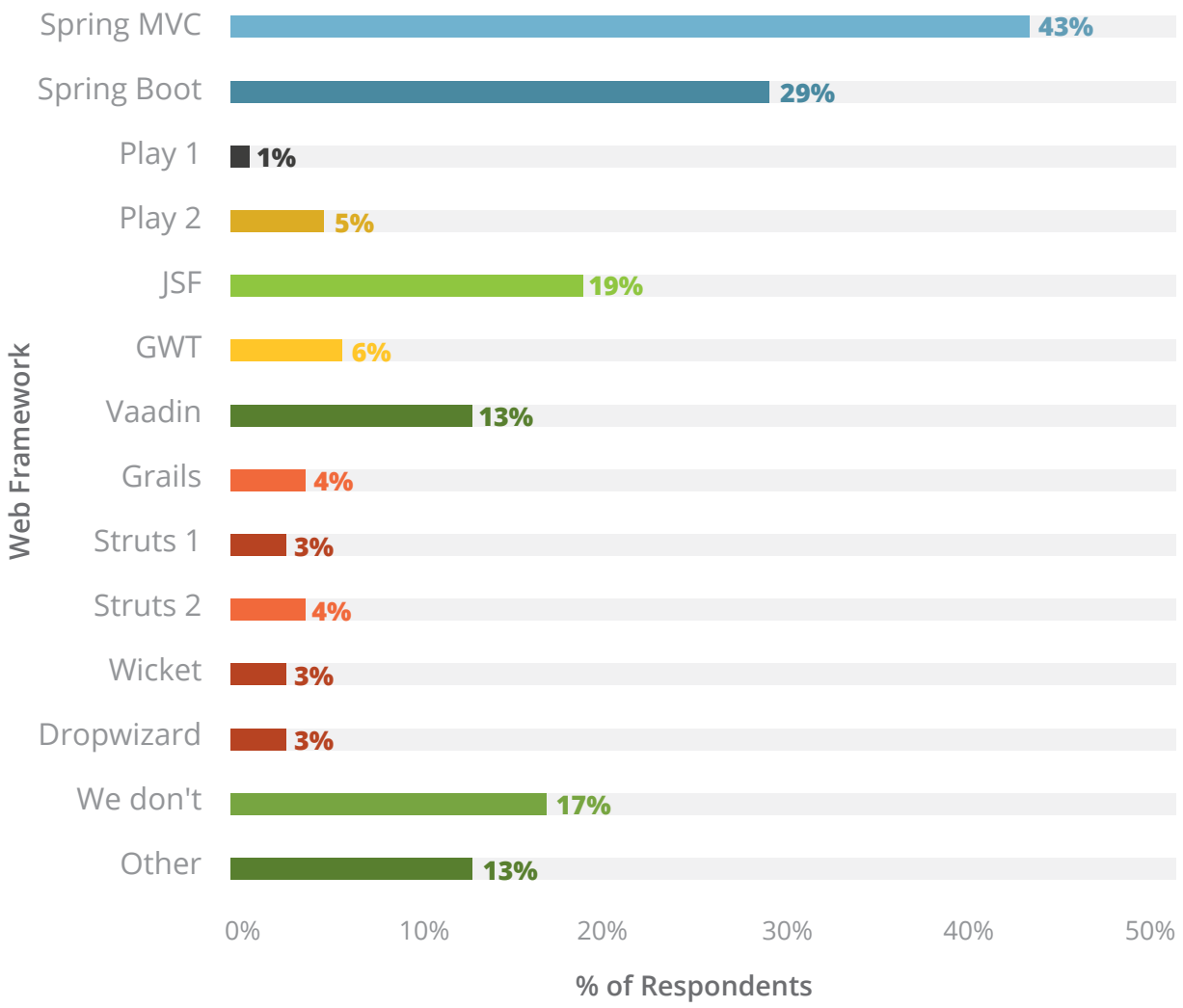
Which Web Framework(s) do you use?

Another topic that developers love to argue over is web frameworks. Which one is best overall? Which is more suitable for their needs? And even, which they need to learn to be best placed for their next job! Now, before you email in, note that this question could be answered with multiple selections, so numbers won't add up to 100, number-nerds.

We can clearly see how dominant Spring has remained in 2016, with 43% of respondents stating they use Spring MVC. Spring Boot was also extremely strong (29%), particularly as a technology that was only created in 2014. This must be very pleasing for the teams at Pivotal. The closest competitor to Spring technologies is JSF at 19%, followed by Vaadin with 13%. Other things to note is the drop-off between Play 2 and Play 1 usage being quite significant, as users clearly choosing to migrate up to Play 2.

Typically, applications use just one web framework. In fact, from our responses, the median number of web frameworks used is 1 with a mean of 1.4. This takes into account all responses, also factoring in all those who said they don't use web frameworks as well as those who do. If we remove the responses for who don't use web frameworks (17%), we're left with 41% of respondents who use one web framework and 42% of respondents who use two or more web frameworks. Therefore, if you're going to take advantage of web frameworks, you'll likely end up using more than just one in your application.

Figure 1.16 War of the web frameworks



Which Continuous Integration Server do you use?

As one might expect from a survey that asks which CI servers people use, Jenkins is the runaway winner. The 10-year-old CI server is used by almost two in every three (60%) respondents. The biggest competition by votes is ironically the "We don't do CI" category, which makes Jenkins' dominance even more impressive. Bamboo is the next most popular CI server with 9% of the votes, with TeamCity, Hudson and Travis CI taking up the rest of the share. In the 'other' category, Circle CI is certainly a server worth mentioning that had a good portion of the remaining votes.

Before we go, we should mention Hudson, the CI server from which Jenkins was forked back in 2011, with overwhelming support from the community. Note that Hudson was transferred from Oracle to the Eclipse foundation in 2012. Since Jenkins owns the market in the CI space and with 20 (twenty) times the number of users (using our sample data) compared to Hudson, it's a mystery why Hudson is still being developed or supported today. The game has long been won by Jenkins and perhaps it's time to unplug Hudson altogether.

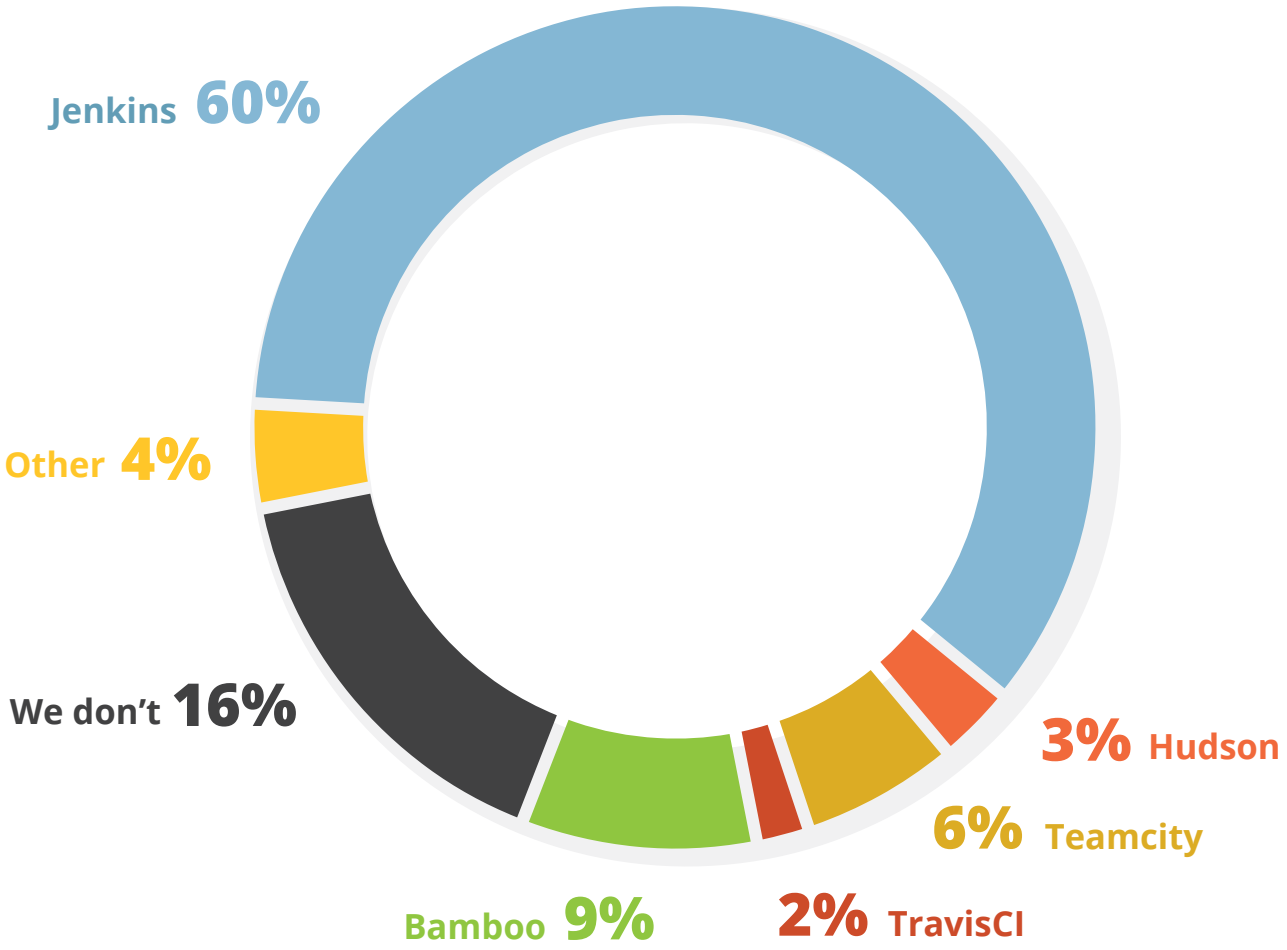
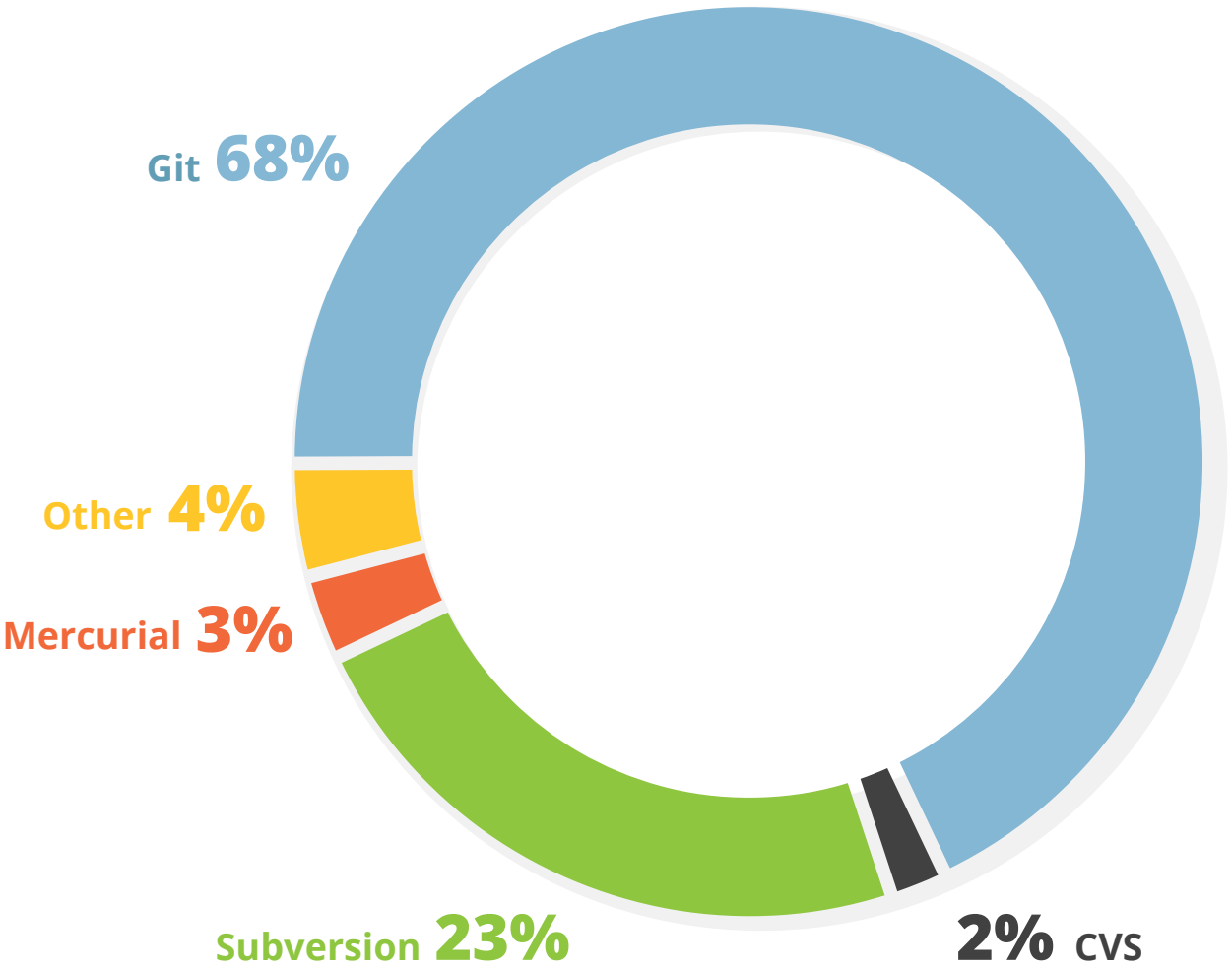


Figure 1.17 Continuous Integration Server Usage

Figure 1.18 Most Commonly Used VCS



Which VCS do you use most often?

It's time for you to cast your eyes upon another technology monopoly! This time, we're talking about VCS tools. Needless to say, saving our files in a directory using a format like: [filname]_[version] is the most popular solution... Oh no, that's incorrect! Git is actually the most popular solution, and by a wide margin! Almost 7 in 10 respondents state they use Git most often, seeing the infamous subversion (SVN) fall back quite a distance behind on 23%. Mercurial and CVS deserve having their names mentioned, taking 3% and 2% of the vote respectively, with many more tools being collected in the 'other' category.

If anyone was in doubt or asking which VCS system won the fight, you're a bit late to the party. Git is most definitely the last VCS standing!

REBEL LABS by ZEROTURNAROUND

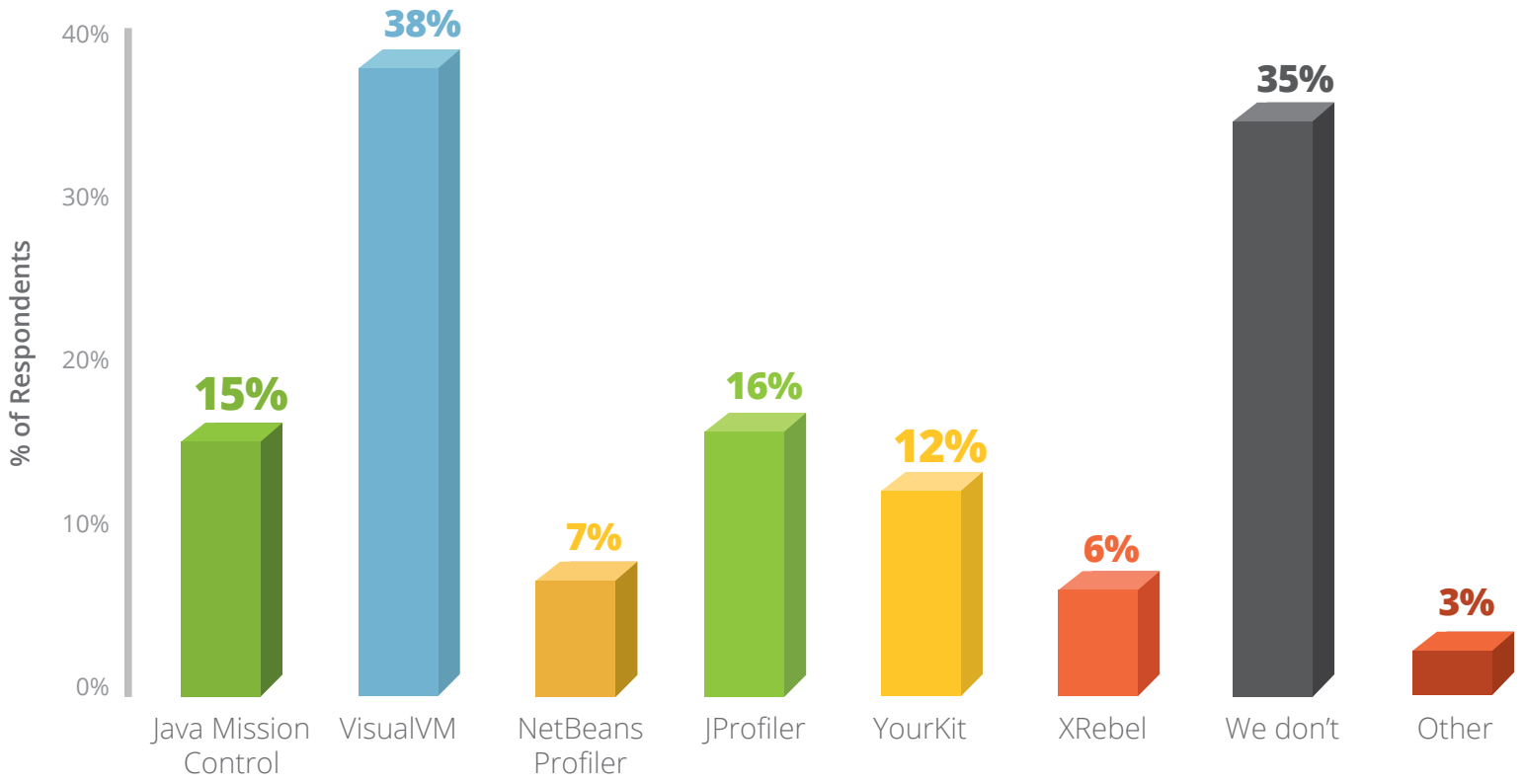
Which Java Profilers do you use?

VisualVM tops this category with 38% of the votes, but sadly 35% of respondents claim they don't use profilers at all. This is unfortunately a typical representation of how performance testing takes a back seat, particularly with developers. Years ago, the same could be said about quality testing during development time.

With the introduction of developer focused profilers like XRebel, with 6% of the votes, we see a glimmer of hope for early application profiling that can be performed as developers write their code.

JProfiler is another popular profiler, shown by the 16% share it got from respondents. It's very pleasing to see 15% of respondents using Java Mission Control. As a tool that's provided with the Oracle JDK and is free for use by developers on non-production data, it's a key tool to have in any performance tester's utility belt. YourKit and NetBeans profiler also perform well with 12% and 7% of the votes respectively.

Figure 1.19
Java Profiler Usage

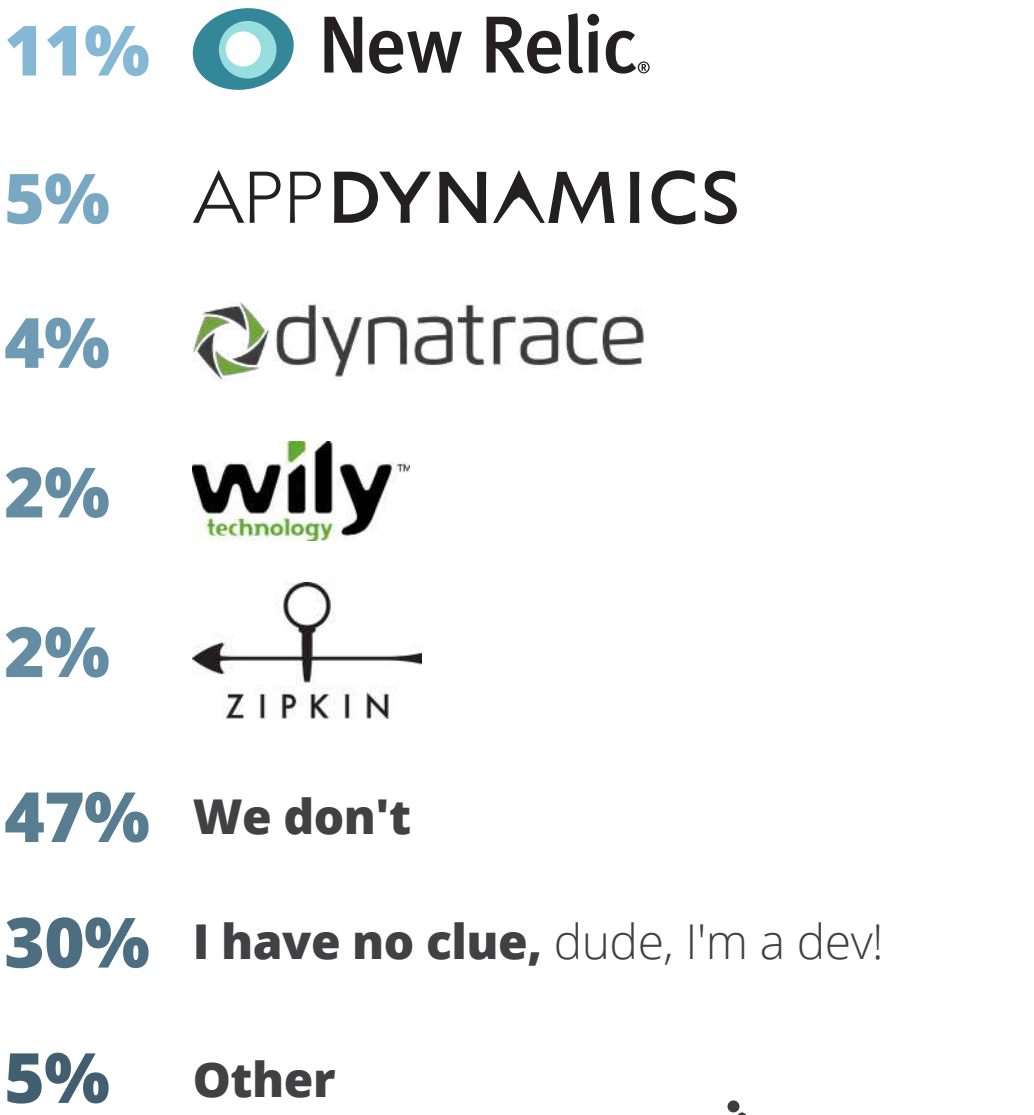


REBEL LABS by ZEROTURNAROUND

Which APMs do you use?

Hang on, APMs? That's not a development thing — perhaps that's why almost one in three respondents had no clue which APMs are used to monitor their applications. This is even worse considering 47% of respondents state they don't use APMs, or perhaps they do but just don't know that they do! As a result, the most popular APM at just 11% of the votes is New Relic. In fact, it's more than twice as popular as the next APM, AppDynamics, with DynaTrace following closely behind with 4%

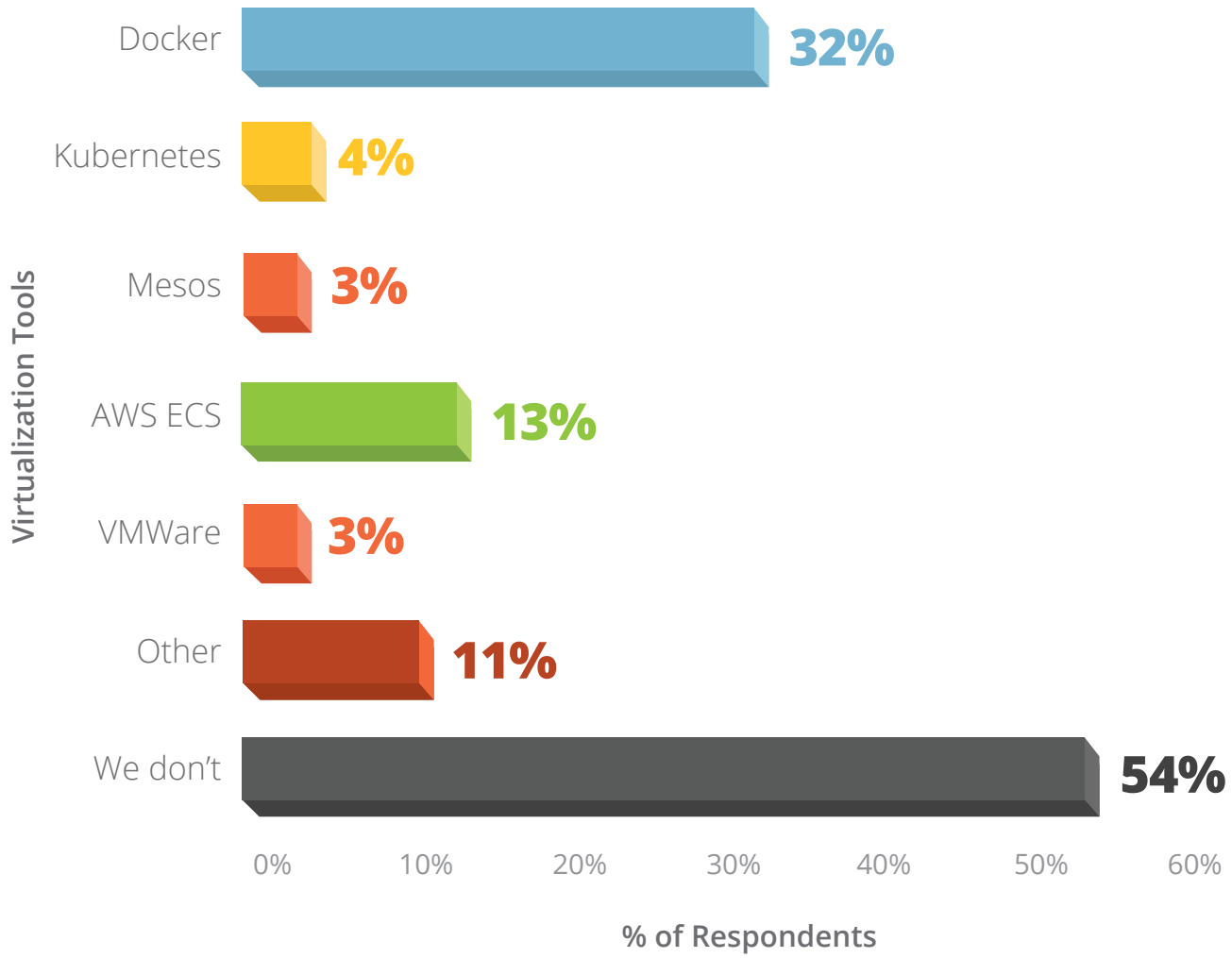
Figure 1.20 APM breakdown



Which Virtualization Environments and tools do you use?

As you would expect, the Docker hype manifests itself into the highest share in our survey among people who use virtualization environments, taking 32% of the votes. AWS ECS lags behind on 13% and there are few others posing any kind of competition beyond this. Interestingly, only 4% state they use Kubernetes, which is surprising given the amount of great press and conference sessions it gets. Perhaps we'll see increased adoption over the coming years. Virtualization is still a practice performed by the minority, with 54% of respondents stating they don't use virtualization environments at all. I expect this will switch in the near future, particularly if Docker's strong growth continues.

Figure 1.21 Virtualisation Environments and Tools in Use

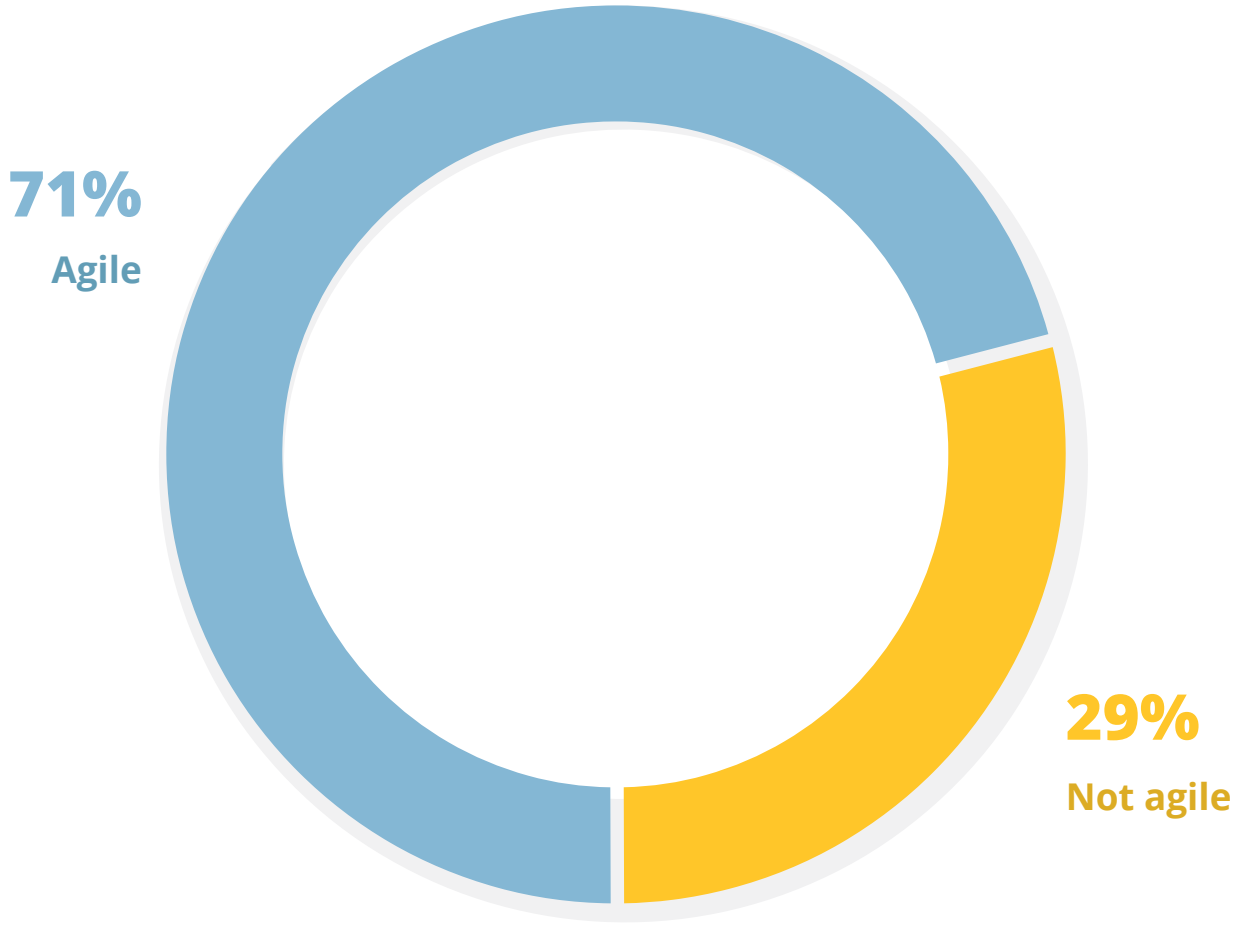


REBELLABS by ZEROTURNAROUND

Is your team agile?

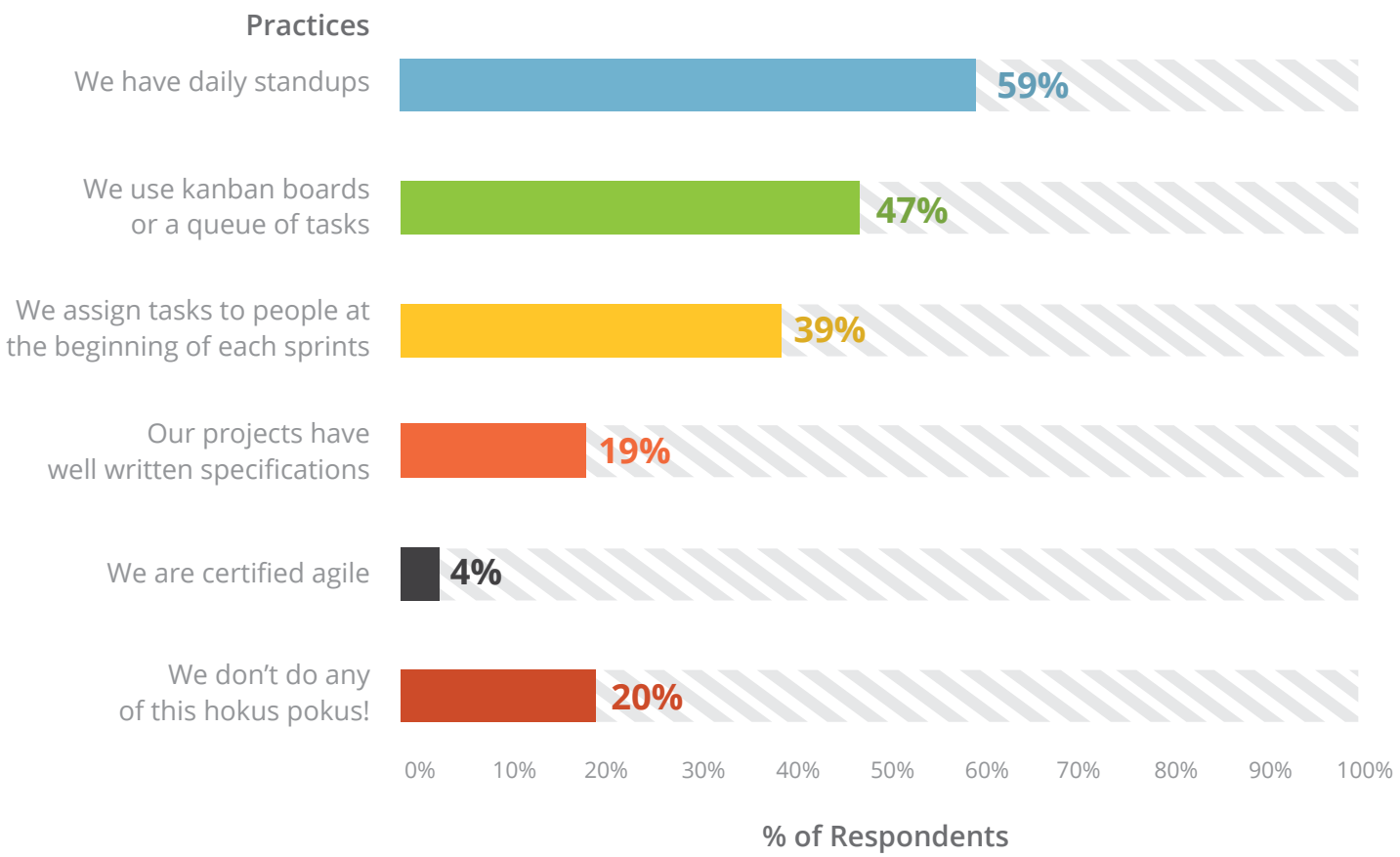
The agile party mark a landslide victory, with 71% of respondents stating their teams are agile. This doesn't mean they are agile, of course. Rather, it means it's the individual's belief their team works in an agile way. Next, we'll look at which practices our respondents follow, and be sure to check out Part 2 when we break it down further to see which practices our agile developers follow! The results are popcorn worthy.

Figure 1.22 To Agile or Not To Agile?



REBEL LABS by ZERO TURNAROUND

Figure 1.23 Adoption of Agile Practices



Do you practice the following processes?

With only 4% of respondents working in a certified agile team, this likely says more about the number of teams who care about being certified agile than the practices most teams follow. One might argue that what works for one team is very different to another, so working in a prescribed way could be counter-intuitive for a team. Almost two in three respondents (59%) have daily standups — which is reasonable — with almost half of respondents (47%) using Kanban boards to manage their tasks. One in five of our respondents (20%) don't care too much for any of these practices, stating they don't follow any of them whatsoever. 39% of respondents state they assign tasks to their team at the beginning of a sprint which I would have expected to have been higher, given it's a core part of agility. Only one in five respondents (19%) claim they have well written specifications for their projects, so not much love for documentation there.

Does your project have formal requirements for the following areas?

Almost two in three respondents have formal functional requirements in their project, which used to be an anti-pattern of agile that encouraged people to ignore documentation altogether. I think this is a pretty outdated view of what it means to be agile and this is backed up with almost two thirds of respondents producing formal documentation. We can see everything else takes a lower priority to function, with security, rather surprisingly, next most popular with one in three stating they have formal requirements in place for their project. Performance is next with 29% of people caring enough to enforce formal requirements for their application. The same number (29%) don't care about any of these formalities, claiming they do not have formal requirements for any of these areas.

Figure 1.24 Use of Formal Requirements by Project Area

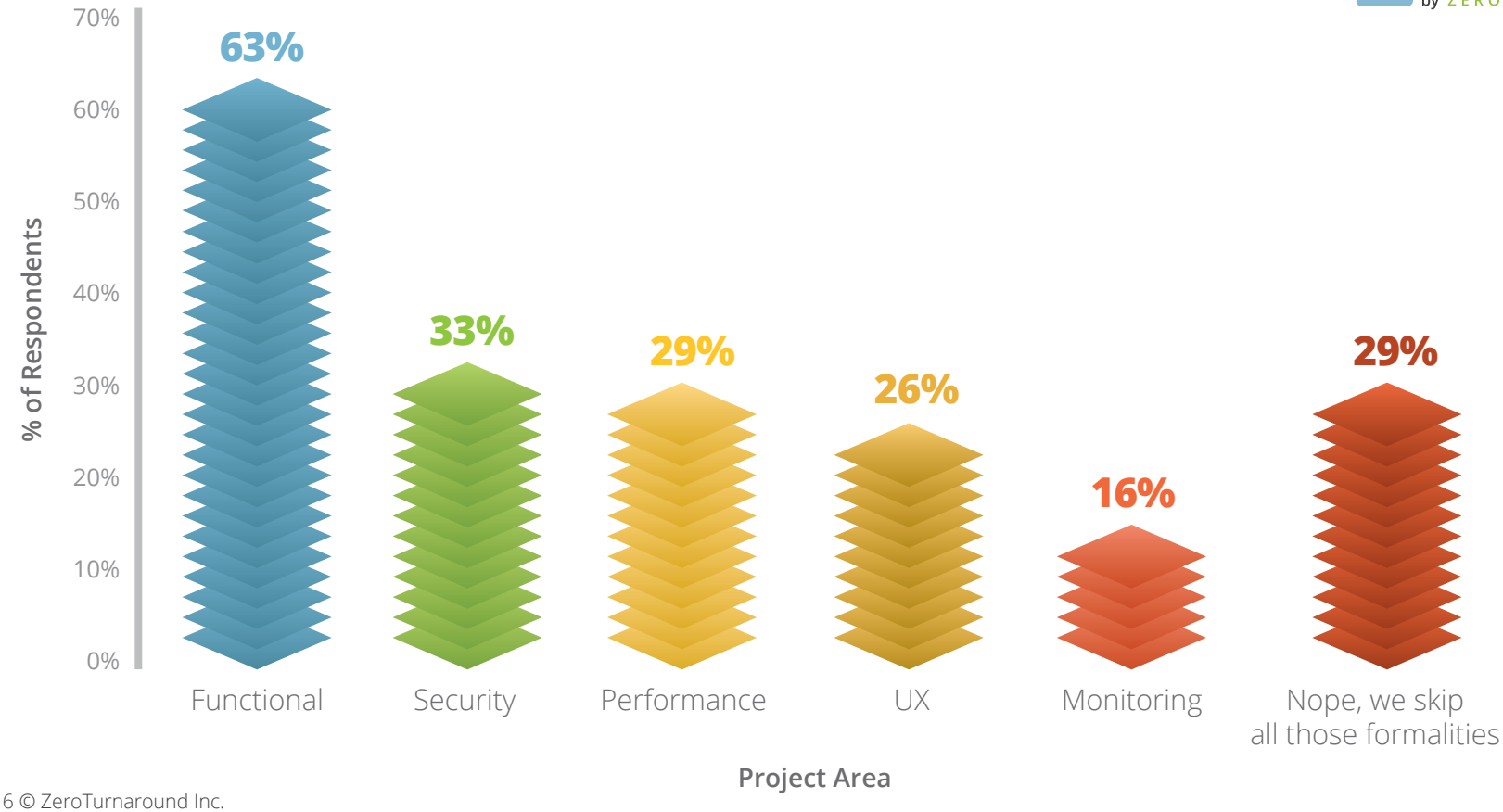
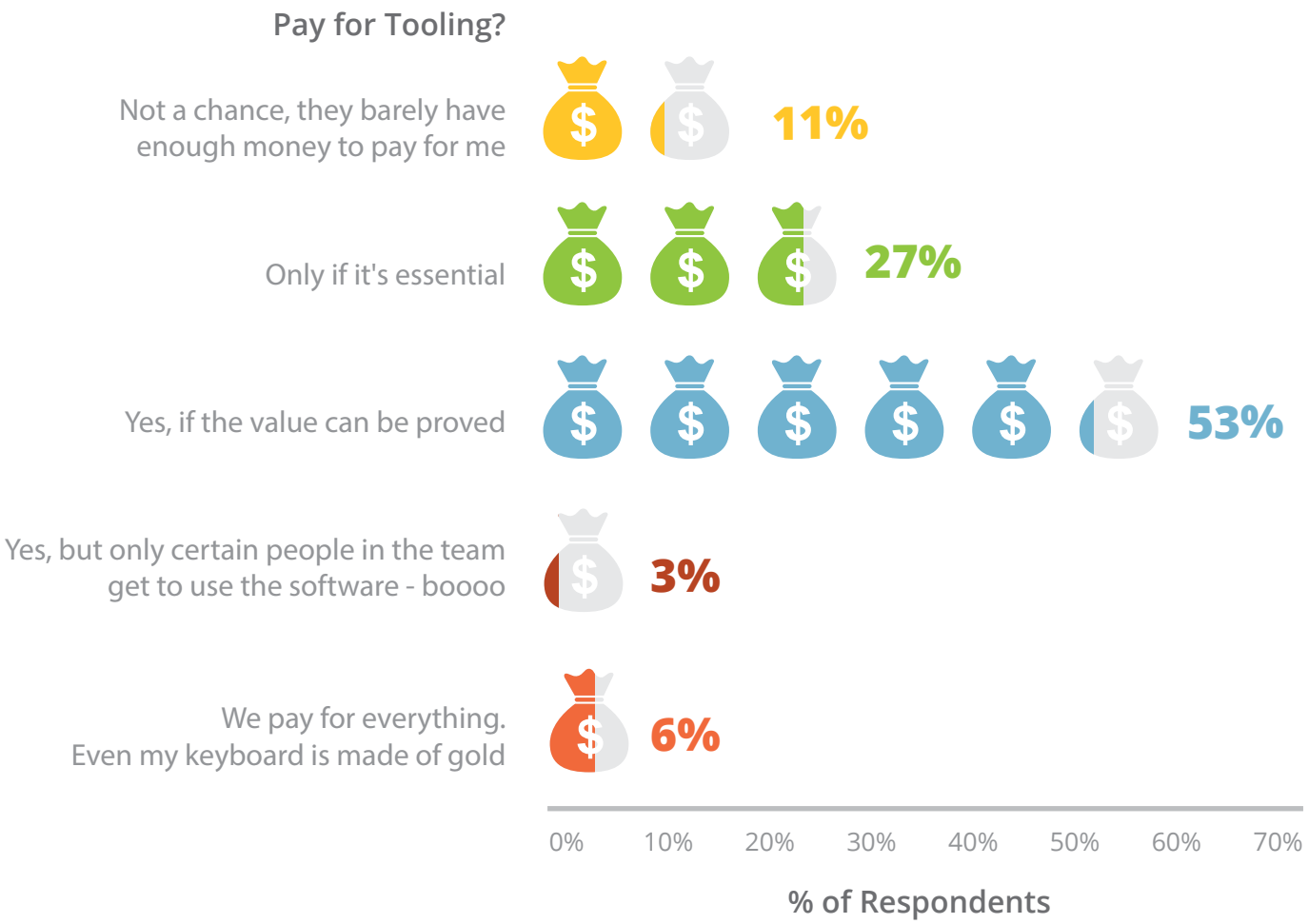


Figure 1.25 I Can Haz Development Toolz?



Are you (or your company) likely to pay for development tooling?

Good tooling invariably comes at a cost. It's important for companies to understand that and empower their developers with the right tools they need to do the job. You wouldn't expect a chef or a builder or in fact any trade to do a job without giving them the right tools to accomplish that task well, but for some reason in software development it's often accepted. First, let's spend a minute thinking about the poor one in every ten developers (11%) whose company will plain not pay for any tooling whatsoever. Done that? Good. It's good to see the remaining nine out of ten respondents at least standing a chance of purchasing the tools they need. In fact, 6% of them will pretty much buy anything, while the vast majority, four out of five, will be able to purchase tools if they are essential or if their value can be proved.

REBELLABS by ZEROTURNAROUND

PART 2:
DATA PIVOTING

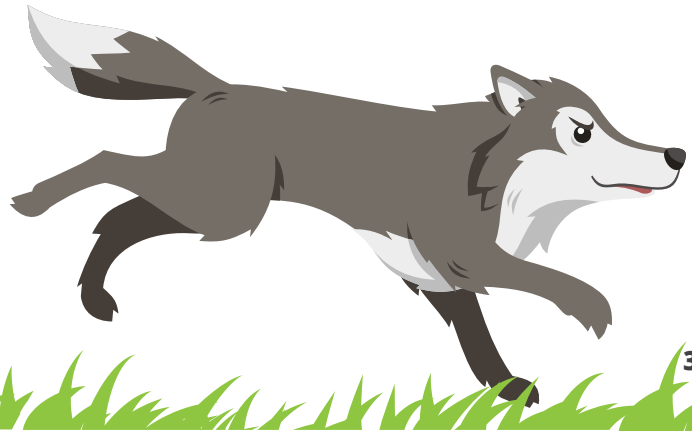
We’re finally in part 2 of the report. You might choose to call this the double money round, but people might consider you a little odd. In this part of the report we’ll pivot on our data to see if we can find trends.

What will we look at here?

We'll start by looking at the differences in behavior between those who think they're early adopters of technology (you know the ones without any money because they just bought a new Apple iPhone 8S plus, with 5GB of memory and a 50MP camera, running on iOS 10), versus those who are technology sheep. After that, we'll analyze trends among those who are using a microservices architecture to see which tools and technologies they're using. We'll also see if those who call themselves agile are indeed agile. First, let's look at when people adopt different technologies.

Early Adopters versus Technology Sheep

This is a really interesting comparison as it tells us a lot about the state of a particular technology. For example, if we see a lot of early adopters looking at a technology, we might assume that it's an up and coming technology that people are beginning to adopt. Or perhaps we can look at how people are adopting versions. If technology sheep are adopting the latest version, we might assume the latest version is well established.



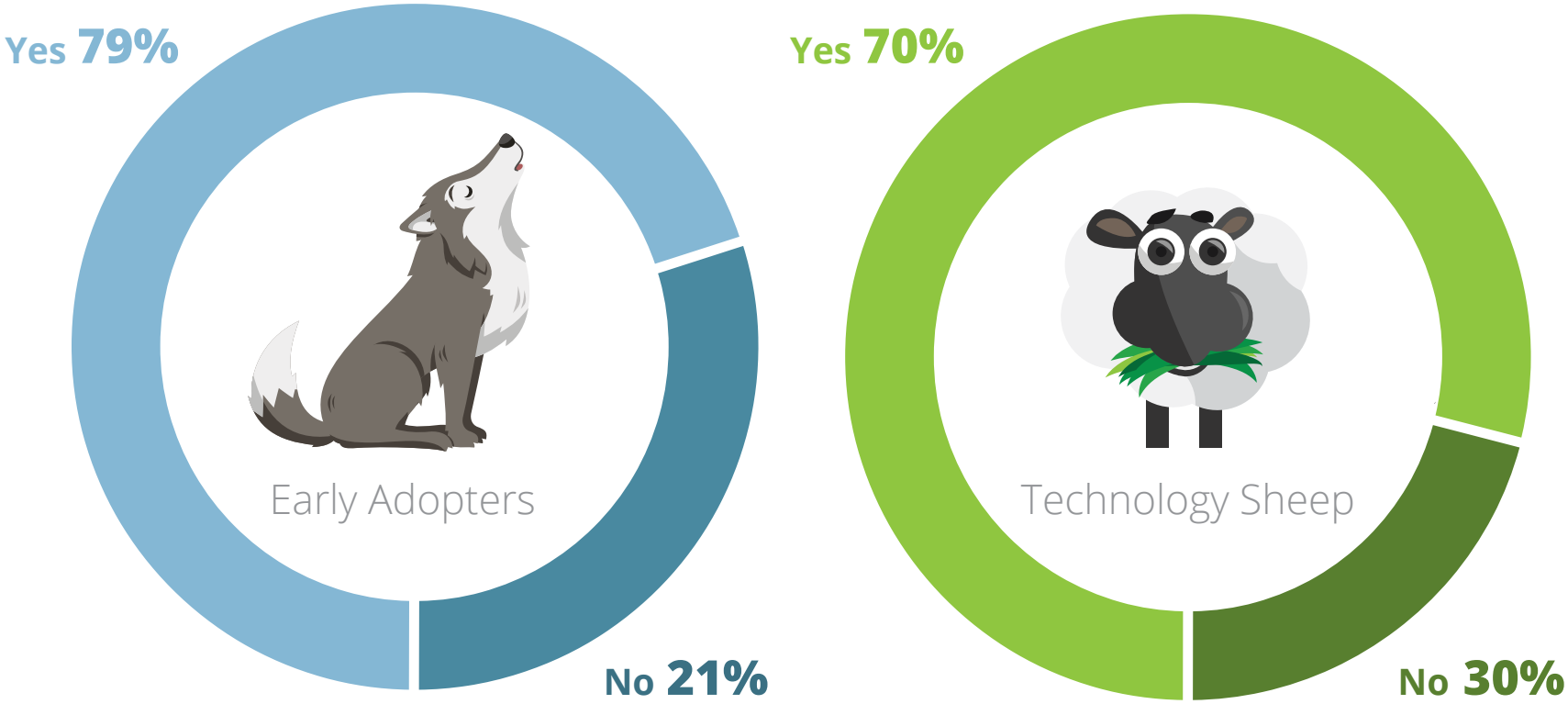
Do early adopters think they're better?

Put frankly, yes, yes they do. Eight in ten early adopters (79%) think they're better than the average in their role, whereas seven in ten (70%) technology sheep think they're better than average in their role.

Therefore, you're more likely to think you're better than the average person in your role if you're an early adopter of technology. This doesn't mean they are, of course, but the mindset is quite different.

In *Figure 2.1*, we see the technology sheep and umm, early adopter wolf. Well we had to pick an animal and the wolf seemed the most opposite to a sheep!

Figure 2.1 Early adoption makes people think they're better?



Are early adopters more experienced?

One might think that you could consider adopting a technology early if you possess the experience to know which technologies were right to adopt, both in terms of the technology and timing. Another train of thought could be to wait for new technologies to survive the early hype period and only adopting once it is an established technology. This could potentially save a lot of wasted time on a technology that might not make the cut.

Looking at the graph in *Figure 2.2*, we see fewer early adopters than technology sheep around the 0-14 years bands and more early adopters with 15+ years of experience. In fact, the mean average shows a difference of almost two years, between early adopters and technology sheep, at 12.8 years and 11.0 years experience respectively. The median average also shows a two year difference, with early adopters having a median 12 years of experience and technology sheep having a median 10 years of experience.

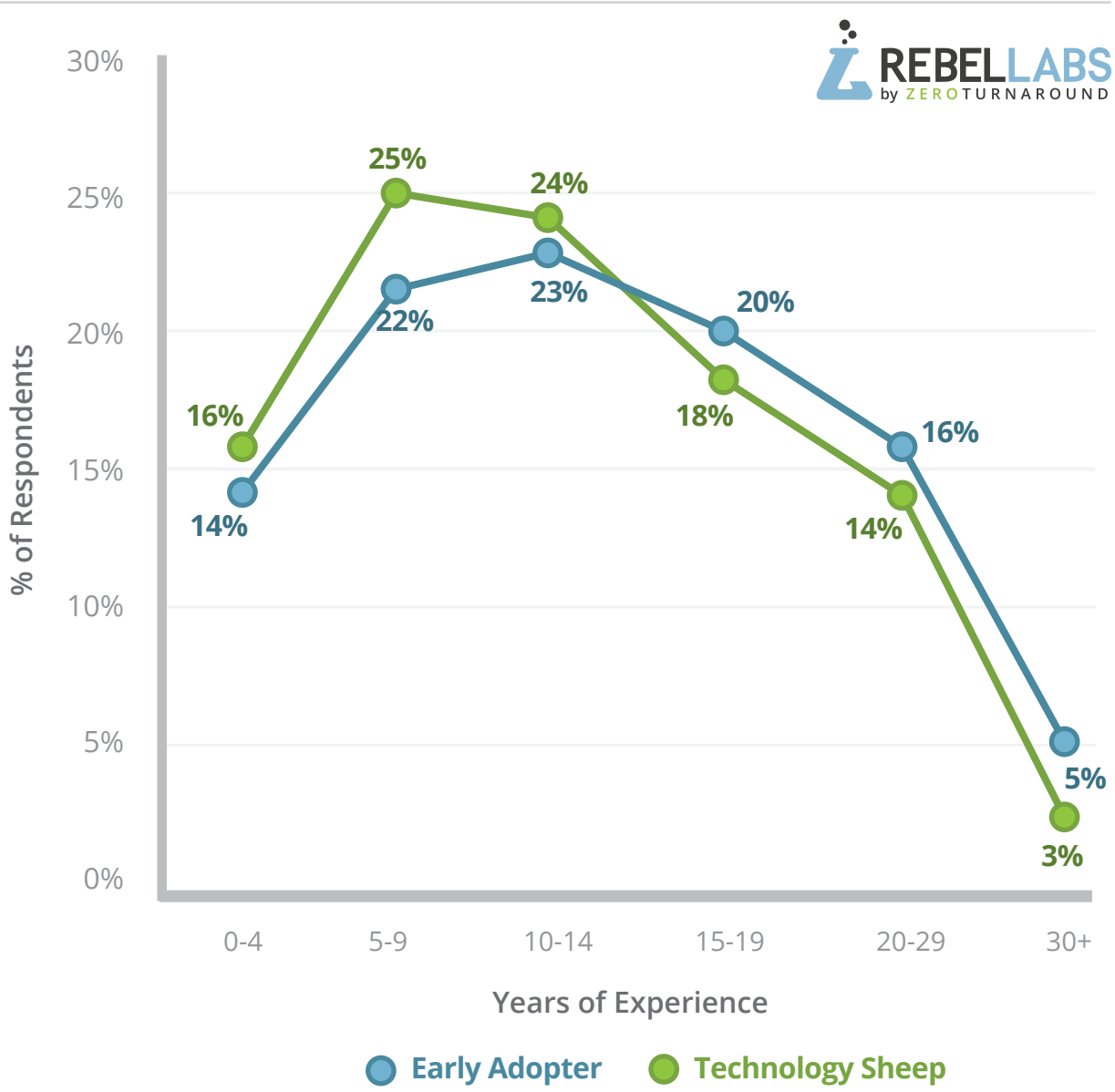
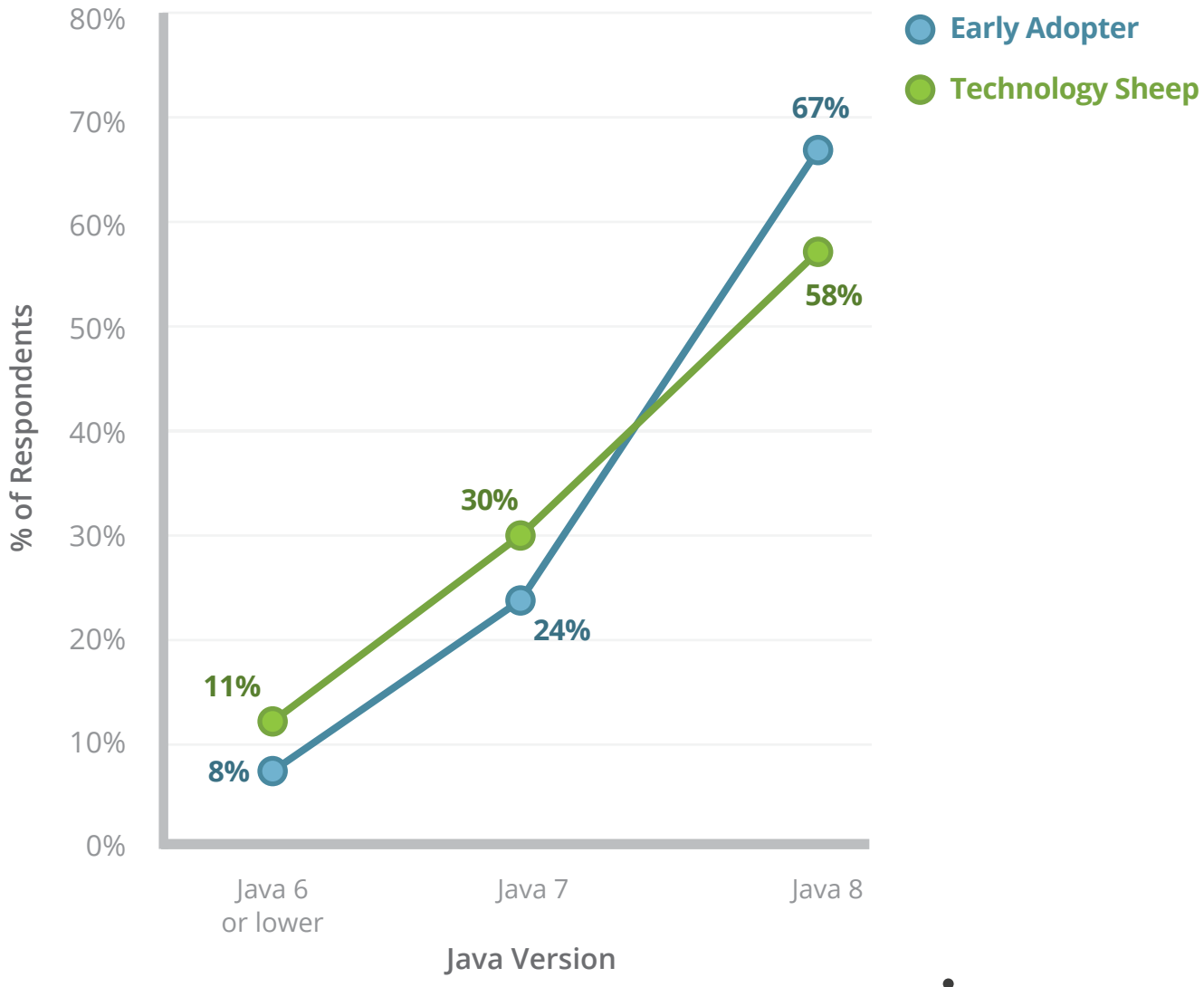


Figure 2.2 Experience Changes Your View of Adoption

Figure 2.3 Java SE Version Adoption



Java SE Version Adoption

As one might expect, an early adopter... Well, adopts stuff early, and this should be apparent if we were to look at the versions that early adopters use. *Figure 2.3* shows just this, with two in three early adopters (67%) using the current Java 8 version. Almost one in four early adopters (24%) are on Java 7 and 8% of our so-called "early adopters" are using Java 6 or older. When we look at the technology sheep, we see more of them using the older versions of Java, but reassuringly the Java 8 adoption is still very high, at 58%. This is significant because technology sheep adopting a new version would suggest maturity and stability in that version. It might suggest the path has been well walked by others and it's a proven working, viable version.

The wily among you will notice that each line only adds up to 99%. Well done! 1% of early adopters already use Java 9 early access builds. That's pretty impressive! 1% of technology sheep don't use Java at all.

REBELLABS by ZEROTURNAROUND

Java EE Version Adoption

When we look at the results for Java EE adoption we see something quite different. In fact, we see the opposite to that which we just saw in our Java SE adoption. For both early adopters and technology sheep we do still see the latest Java EE version to be the most popular among Java EE users. However, when we compare the two sets of results, we see unusual results. A higher proportion of technology sheep are on the latest version of Java EE compared to early adopters. But that doesn't make sense. Surely early adopters should be the first to use the latest version. Well, actually when we look at the numbers for those who do not use Java EE, we notice 46% of early adopters don't use Java EE, compared to 38% of technology sheep. This means that early adopters are migrating away from the Java EE platform, which isn't the kind of news that the Java EE community want to hear. Amusingly, almost one in ten (8%) "early adopters" still use J2EE.

Figure 2.4 Java EE Version Adoption

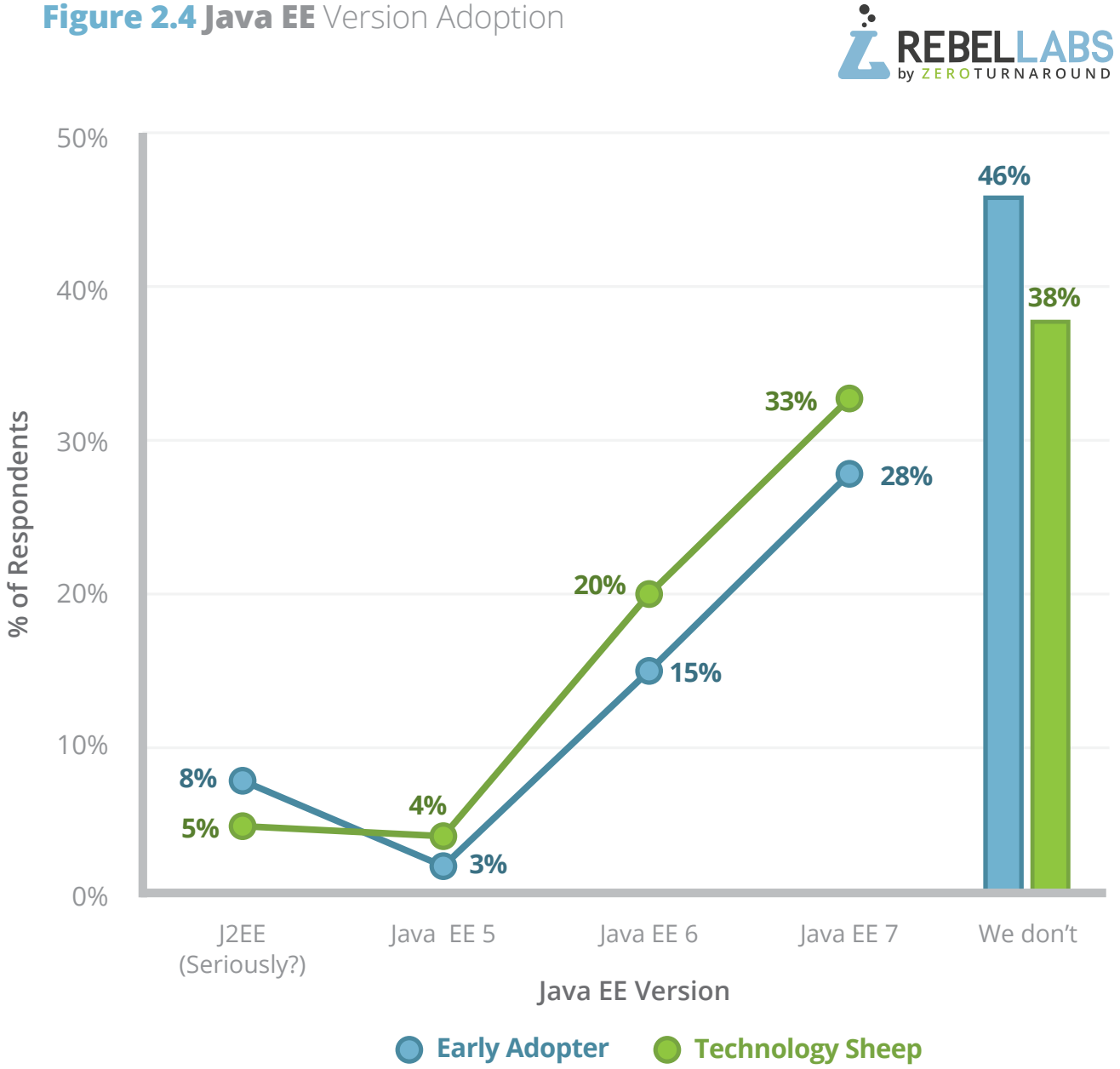
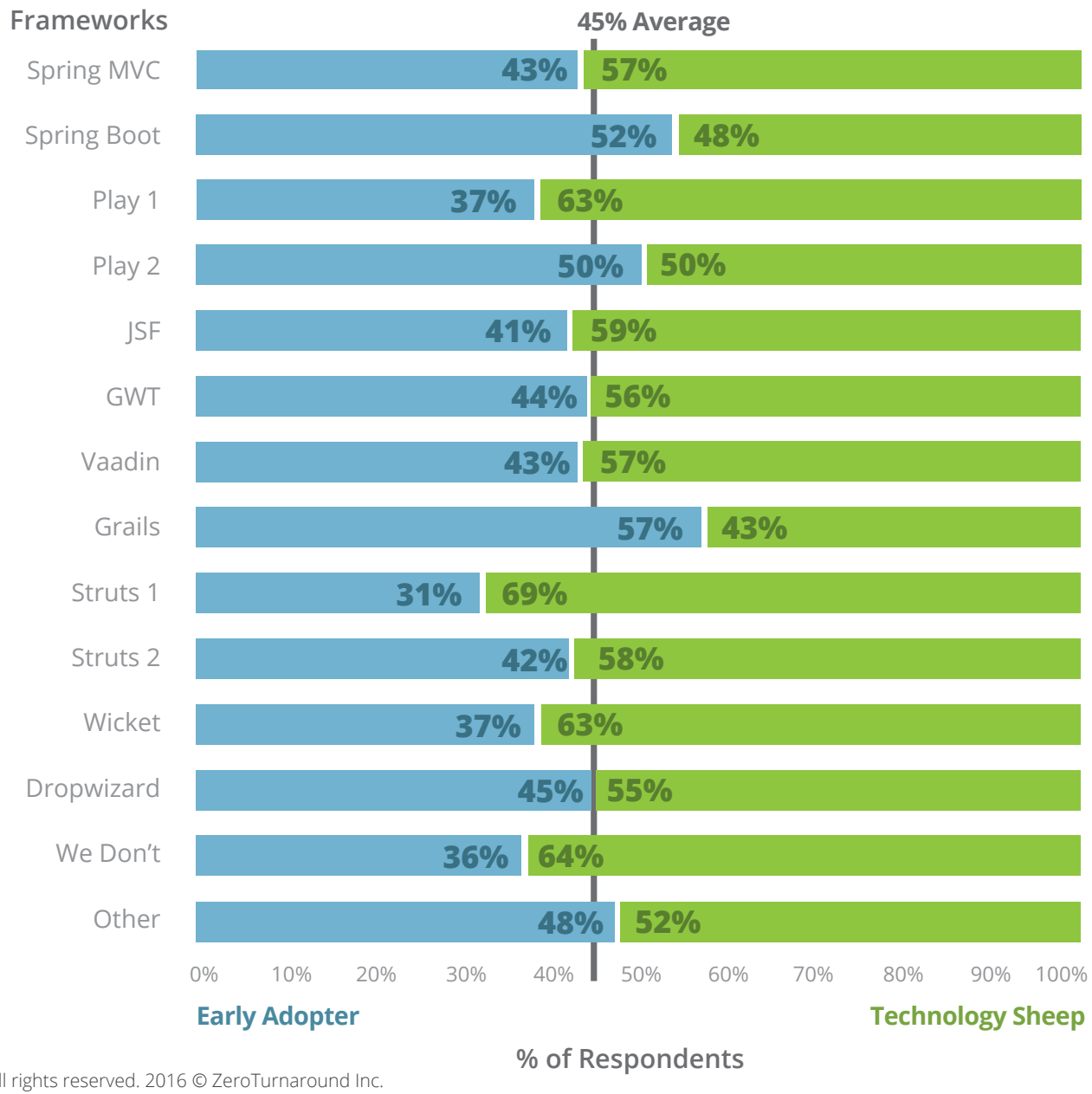


Figure 2.5 Web Framework Usage broken down by Adopter types



Web Frameworks

Now let's check out what percentage of early adopters use particular web frameworks compared to technology sheep. This will allow us to work out which frameworks are gaining adoption, which are established and which are on the way out. The early adopter/technology sheep split across all respondents is 45:55. *Figure 2.5* shows this ratio as a gray line so that we can see when results are significant.

Let's first focus on those web frameworks that are close to the 45:55 average ratio. Those that are just below this average ratio (have fewer early adopters to technology sheep than average) are Spring MVC, JSF, GWT, Vaadin and Struts 2. Dropwizard sits bang on the 45:55 ratio line. Those web frameworks that are further away from the average ratio, and perhaps statistically more important, include Spring Boot, Play 2 and Grails which all have a higher proportion of early adopters than average. These can be considered upcoming technologies. Certainly this could be argued with Spring Boot which has seen a lot of uptake over the last couple of years. Conversely, Play 1, Struts 1 and Wicket are three technologies which have a higher than average proportion of technology sheep. This suggests that these frameworks are on the decline, which looking at the list, is arguably the case.

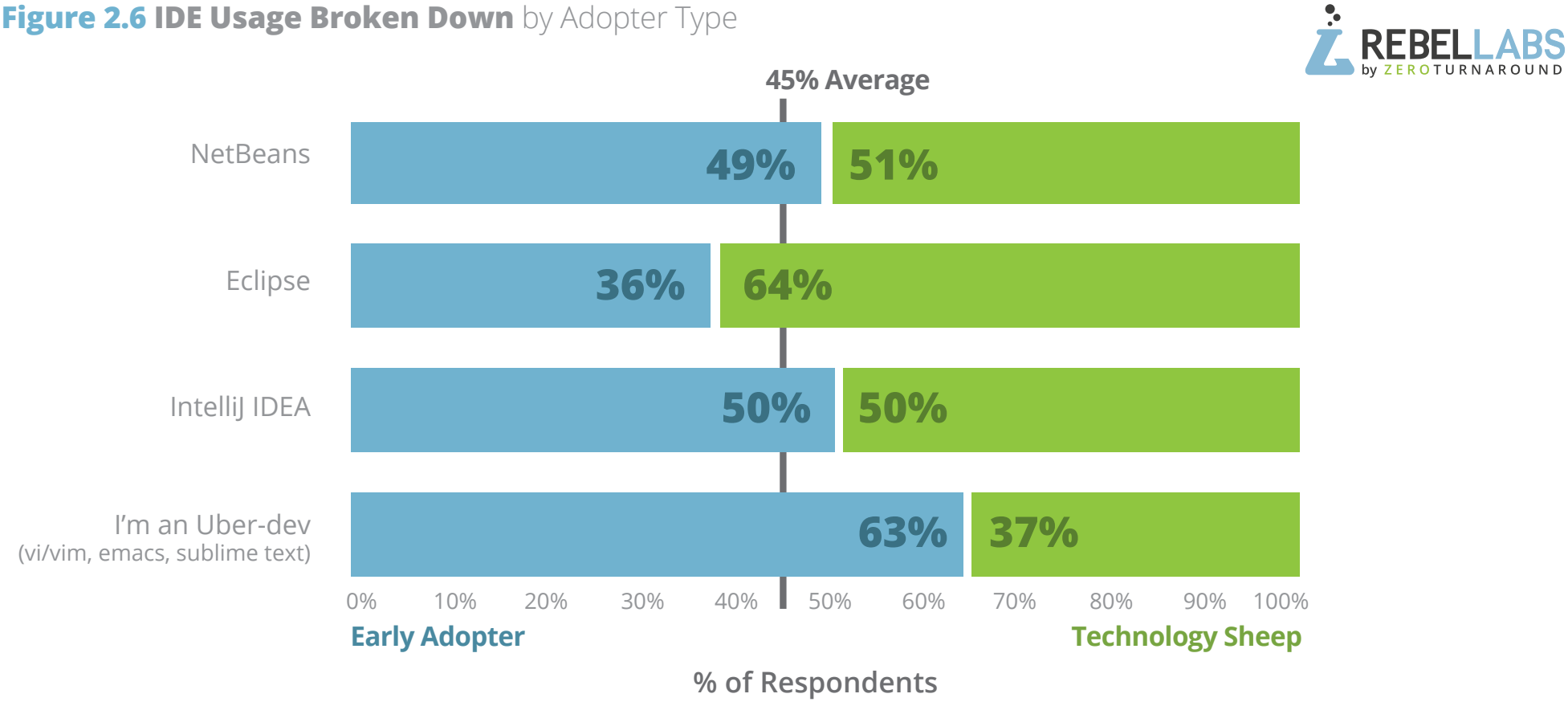
IDE

Let's fan the IDE flame war by introducing the early adopter angle! Once again we'll add the average ratio line of 45:55 in *Figure 2.6* to show the overall average proportion of early adopters to technology sheep. In fact, only one IDE has fewer early adopters than the average.

It's quite significantly lower with only 36% of Eclipse users (just over one in three) being early adopters. We see that both IntelliJ and NetBeans have more than the average number of early adopters with 50% and 49% respectively. Our uber-dev overlords run away with the highest proportion of early adopters — we bow down to your greatness, uber-devs.

We'll take a look at how IDE usage trends have changed over the years and perhaps make sense of this number. If early adopters are moving away from Eclipse, one might wager the technology sheep could later follow.

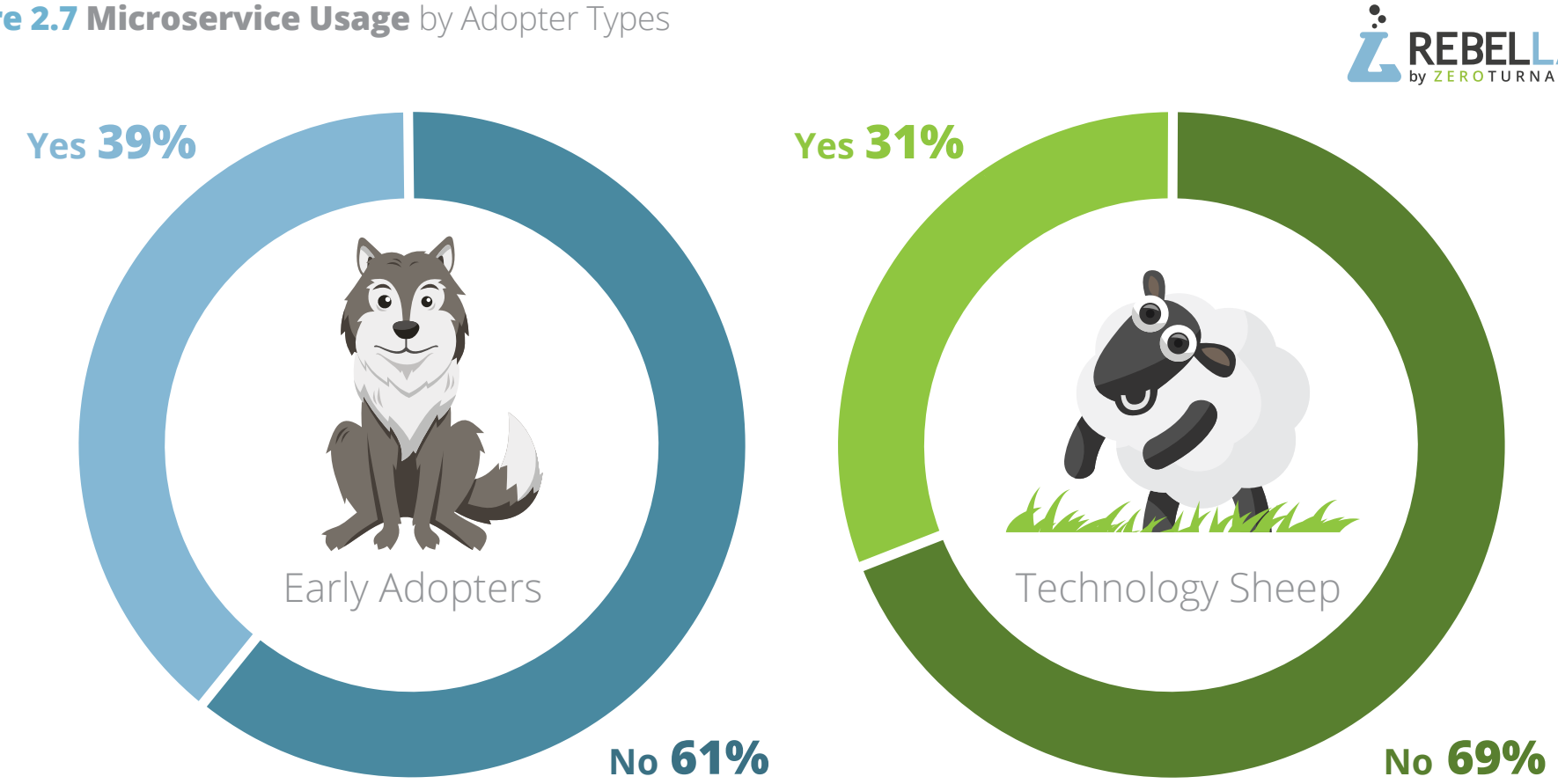
Figure 2.6 IDE Usage Broken Down by Adopter Type



Microservices

Some might say a microservices architecture is a new paradigm or pattern. Others might say it's just a name for the practices they already follow. Regardless of the hype, we see that almost four in ten (39%) early adopters are using a microservices architecture, depicted by our early adopter wolf, shown in *Figure 2.7*! Technology sheep on the other hand are slightly less emphatic with their adoption, although with a little over three out of ten technology sheep using microservices, we can assume they're becoming an established viable alternative.

Figure 2.7 Microservice Usage by Adopter Types



Microservices

Next, we'll dig through our survey responses and pivot on the question that asks if people are currently using a microservices environment. In this section we'll cover application servers, databases, web frameworks and the virtualization tools that respondents who use microservices are using. We'll also check out which roles in an organization find a microservices architecture easier and which find it harder.

Tools

So we'll start the tools section with application servers. To adopt a microservices architecture, it's clear that our applications need to be designed in such a way that they can interact with each other in a modular way. But do those who adopt a microservices architecture favor a particular application server, database, web framework or virtualization tool? Let's find out!

App servers

We'd expect the most lightweight servers to be used for environments that use microservices, and the results don't live up to our expectations. Those using a microservices architecture are twice as likely to use Jetty as those who haven't adopted microservices. Tomcat is also used more in microservices architectures, but while the absolute percentage increases by 6%, the same as Tomcat, it's relatively a much smaller increase for an application server that has 40% of the market among the more traditional environments.

So why does Jetty take such a big slice of the pie when it comes to microservices? Two words: Spring Boot. A framework that's quickly becoming one of the favorites among users of microservices. Spring Boot will embed a Jetty or Tomcat server into the war file that it creates.

WildFly, JBoss EAP, WebLogic, WebSphere and GlassFish they make up 38% of the users who are not using microservices (amusingly still less than Tomcat!). If we look at the usage of the same group of servers that are used by respondents that are using microservices, we see the share slashed down to 23%. WebLogic takes the most significant damage, more than halving in its usage in a microservices architecture, from 9% to 4%.

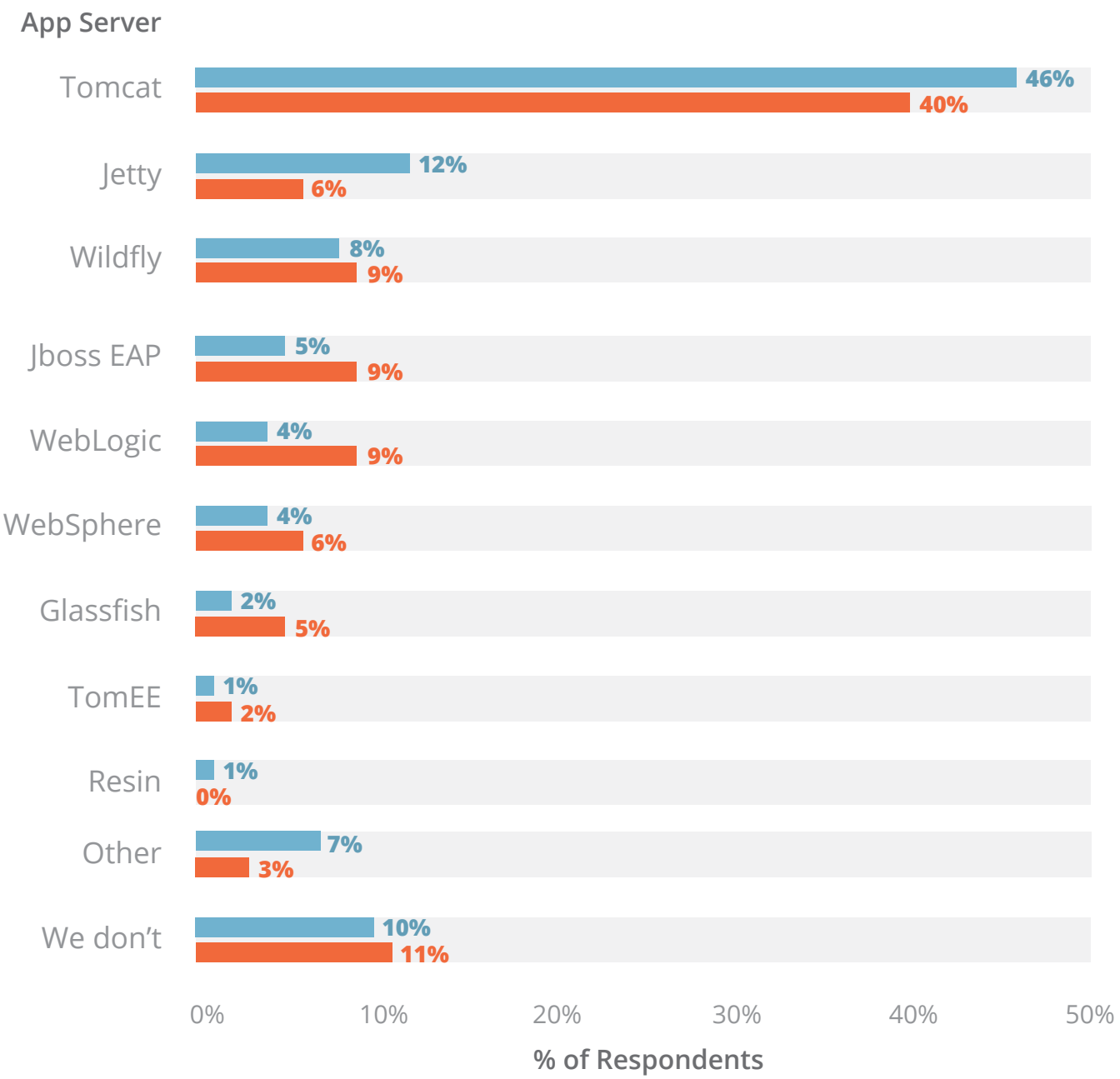


Figure 2.8

Application Server Usage

based on Microservices Adoption

Microservices Users

Non-Microservices Users

REBELLABS by ZEROTURNAROUND

Databases

Everyone has data! I don't care whether you're using monoliths, megaservices, microservices or nanoservices. You're going to have data. You are also going to need some place to store it! The question is, if you pick a microservices architecture, are you more likely to pick certain databases over others? SPOILER — Yes, yes you are! Check out the results on the next page for the full results. In fact, PostgreSQL, MongoDB, Redis, Cassandra increase most significantly! PostgreSQL usage increases from 25% up to 36%, and MongoDB sees an even bigger increase from 10% up to 24%.

Redis and Cassandra show us something subtly different. We see databases that don't have a large share among respondents who don't use microservices excelling in the microservice environment. Redis increases from 5% to 16%, over three times its non-microservices share. Cassandra increases from 2% up to 12%, six times its non-microservices share. Neo4J and Couchbase also show strong signs of relative increased usage in microservices environments, although their lower market share means the absolute percentage increase is smaller than the other vendors.

Oracle, the market leading database offering is one of the few that sees a reduction in usage among microservice users. In fact, the 8% reduction is a significant one, as on 34% it makes Oracle DB to the 3rd most popular database used in the microservices environment. Weirdly, IBM's DB2 which you would imagine would suffer the same fate, increases its share among microservice users from 6% to 8%.

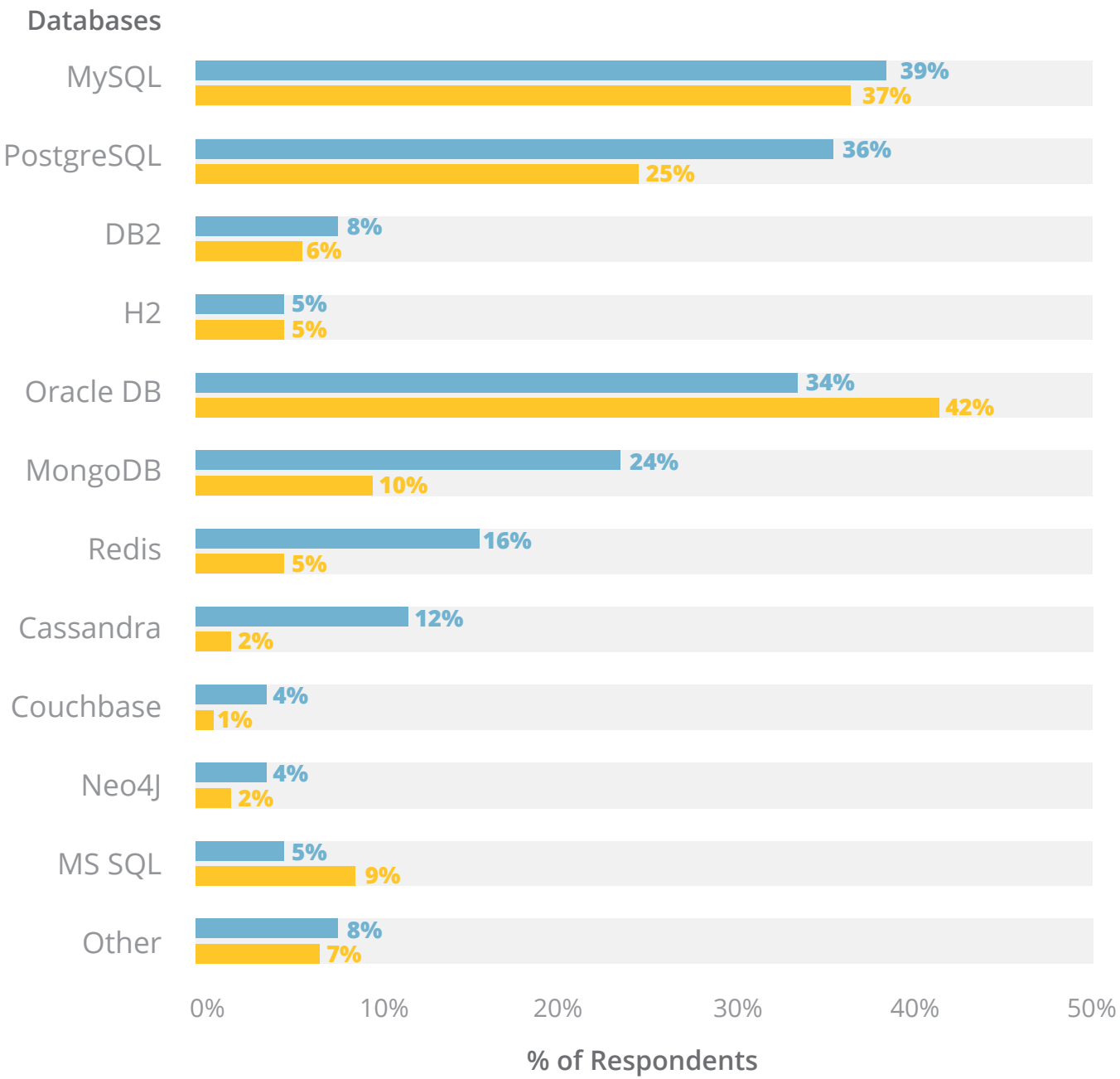


Figure 2.9
Database Usage based on
Microservices Adoption

Microservices Users

Non-Microservices Users

Web Framework

You only need to glance at *Figure 2.10* , on the following page, to spot the two most popular web frameworks that are used by microservices. These are Spring MVC and Spring Boot with 52% and 48% of respondents all jumping for joy stating they use them. So almost one in two microservice-y people are using Spring Boot already, and it's only been available for a little over two years! Not just that, but Spring Boot is four times more popular than any other non-Spring technology (JSF) for microservices users.

Play 2, another technology which is being heavily used by microservice architectures, increases its share to 7% from the 3% share it had from non-microservice users. This shouldn't be too much as a shock, given the part it plays as part of LightBend's new microservices offering, Lagom.

Some frameworks have shown a reduction in popularity among microservice users, none more so than JSF that saw a reduction in usage from 22% of non-microservice users to just 12% of microservice users. GWT, Vaadin and Struts also show a reduction in usage.

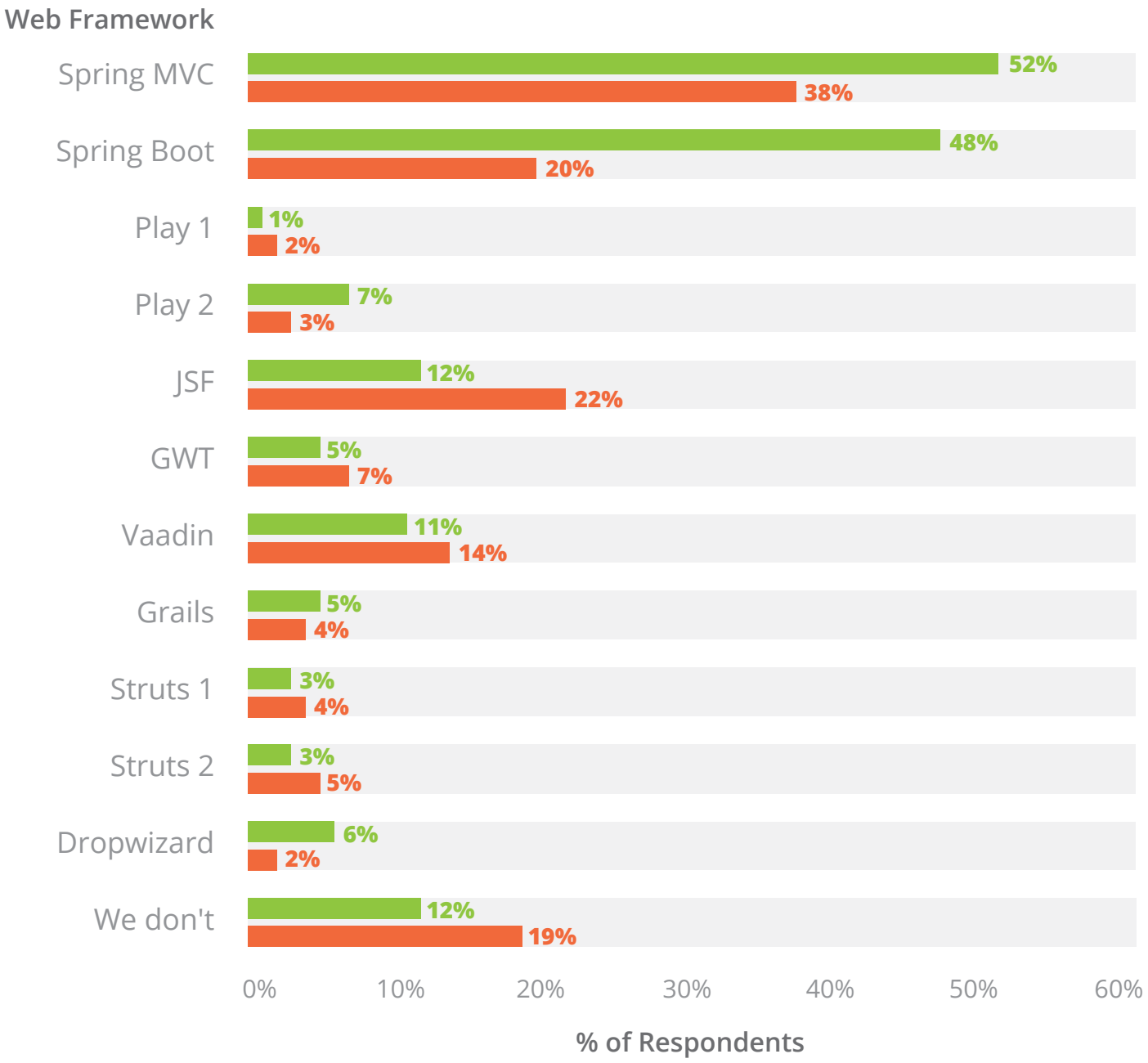


Figure 2.10
Web Framework Usage based on Microservices Adoption

Microservices Users
Non-Microservices Users

Virtualization tool

One technology that you might expect to grow at a similar rate to (if not faster than) microservices is virtualization, Docker in particular. We can see that one in two (50%) microservices architectures use Docker in their environments, compared to just over one in five (22%) non-microservice users.

I mentioned that the lack of Kubernetes usage surprised me in Part 1, and we can see a Kubernetes usage increase in microservice environments by approximately four times compared to more traditional environments.

Just one in three respondents (35%) have adopted a microservices environment but do not use a virtualization tool whatsoever. Compare this to 64% of respondents that have not adopted a microservices architecture who state they do not use a virtualization tool.

Figure 2.11 Virtualization Tool Usage based on Microservices Adoption

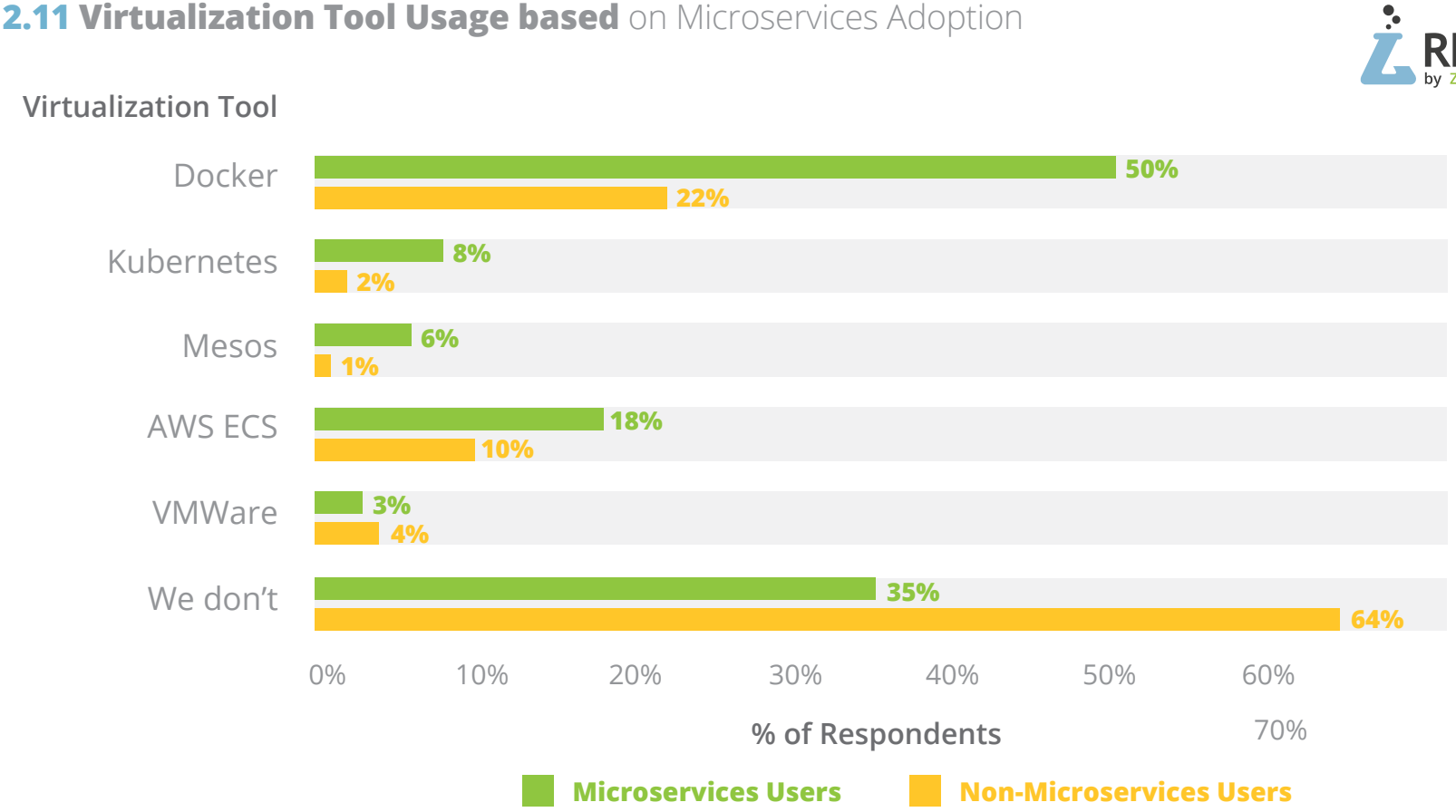
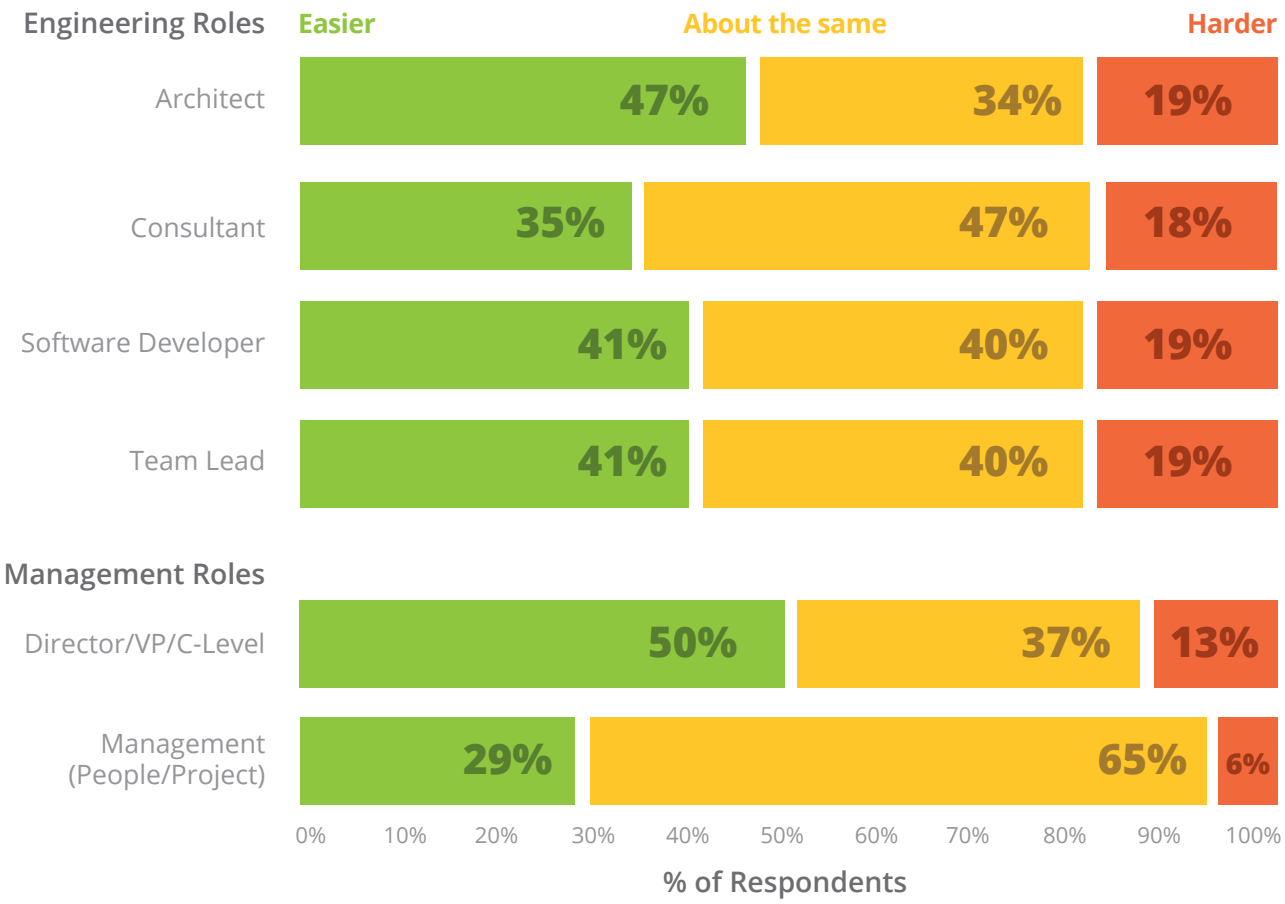


Figure 2.12 Does a microservices architecture make your job role easier?



Are microservices easier, by role?

The final pivot we will perform is to understand which roles become harder or easier when they work with microservices. To make the data easier to read, we've compressed the "*much easier*" and "*easier*" options to just one "*easier*" option, and the same with the "*harder*" options. We've also split the roles up so that you can see the difference between the technical and non-technical roles.

There's a clear bias towards people finding microservice architectures easier to work in, however, it's not that clear cut. For instance, if we assign numbers to the the much easier — much harder scale from 1 to 5 respectively ("*About the Same*" being 3), the mean average ranges from 2.5 to 2.8, which would round up to "*About the Same*". The median across all roles would always be 3, or "*About the Same*".

Non-technical roles are less likely than techies to say their job is harder as a result of adopting microservices, but do disagree when it comes down to whether their jobs are easier as a result of microservices. Well, guess what, it's the Directors, VPs and C levels that ultimately make the decisions! All techies largely follow a similar trend with around one in five finding that microservices has made their job harder and the remainder are evenly split as to whether their job is easier or around the same.

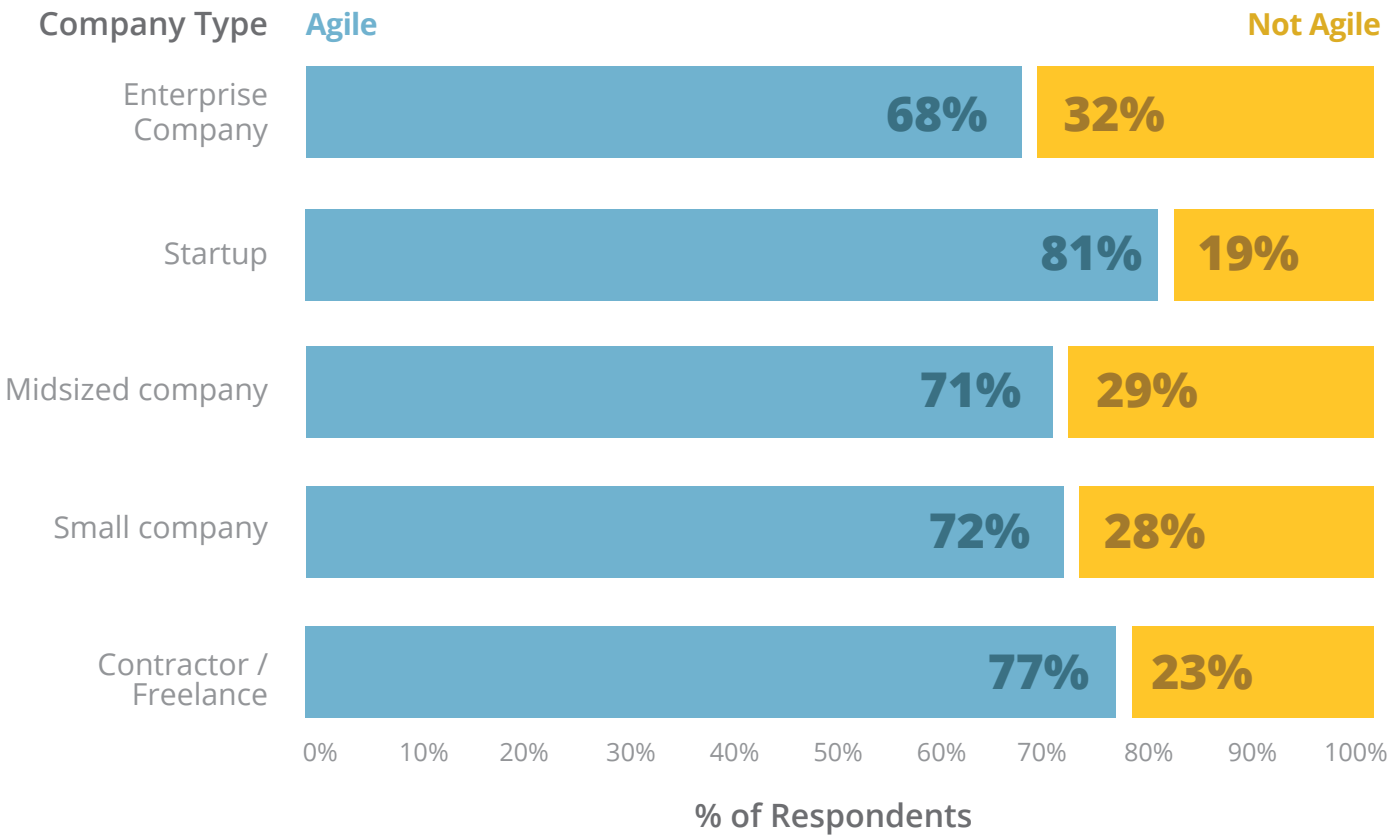
Agility

Those of us who have been agile for years and years won't consider being part of an agile team to be new or amazing. But there are people who do, whether that's by choice or because the team they work with hate change so much, they even wear the same clothes to work each day. Irrespective of the reasons, we can certainly look deeper in the practices and processes which teams use to understand what people are actually doing, as well as whether they label it agile or not.

Company Type

First of all, we can see which companies house the most agile or um... Whatever the opposite of agile is. Unagile? Maybe waterfall-esque? We can see in *Figure 2.13* that the lowest proportion of agile teams exists in enterprise companies, with 68% of enterprise respondents claiming they're agile. This isn't that bad actually, considering small and midsized companies are nearby, on 72% and 71%, respectively. As you might expect, startups are the most agile, with just over eight in ten claiming they're agile.

Figure 2.13 Agility Breakdown by Company Type



REBEL LABS by ZERO TURNAROUND

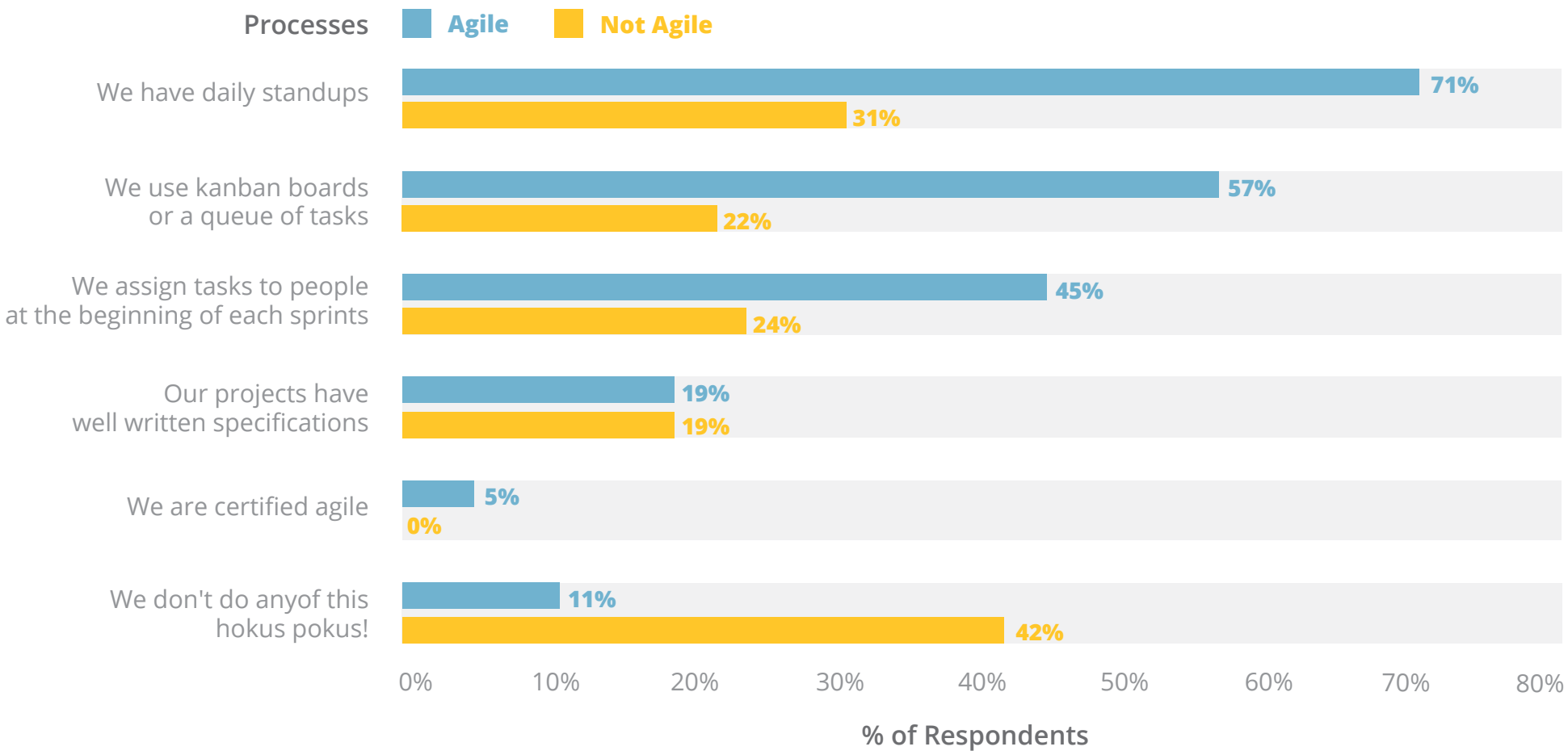
Agile processes

We can now split the question which asked about the processes people follow into two groups. Responses from those who claim to be agile, and those to claim they are not agile. As you can see in *Figure 2.14* on the following page, we have a high percentage of people who claim to be agile having daily standups. Of course, standups aren't for everyone, so it's not surprising to see only seven out of ten (71%) respondents doing this. Naturally, you don't have to be agile to have a standup, so three out of ten (31%) non-agile respondents also have daily standups.

The use of Kanban boards is adopted by 57% of respondents stating they're agile, with just 22% of those not claiming to be agile using Kanban boards. Assigning tasks at the beginning of sprints, or at least development cycles is more popular among agile teams However, one might expect the adoption of this practice to be higher than 45% among agile teams.

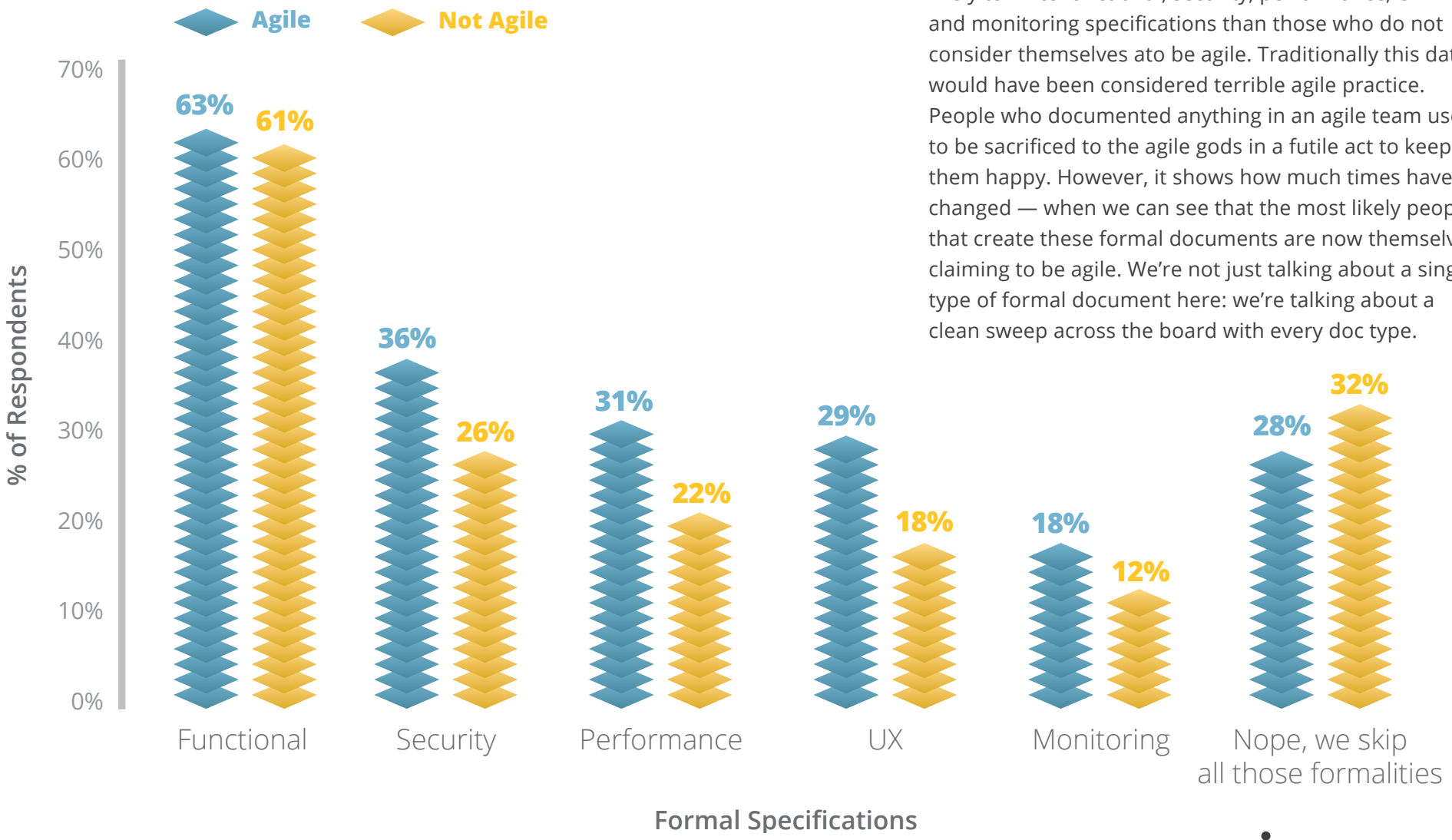
Here's where the figures start to get interesting. When we look at well written specifications, a task you wouldn't traditionally associate with agile development at all (although times have changed now and many agile teams do see this as an acceptable practice), we can see that almost one in five respondents of both agile and non-agile teams create these docs. The data also shows us that 13% of agile respondents only perform standups and don't use Kanban or assign tasks at the start of sprints. OK, there's more to agile than just those tasks, but I think there are many groups that do believe they're agile just because they stand up and chat about kittens each morning.

Figure 2.14 Process Usage by Agile Adoption



REBEL LABS by ZERO TURNAROUND

Figure 2.15 Formal Specification Usage by Agile Adoption



Let's be specific

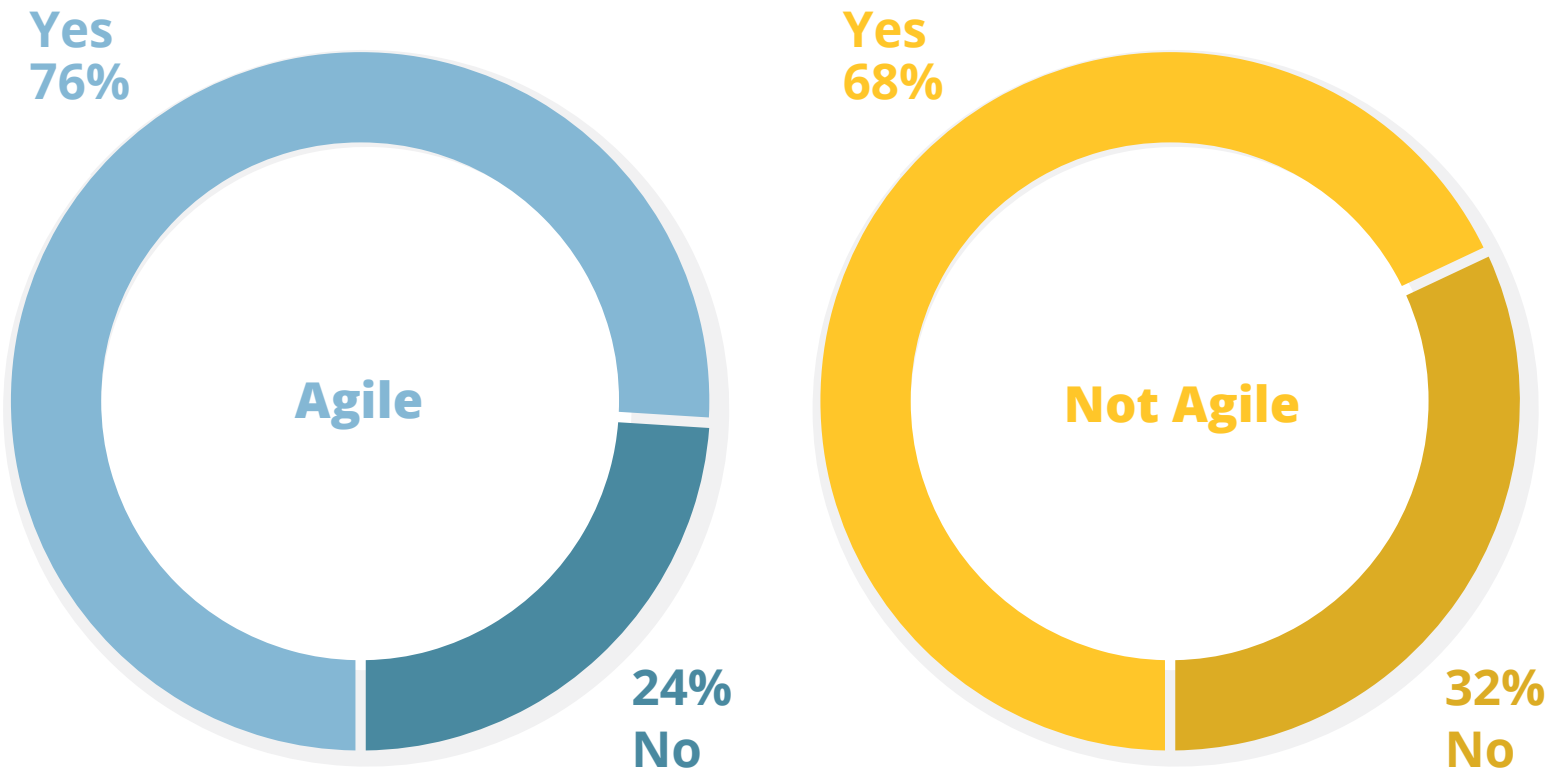
Those who consider themselves to be agile are more likely to write functional, security, performance, UX and monitoring specifications than those who do not consider themselves ato be agile. Traditionally this data would have been considered terrible agile practice. People who documented anything in an agile team used to be sacrificed to the agile gods in a futile act to keep them happy. However, it shows how much times have changed — when we can see that the most likely people that create these formal documents are now themselves claiming to be agile. We're not just talking about a single type of formal document here: we're talking about a clean sweep across the board with every doc type.

Do agile teams think they're better?

The final question in part 2 is, are people in agile teams more likely to think they're better than people in waterfall-y teams? Simply put, yes they do! The difference isn't just one or two percent of people, it's actually almost one in ten people!

76% of respondents claiming to be agile believe they're better than the average person in their role, compared to 68% of non-agile respondents.

Figure 2.16 Are Agile Developers More Likely to Think They're Better?



PART 3: TRENDS

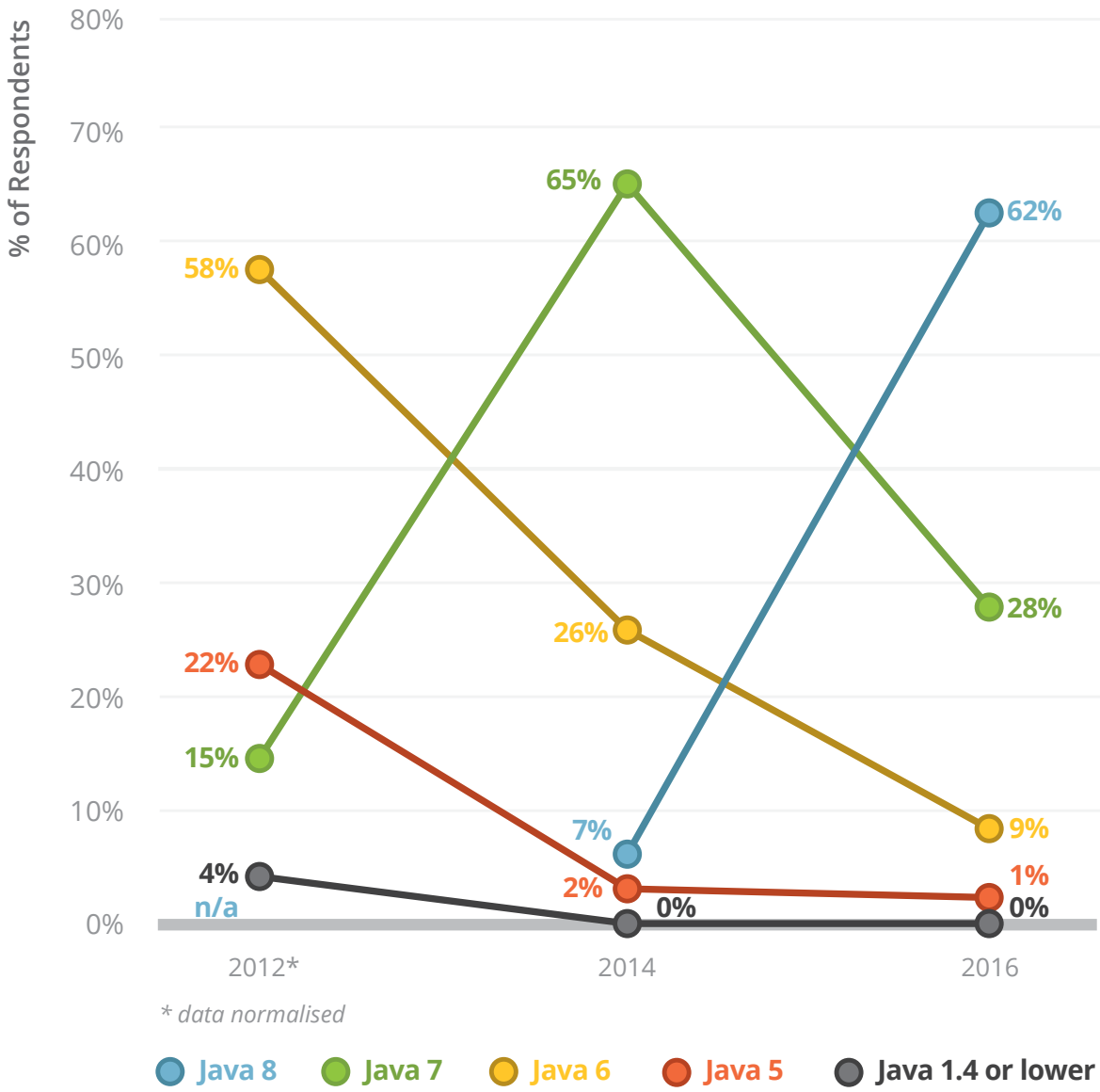
This part of the report is going to look at data trends. However, we're not just going to look at this year's data, oh no! Since we ask a lot of the same questions every couple of years, we can begin to see trends over time, by comparing answers given to the same questions in 2016, 2014 and 2012. This might give us some understanding of what we might expect in 2018, however it would be a little foolhardy to assume that purely based on a few years of data. That said, the numbers and graphs created in this section sure do show a consistency and pattern across the years. Note that some of the data from previous years had to be normalized to cater for questions that allowed multiple answers. These years have been asterisked and marked for your awareness.

Java SE Version Adoption

It's great to see a very consistent level of upgrade and migration in Java versions as the years go by. Migration is a particularly time consuming and thus expensive activity for larger organizations for what some people might consider 'minimal gain' compared to, for example, a new application feature. Looking at you project managers! The performance increase alone is a good reason to upgrade, but with newer concurrency libraries, functional approaches and much more in newer versions, the benefits to a developer and a production environment are very clear.

It's also good to see such a drop-off on the older versions, particularly for library and tool manufacturers who can start to remove support for older versions of Java. Given Java 9 will not be released till March 2017, if current deadlines are met, I would expect to see Java 8 reach the 70-80% adoption mark even before we start to see Java 9 begin to ramp up. I don't expect developers to rush out and adopt Java 9 as quickly as they did with Java 8, as there doesn't seem to be enough eye-candy being delivered to developers. There's also a lot of churn with a new default garbage collector and modules, so it wouldn't surprise me to see Java 8 being the dominant version, even in 2018. By then I would expect Java 6 usage to be very low, perhaps close to if not actually at 0% adoption, with Java 7 close by around 5%. I'm already looking forward to revisiting these predictions in our 2018 report!

Figure 3.1 Java SE Version Adoption Since 2012



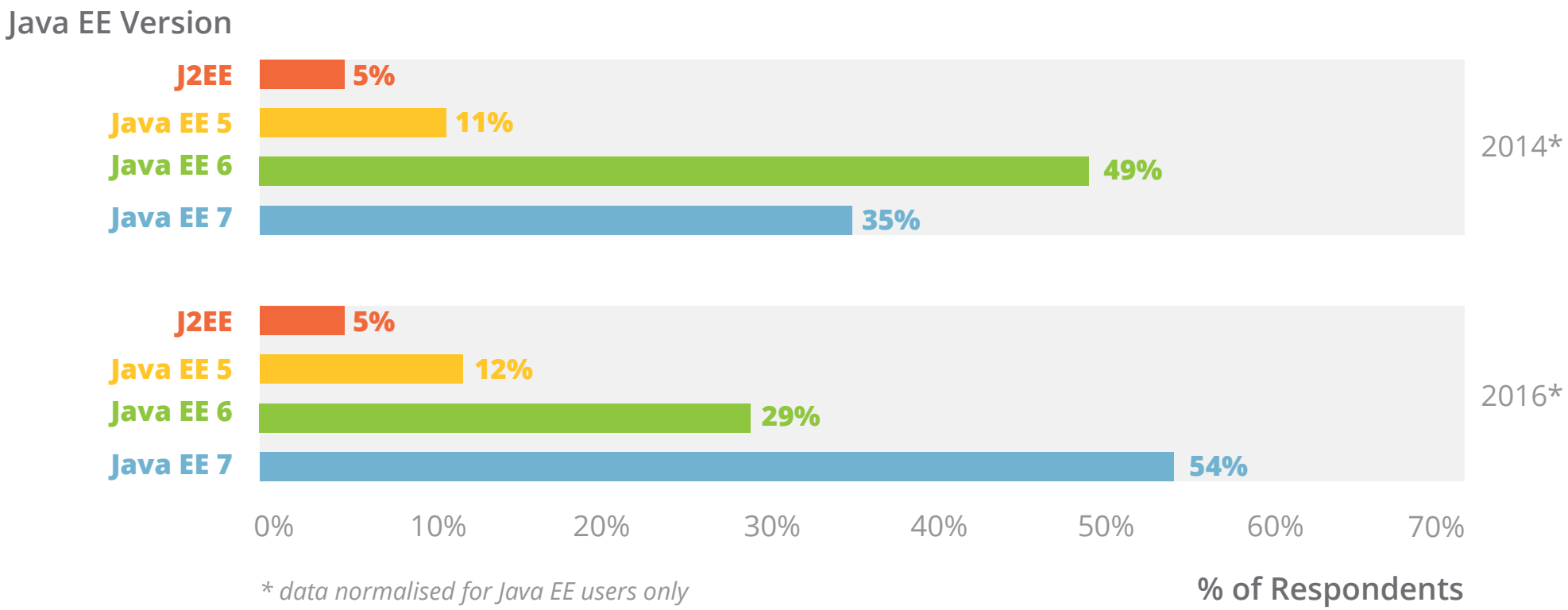
Java EE Version Adoption

We'll now move on to our enterprise brethren, Java EE. We can see two interesting things from this data, shown on the following page. Firstly, Java EE 7 has not just taken the mantle from Java EE 6, but is more popular today than Java EE 6 was back in 2014. The second important point is that people are not moving off of Java EE 5 and J2EE. Given how old these technologies are, and more importantly, how far Java EE has come since the J2EE standard, it's puzzling to understand why there are still so many persisting with these legacy environments.

However, what could be more worrying for the Java EE community is what isn't shown in *Figure 3.*2. We've normalized the data to show which versions Java EE developers are using, so if you flex your mathematical brain, you'll see the yearly percentages add up to 100%. Let's take a moment to see how many people aren't using Java EE. From the 2014 survey, 32% of respondents were not using Java EE, compared to 42% in 2016. This means that 15% of the Java EE users in 2014 decided to stop using Java EE in the past 2 years. This supports the data we saw in *Figure 2.4*, showing that early adopters are looking at alternate frameworks. Summed up, this doesn't look great for the future of Java EE community.

If we look at the state of the Java EE 8 JSRs, it already looks like deadlines will be missed, particularly with the Oracle-led JSRs, which has already been so well documented in the news. This is also the reason why the Java EE guardians group was formed, to help bring awareness and change into the current state of Java EE. I expect the adoption of Java EE will continue to reduce, particularly if Oracle's lack of interest in the technology continues, as vendors and users will have low confidence in its future. I expect Java EE 7 to continue to take share from Java EE 6 users, and while I hope to see the Java EE 5 and J2EE usage drop, the data just isn't there to back it up.

Figure 3.2 Java EE Version Adoption Since 2014



IDEs

In our 2014 report, 49% of respondents stated they'd like to be using IntelliJ IDEA as their IDE. These respondents may have already been using IntelliJ, or they could have been using other IDEs. Since then, as we saw back in *Figure 1.11*, IntelliJ has taken the mantle of most popular IDE, by usage, from Eclipse. When we look at the trend in *Figure 3.3*, we could have seen it coming, as the rate of adoption has remained consistent enough for both Eclipse (negative) and IntelliJ (positive).

I don't expect this to continue at the same rate beyond 2016, but wouldn't be surprised if we see IntelliJ take a 10-15% lead from Eclipse. I don't think NetBeans will change that much, although with Oracle's very average efforts in showing support for the NetBeans platform, their own steward could be their very downfall. There are however, many strong individuals on the NetBeans team that, despite a lack of support, will do everything they can to make the platform continue to be successful.

Figure 3.3 IDE Usage Since 2012

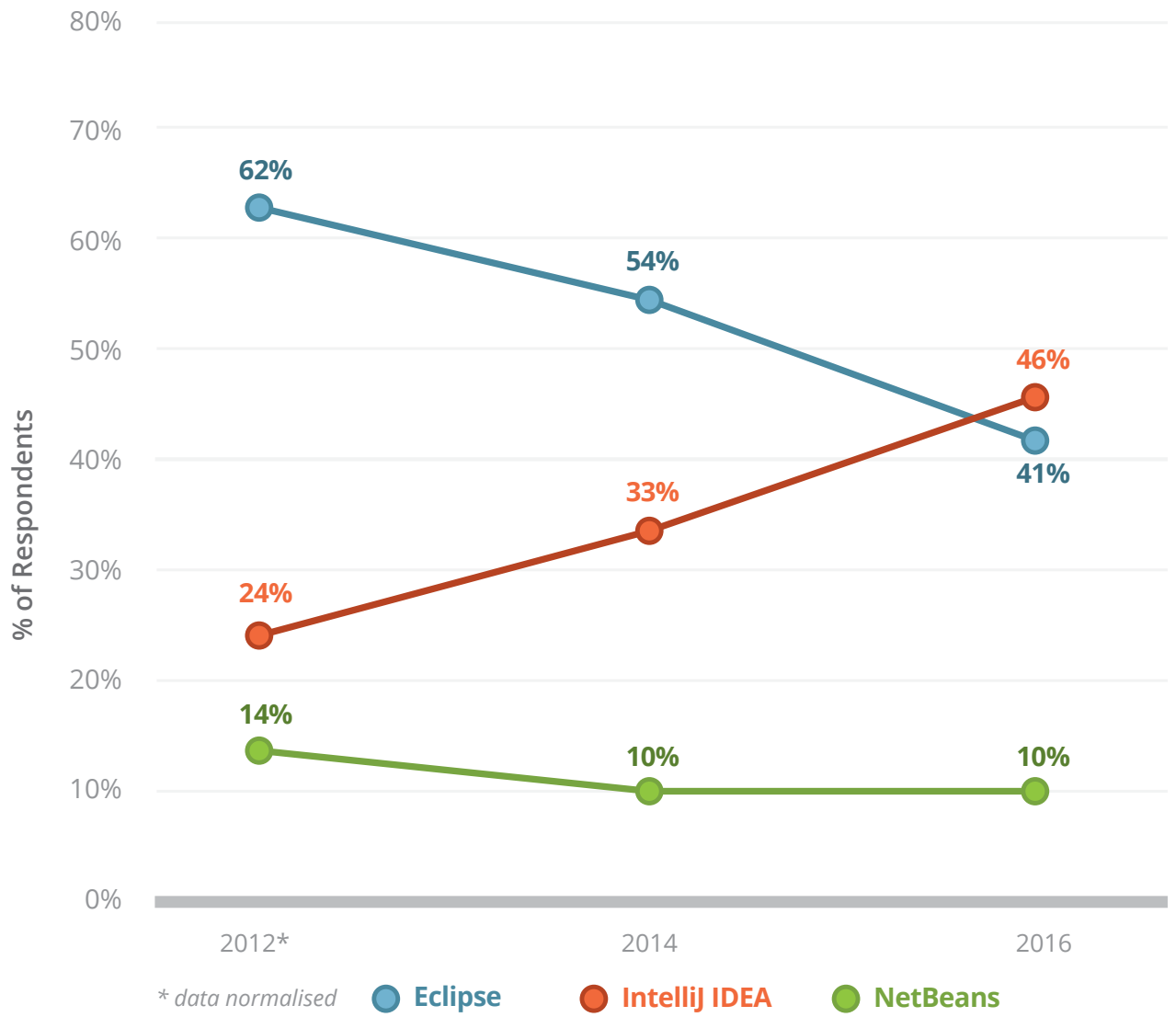
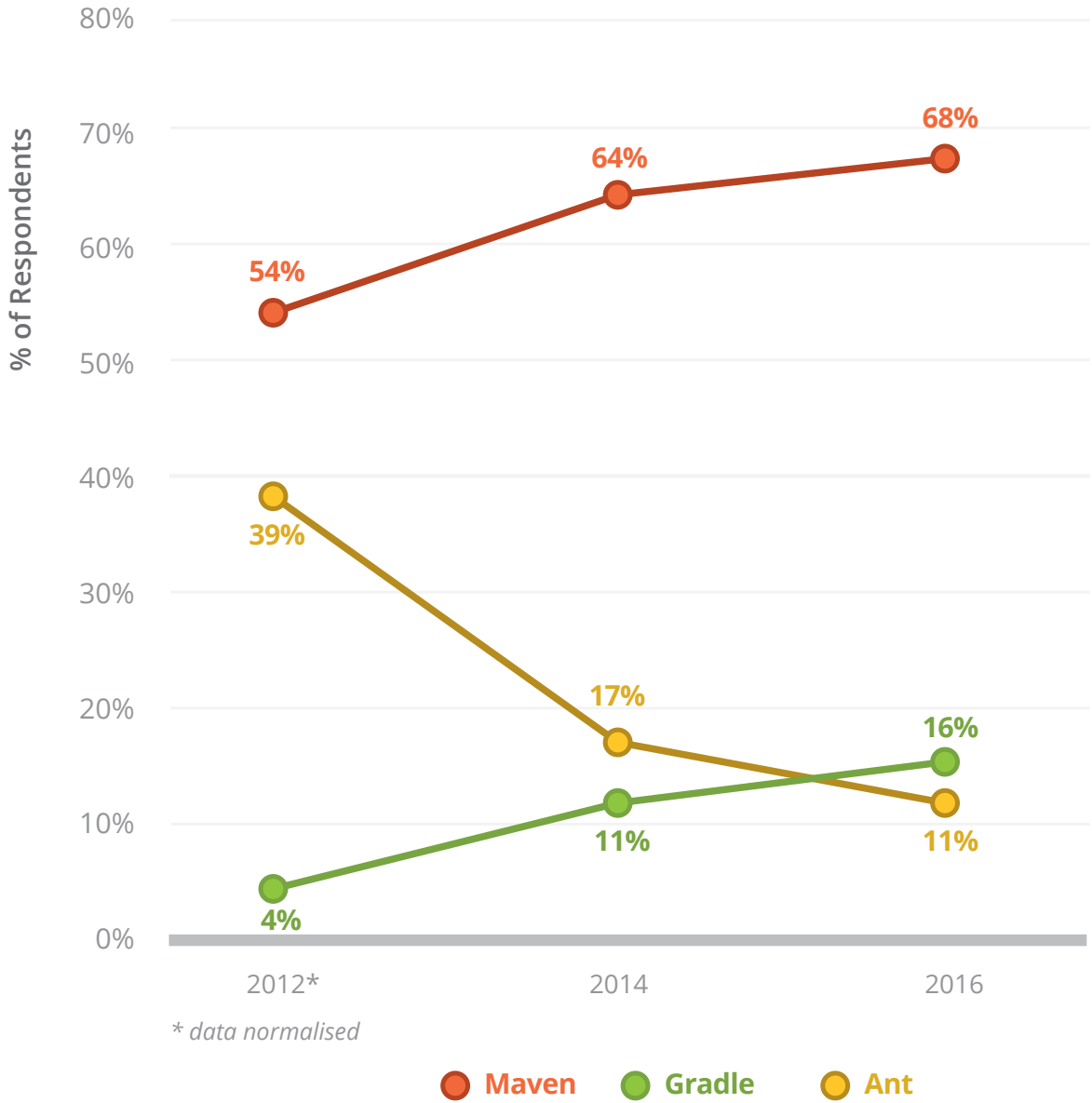


Figure 3.4 Build Tool Usage Since 2012



Build Tools

Another category, rife with flame war, is the build tools category. But is this even justified? Well, if we were to look at the build tool adoption from 2012 to 2016, we can reasonably state that it's a one horse race that has already been won by Maven. In fact, if it were a horse race, Ant would have already been shot, and people might have already thrown away their Gradle 500-1 betting slip. However, they may be light at the end of the tunnel, as Gradle is being used heavily in Android development environments. The question about whether the light is indeed the end of the tunnel or a Google train called Bazel, we'll have to wait and see.

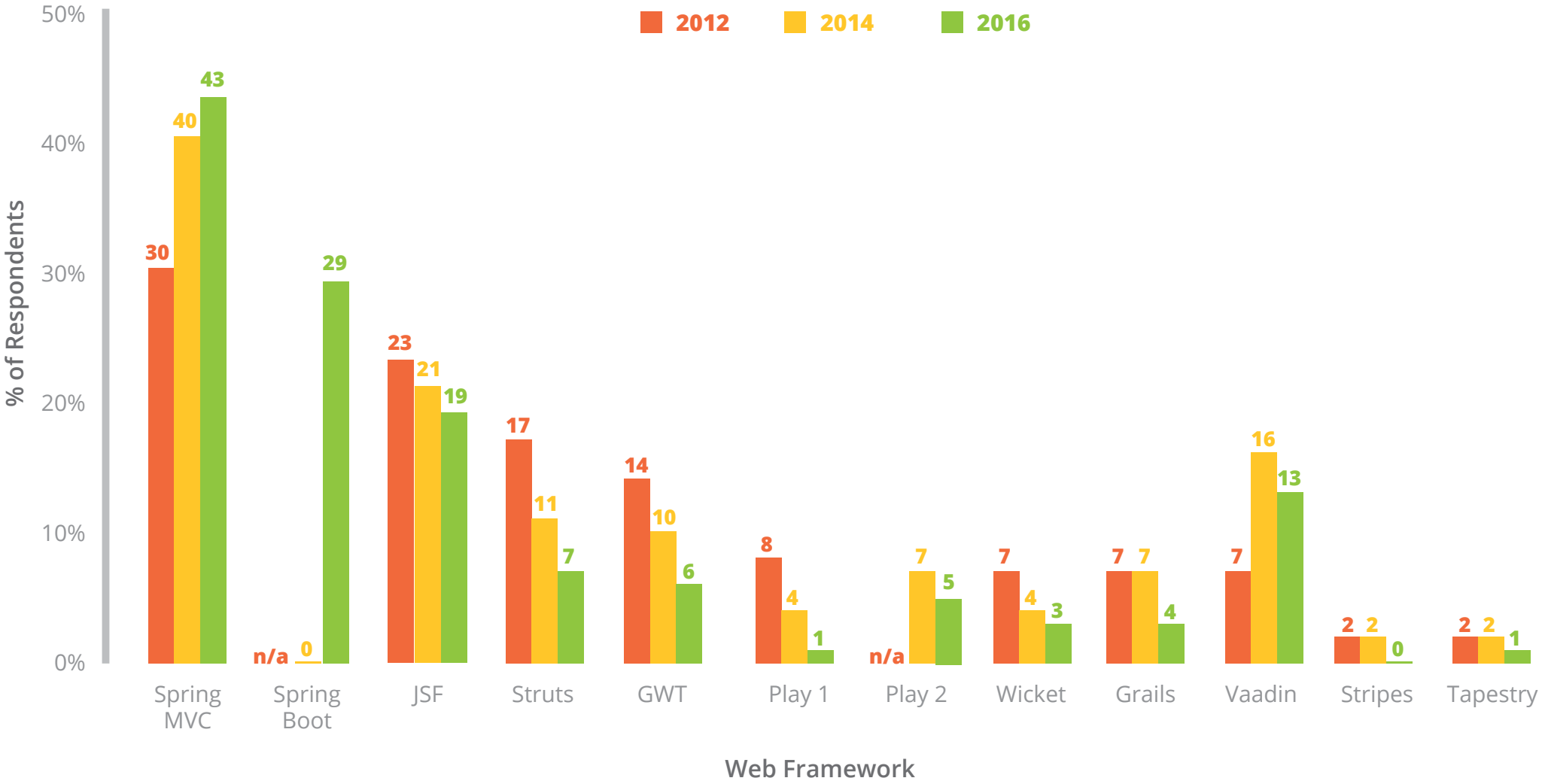
Now for some predictions. In the next couple of years, I don't see Maven really moving much in terms of market share. It could perhaps dip, if Gradle's adoption rate does increase, although Gradle seems to be taking more share from Ant than Maven. I think Gradle will continue to rise slowly, particularly if the support from the Android community continues. Ant on the other hand will continue to fall, most likely to 6-7% by 2018.

Web Frameworks

In *Figure 3.5* on the following page, we can see Spring's dominance over the past four years increase as they monopolize the web framework market. Pivotal can be very happy with their Spring Boot adoption rate and this sets them up nicely as the microservices market begins to grow. JSF is slowly decreasing, although this might just be a result of reduced Java EE adoption overall. Struts, GWT, Play 1, Wicket, Stripes and Tapestry struggle to stay popular, while Vaadin is one of the few frameworks that has shown resilience during the Spring domination.

Going forward, I expect to see Spring continue to dominate with Spring Boot, looking like it wants to take over as the overall leader. In 2018 I think Spring Boot and Spring MVC will be pretty much neck and neck with Boot looking the more likely to want to dominate, riding the wave of the microservices architecture beneath it. I think JSF will still be relevant, although it will need proper backing from our Java stewards to make sure people can have confidence in it progressing and staying up to date. It will likely continue its slow decline, given the Java EE 8 date looks all but certain to slip, although I don't think it will drop too far. Play 2 will more than likely increase its adoption rate, given the leading role it's playing in Lagom, with Akka. Other frameworks, like Struts 1, Play 1, Wicket, Stripes and Tapestry may be mentioned in our 2018 report, but more than likely just in the obituary column.

Figure 3.5 Web Framework Usage Since 2012



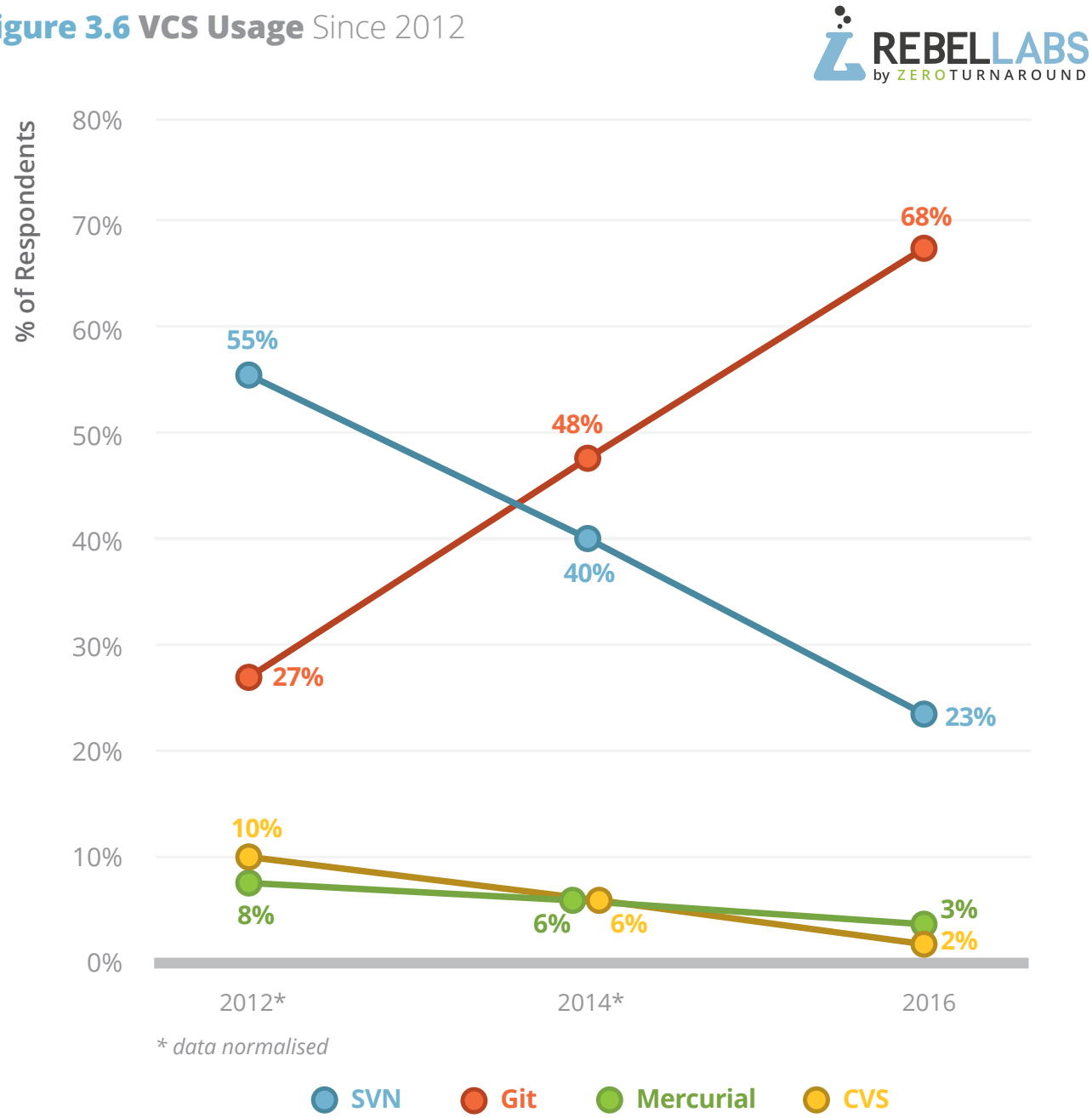
REBELLABS by ZEROTURNAROUND

VCS

The final comparison we'll make in this part of the report, and in fact the last graph of the entire report (Thanks to our designer Ladi for being so understanding!) covers our VCS tools. The trend here couldn't be more clear cut. Mercurial and CVS, you're coming to the end of a losing battle. SVN, well... Mercurial and CVS are just keeping your deathbed warm. Git is the clear winner with a very healthy and consistent market gain, culminating in a 68% market share.

My predictions are simple. I think Git growth has to slow down, but I still see room for Git to grow to 80% market share in 2018 before tailing off. Mercurial and CVS will likely go the way of the dodo by 2018, with SVN's share reducing heavily again to around 5-10%.

Figure 3.6 VCS Usage Since 2012



Tools that excite developers

This year the survey included a couple of self-reflection questions asking more about one's own experiences with particular tools or technologies. The question in the 2016 survey was as follows:

What tool, technology or library are you super excited or proud about having used or planning to use in 2016?

Note that we allowed free form answers to this question. This freedom, while useful for discovering interesting lesser known projects and tools, makes the analysis a little more manual than just poking Excel spreadsheet with a few formulae.

After exporting all the responses, we made the following transformations:

- Convert the responses to list of comma separated names
- Convert the responses to lowercase
- Split by ", "
- Count the frequencies of the terms in the list
- Apply a list of the synonyms like:
    - "Spring boot", "springboot", "spring boot 1.3.5"
    - "Java EE", "JavaEE", "Java EE7"
    - "Java EE 6", "JavaEE6"
- Count the frequencies again and collect the top entries using the famous word frequency count Java 8 stream example program.

Now we have a pretty decent name-dropping list where unrelated technologies go hand in hand. However, this is a meaningful result that shows where the industry at large is heading. Without further ado, we present the top 10 most frequent technologies mentioned in the response to the question above:

- **Docker**
- **Spring Boot**
- **Angular**
- **Java 8**
- **Vaadin**
- **Kotlin**
- **Microservices**
- **Scala**
- **Akka**
- **RxJava**

We don't claim that it's the definitive list of things to master, but it is quite telling that more people are excited about and proud of using these tools. Surprisingly, there wasn't a significant difference in the exciting technology choices between those who claimed they are better than the average person in their role versus their more humble respondents. Go figure, we can think we're different, but we all solve similar problems using similar tools.

SUMMARY

You've made it to the end of the report — well done!
Or did you just skip here to get to the TL;DR? :) Either way, we'll now cover all the highlights of the report in an easy to consume manner. Get ready for a fast paced statistic-fest!

TL;DR For those in a hurry

THIS REPORT IN GENERAL

Here's some background on the report and the data.

- **2040** participants completed the survey in full.
- The survey took place from February through March, 2016.
- **$1000** was donated to **Devoxx4Kids** — hooray!
- No geeks were harmed in the creation of this report.

RAW DATA

This section of the report provided the raw results of each question in the survey.

- The average respondent was a software developer working on a full stack web application, with approximately 10-12 years of experience.
- 63% of respondents worked for an enterprise or a mid-sized company.
- Three out of four respondents (74%) think they're better than the average person in their role.
- One third of respondents **(34%)** have adopted a **microservices** architecture.
- Of the two thirds (66%) that haven't adopted microservices, only 12% are currently planning to do it in future.
- **Java 8 is mainstream**, with **62%** of respondents using Java 8 in **production**.
- **Java EE 7** is the most popular version with **31%** of respondents stating they use the latest version.
- **42%** of respondents don't use **Java EE** at all.
- **46%** of respondents use **IntelliJ** more often than any other IDE, finally overtaking Eclipse on 41%.
- **68%** of respondents use **Maven** as their main build tool, with Gradle on 16%.
- **Tomcat** is still the most popular application server in production and development with **42%** market share.
- **Oracle DB** just pipped MySQL to the most used database with **39%** and 38% of respondents claiming to use each, respectively.
- **MongoDB** is the most popular NoSQL DB with **15%** of respondents using it today.
- Spring totally dominates the web framework market with **Spring MVC** and **Spring Boot** taking first *and* second most popular frameworks with **43%** and **29%** using them, respectively.
- **Jenkins** dominates the CI Server market with **60%** of respondents favoring the butler solution.
- **Git** owns the CVS market with **68%** of the share, as SVN struggles on 23%.
- **Visual VM** continues to be the most popular profiler with 38% respondents making use of it.
- Developers don't care about APMs to the extent that 30% don't know which APM is used on their application.
- **New Relic** is the leading APMs with 11% share.
- One in three (32%) use **Docker** but over half of respondents (54%) don't use virtualization environments at all.
- 71% of respondents claim to be agile.

TRENDS

This section pivoted the data for find trends in our respondent's answers

- Both early adopters (67%) and technology sheep (58%) favor **Java 8**, showing how established the latest version is.
- Early adopters are more likely to **move away from Java EE**, with more technology sheep (33%) preferring the latest version than early adopters (28%).
- Early adopters favor Spring Boot, Play 2 and Grails, snubbing Struts, Wicket and Play 1.
- **Jetty** and **Tomcat** are the preferred application servers for a microservices architecture, with WebLogic, WebSphere and GlassFish in particular seeing reduced usage (almost half) in microservices environments.
- PostgreSQL, MongoDB, Redis, Cassandra and Couchbase see heavily increased usage in microservices environments (up to six times), with Oracle DB dropping by 8%.
- **Spring Boot** and **Play 2** are twice as likely to be used in a microservices environment than not, whereas JSF is almost half as likely.
- One in two (50%) microservices environments use **Docker** for virtualisation.
- Agile teams are more likely to write formal technical specifications than non-agile teams.

WHAT HAS CHANGED IN THE LAST 4 YEARS

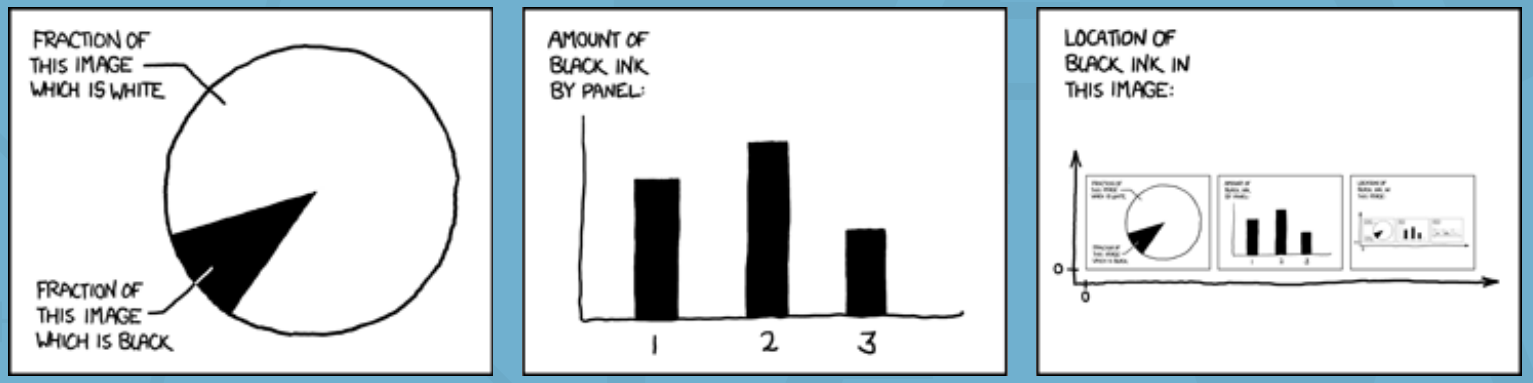
This section looked back through our reports to show the technology trends over the last four years.

- **Eclipse** usage continues to decline consistently. **IntelliJ** share continues to grow consistently. IntelliJ predictably overtakes Eclipse by usage.
- **Maven** continues to be market leader with steady growth.
- **Gradle** continues to grow slowly but not significantly enough to challenge Maven.
- **Spring**'s dominance is shown in its continual growth from 2012 to 2016, including very fast adoption for Spring Boot.
- **JSF** usage declines slowly over time, while Stripes, Tapestry, Wicket and Play 1 look to be on the way out.
- **Git** shows fast growth over the last four years from 27% to 68% share. SVN on the other hand shows substantial market share loss in the same period from 55% down to 23%.

GOODBYE AND A COMIC

Thanks for reading! I hope you found the results enlightening and can use this information along with your own data to better help you understand the landscape of our wonderful ecosystem, as well as give you more confidence in your future decisions.

Given we've looked over so much data, and analyzed so many charts, I'll leave you with an xkcd comic, on graphs themselves.



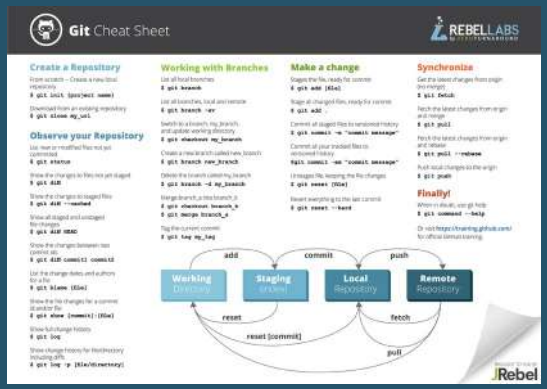
https://xkcd.com/688/

Learn more...

REPORT

Developer Productivity Report 2015: Java Performance Survey Results

CHEATSHEET



Git Commands and Best Practices Cheat Sheet

WEBINAR

Masterclass: Intro to XRebel
A live web session by ZeroTurnaround

THANKS FOR READING!

NO DEVELOPERS WERE HARMED IN THE MAKING OF THIS REPORT :)

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
Jankovcova 1037/49
Building C, 5th floor
170 00 Prague 7, Czech Republic
Phone: +420 227 020 130

Report Author:
Simon Maple (@sjmaple)

Report Contributors:
Oleg Shelajev (@shelajev), Mart Redi (@martredi)

Report Designer: Ladislava Bohacova (@ladislava)