DEVELOPER PRODUCTIVITY REPORT

JAVA TOOLS, TECH, DEVS AND DATA

2012

Can you handle it?

ZEROTURNAROUND

TABLE OF CONTENTS

INTRODUCTION

Productivity Survey

Digging into data searching for insights is always an exciting activity. Last year's Java EE Productivity Report was focused on tools/technologies/standards in use (exclusive selections) and turnaround time in Java (i.e. how much time is spent per hour redeploying/restarting).

In this year's productivity report, we expanded the selection of technologies and tools available to Java development teams to choose from, made them non-exclusive, and covered more areas, for example Version Control Systems and Code Quality Tools. We also focused more on the question, "What makes developers tick?" and learned a lot about how devs spend their work week, what elements of the developer worklife increases/decreases efficiency, and what stresses out developers. We found a lot of interesting trends and insights, and broke them down into 4 parts:

Part I: Developer Tools & Technologies Usage Report
coverage of Java versions, JVM languages, IDEs, Build Tools, Application Servers (containers), Web Frameworks, Application (server-side) Frameworks, JVM Standards, Continuous Integration Servers, Frontend Technology, Code Quality Tools, Version Control Systems

Part II: Developer Timesheet
How do devs spend their work week?

Part III: Developer Efficiency
What aspects of your job make devs more/less efficient?

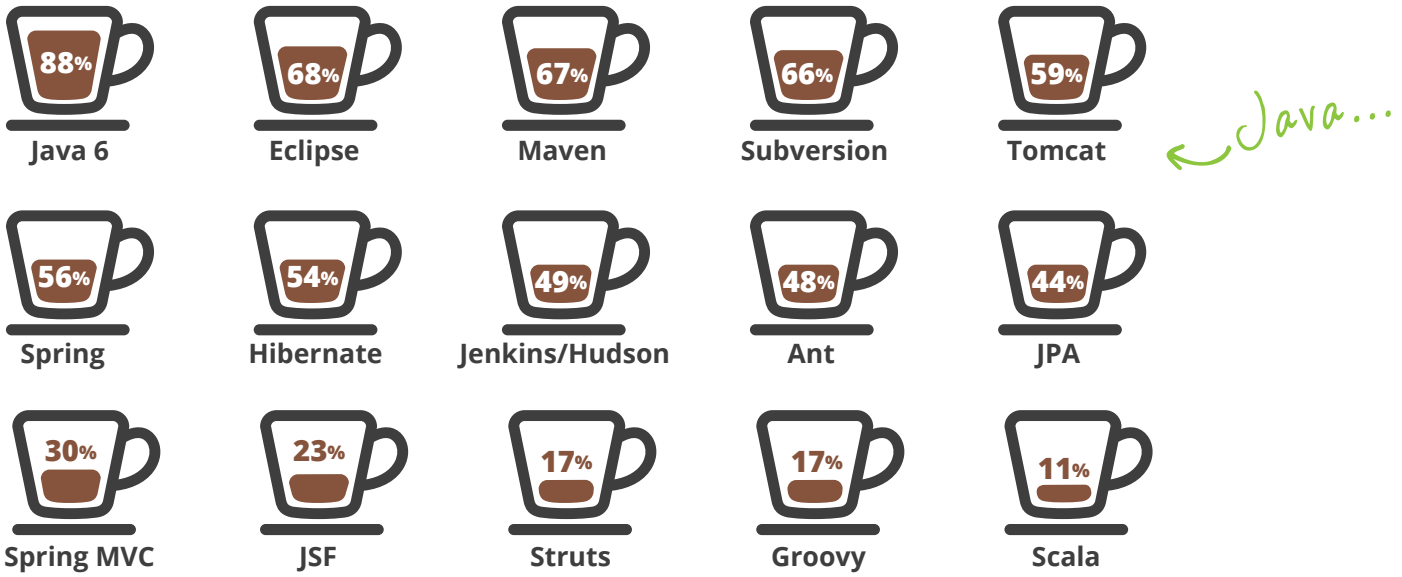
Part IV: Developer Stress
What keeps devs awake at night?

But before we go any **deeper**, let's review the data **as a whole**.

Although we ran the survey through an open call on the web, there is a certain amount of bias present in it. Obviously, our customers are more likely to reply to the call than anyone else. Also, it's likely that people who find and take such surveys are less conservative in their technology choice than those who don't. To protect against such bias, we asked for the size of the company and the industry, to be able to normalize the data, if there is any overrepresentation. However the no size or industry was overrepresented, so aside from some "early adopter" bias there shouldn't be a lot of misrepresentation.

Another thing to understand is what are we measuring. Popularity can be measured in different ways -- number of users, number of organizations, number of lines of code, etc. In this survey we measure the penetration of tools & technology in the market. So 10% of respondents means that likely 10% of the organizations that do Java development use this tool or technology somewhere in the organization. It does not tell us whether it is critical for them or not, whether it is used a lot or only in rare cases and whether it is used by everyone or only the respondent.

Tools & Tech Leaderboard 2012



PART I
DEVELOPER TOOLS & TECHNOLOGIES & USAGE REPORT

When the Machines Took Over

Before getting into the stats and analysis, check out this introduction, where we quickly review some things happening in the industry, which are the *de facto* tools & technologies used by the majority of developers, what tools are increasing or losing market share compared to last year, and where things generally stand among our respondents.

Tools & Technologies **used by 66%** + of Respondents

Java 6 is the overwhelming version of Java, used by 88% of respondents. But more interesting and exciting is the fact that 23% of respondents are already using Java SE 7. This is amazing penetration, considering it came out less than half a year before the time of this survey. This gives hope that as Java SE 8, 9 and 10 come out in the next 6 years or so, a lot of engineers will be able to benefit from the changes quickly.

The tools with over two thirds of respondents -- **Eclipse, Maven and Subversion** -- are now the *de facto* standard in the Java development environment. This universal popularity does not mean that they are not without challenge -- indeed, Maven is often supplemented by Ant, itself used by almost half the respondents. Subversion is slowly losing ground to Git and to a lesser extent Mercurial. In the IDE space, both IntelliJ IDEA and NetBeans have made great progress in challenging Eclipse since the last year.

Tools and Technologies **used by ~50%** of respondents

The tools and technologies that are used by about half of respondents are JPA, Ant, Jenkins/Hudson, Tomcat, Spring and Hibernate. These are all proven technologies and with the exception of Jenkins have had a stable large market share for years. Jenkins has grown a great deal in past few years and probably will continue to gain market share unless challenger(s) emerge. Tomcat will quite likely join the *de facto* standard group next year (and see the jump in popularity behind Jetty in the App Servers section) and Spring and Hibernate are still holding ground well against Java EE, but it will be interesting to see what will happen in the long run, as Java EE keeps getting lighter, while Spring keeps getting heavier.

JVM languages & Web frameworks **moving forward**

It is interesting to see that Groovy is now at 17% and Scala at 11%. Groovy has become the scripting language of choice on the Java platform and a lot of amazing tools are based on its use in a Java environment, e.g. Gradle and Spock. Scala has established itself as the Java alternative on the JVM and is especially gaining ground with those who need a highly distributed environment or great messaging, but don't want to use Erlang. Akka and Lift are the two killer frameworks that enable such use.

The cool newcomer award goes to Clojure, which managed to quickly gain almost as much support as JRuby in a short amount of time. The web framework market remain as fragmented as ever, with Spring MVC, JSF and Struts leading the flock.

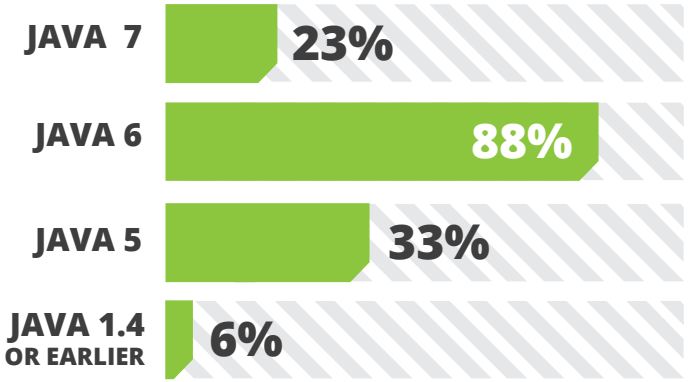
An interesting thing is that these numbers change little if we choose a particular sector: big companies, medium companies, small companies, industry verticals. There are a few correlations here and there (e.g. small companies have more preference for free tools and technologies), but the picture does not change a great deal. To us this says that the community has a larger role in determining the tools and technologies used than the business does, which means that it's a great time to be a software engineer.

Java **Versions**

We were happy to see that Java 6 has been adopted so widely, used by 88% of respondents. It didn't add much to the language compared to Java 5, but brought many performance improvements and some additions to the library. It's the most stable version of Java currently and still receives new improvements.

We like that Java 1.4 seems close to being phased out, but 6% of users are still stuck with it. It is likely that tools such as JRebel, build tools and IDEs will continue to support it for some time. Java 5 is still hanging on with 1/3 of respondents using it.

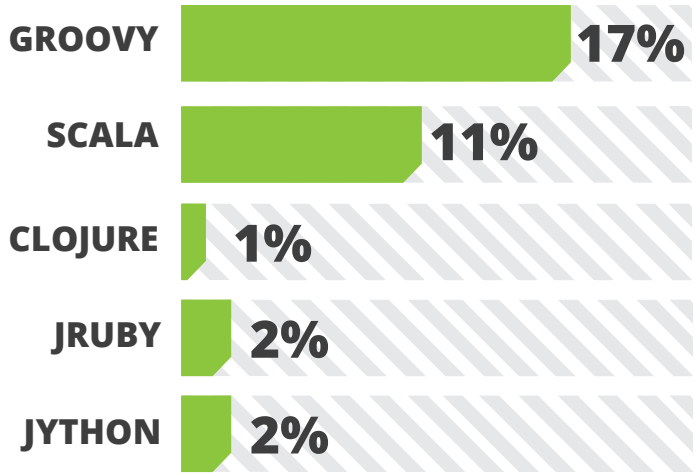
Java 7 has been out for less than a year, and at 23% adoption we were slightly surprised at the speed of uptake. It brought some new small language features and VM improvements, but had some serious issues at release time. It is reasonable to assume that some developers will skip Java 7 completely as they expect the big release of Java 8, which will bring closures and modularity to the language and platform.



Alternative **JVM Languages**

Groovy has been the dynamic JVM language of choice for years and this fact is reflected in the survey results, although we must admit feeling a bit suspicious at the numbers. Groovy is also trying to appeal to fans of static typing with Groovy++, but Scala seems to have established itself as the statically-typed Java alternative on the JVM - a position which newer languages such as Kotlin and Ceylon will find hard to overtake considering Scala's head start.

Clojure, JRuby and Jython remain as rather niche languages, although Python and Ruby are quite popular outside of the JVM. However, considering the plethora of JVM languages out there, Clojure's quick capture of even 1% of the JVM developers is somewhat impressive.



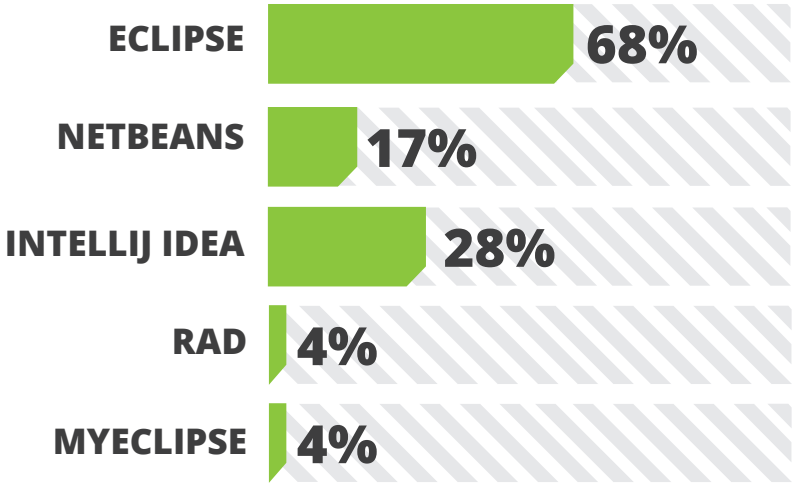
IDEs

The number of Eclipse users has been around the two thirds mark (68%) for a while. It's by far the most popular IDE among Java developers, and receives lots of small improvements each year - the devil is in the details. IntelliJ IDEA has a number of users who swear by it and would never switch to Eclipse, and it seems to have gained more ground as well (28%) compared to our last survey where we saw 22% of Java developers using IDEA. The free Community Edition might have something to do with increased adoption.

NetBeans remains in the third place among the free Java IDEs. It held a roughly 12% market share last year, and has seen some increased use since, potentially after the release of NetBeans 7. We saw NetBeans 7.1 came with many good improvements and 7.2 is coming very soon (presumably, in June 2012). NetBeans seems to be evolving quite actively, which is very positive.

The two commercial options, MyEclipse and IBM Rational Application Developer hold small but presumably dedicated market shares as well. With MyEclipse Blue positioned as a more lightweight alternative to RAD, it will be interesting to see if MyEclipse will be stealing away more RAD users over the next couple of years.

NetBeans uptake increased 80% since 2011

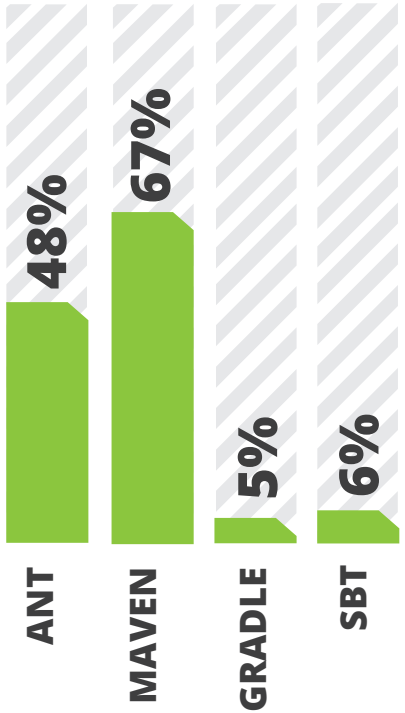


Build **Tools**

There's no beating Maven - although some detest its verbose XML files and ability to download the Internet now and then - two-thirds (67%) of Java developers surveyed use it. Ant is not far behind with almost half of the user base (48%).

This shows that both scripted builds (Ant) and declarative builds with dependency management (Maven) have their place and neither build story will disappear. It is likely that many builds continue to be migrated from Ant to Maven, though, as some previous surveys have shown them to be on more or less equal standing (50% vs 53% in our survey last year).

While in the Alternative JVM languages section, it is somewhat surprising that while more respondents reported using Groovy (17%) than Scala (11%), SBT is slightly ahead of Gradle (6% vs 5%). We guess that comes from the high percentage of Scala projects that are using SBT, and Groovy's use is probably more as scripting language where build tools are not needed.

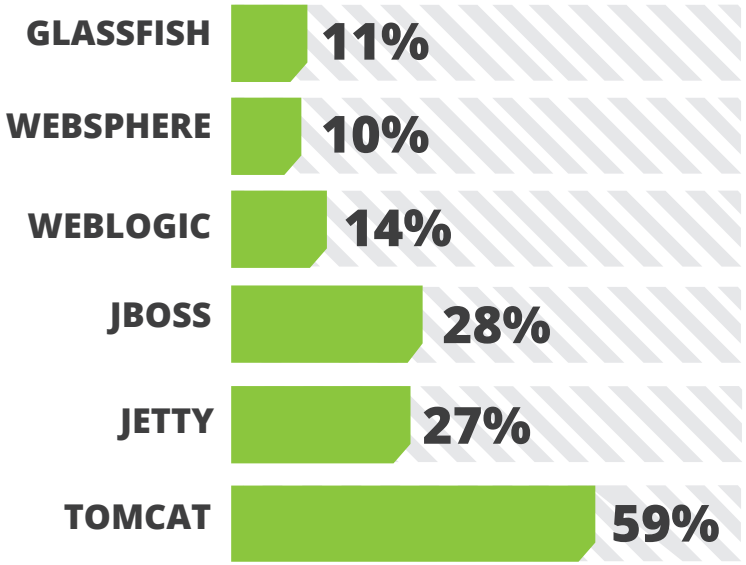


Application Servers **& Containers**

With no surprise, Apache Tomcat remains the most widely-used open-source application server. JBoss remains popular, but the major usage increase we see here is with Jetty, which scored only 8% of users in the last report.

Considering the popularity of open-source application servers, it's makes sense to see Tomcat, JBoss and Jetty filling the ranks, but there are small increases in use for all app servers, and Weblogic is a good example to view. Speaking with customers all over the world, we're seeing more and more the use of multiple application servers in development teams in a single organization. That is, using a lightweight container (Tomcat, Jetty) during development, and deploying to a full-blown application server - like Weblogic - in live environments.

Speaking of the containers, there were a few noticeable releases: Jetty 8, JBoss7, WebSphere 8.5 with Liberty profile, WebLogic 12c and Glassfish 3.1.



Jetty is a great container, and many developers enjoy the fact that it is lightweight, embeddable and you can even get some "enterprise-y" stuff done with it. Jetty 8, first released in October 2011, is the next major version being developed under the Eclipse Foundation umbrella. The main feature is Servlet 3.0 support.

Jetty questions are very popular at StackOverflow and the documentation is quite clean and easy to follow. Combined the facts above, it makes it clear why Jetty is so beloved by developers.

According to the stats, JBoss adoption bumped up since last year. And we think it is due to the awesome startup times for JBoss 7. For the developers who spent a lot of time developing with JBoss 5, whose startup takes nearly a minute, JBoss 7 is a breeze. JBoss 7 is lightning fast compared to the previous JBossAS versions - it starts in nearly 3 seconds. The whole Java EE stack in 3 seconds - that is very cool. For sure, such an improvement couldn't go unnoticed by developers, hence the popularity.

The coolest feature of the upcoming WebSphere Application Server 8.5 release (to be released on June 15th) is definitely Liberty Profile - a lightweight application server with blazing fast startup. If you have promised to eat your hat when WebSphere starts up as fast as Tomcat then you'd better start shopping for a chocolate hat. Liberty Profile implements a subset of Java EE 6 and is small enough to fit in a 33 MiB zip archive. WebSphere die-hards who don't trust such a tiny app server or don't wish to limit themselves with the trimmed down feature set of Liberty Profile, have no fear, you can still get your beloved full-featured WebSphere Application Server for your mission critical applications.

WebLogic 12c has 200 new features and lots of improvements. What we really liked is that it is now distributed as a zip archive. No installers, just unzip the archive and you're ready to go. This is very convenient for development and testing when you want to quickly install a clean instance for an application server just to verify the setup or test the application in a clean environment.

That said, WebLogic is now a bit friendlier to developers. Also the download archive has shrunk quite a bit: only 166MB for WL 12c versus 318MB for the 11g zip distribution.

However, the gained popularity is hardly related to the new version as this application server is mostly used in large enterprises, where the migration process might take a while and new versions of middleware aren't adopted from day one.

Web **Frameworks**

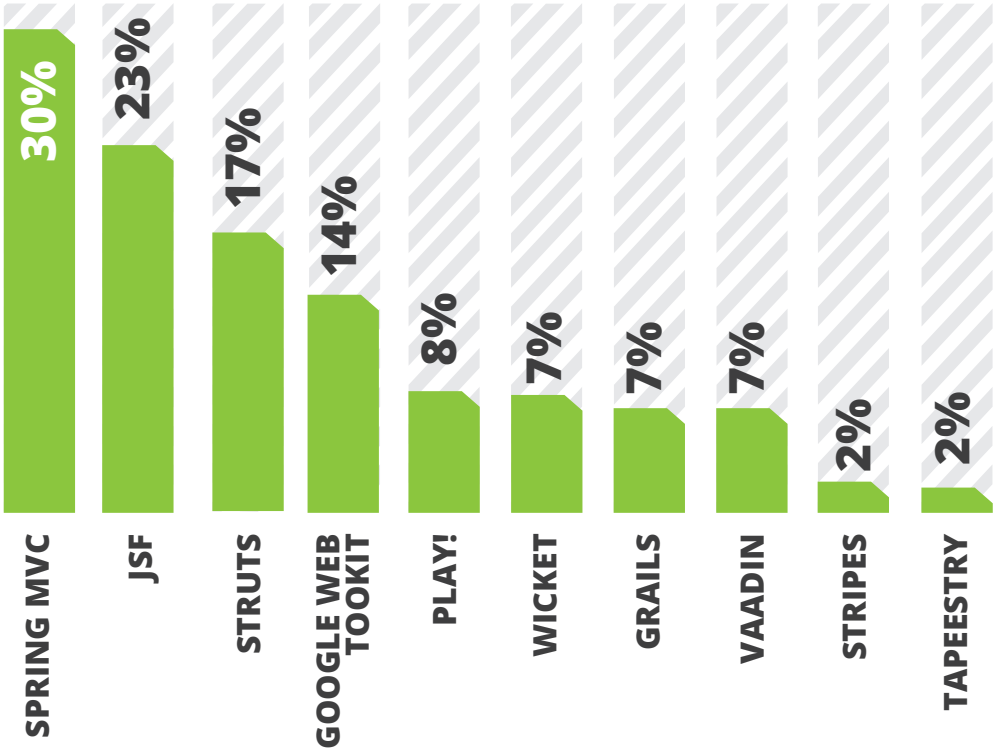
The results of Web Frameworks are sadly very boring, and more or less align with last years results, with only small changes accounted as measurement defects.

As expected Spring is the first with 30% of the market share. 30% is big drop from 48%, but that comes from the separation of Spring MVC and Spring. JSF-s is next with 23% which is 1% lower than last year and can be accounted for a measurement defect. Struts is 3rd this year, but the results of Struts include both Struts 1 and Struts 2 so the sum is the same as last time. The first framework that has risen is GWT with 2% increase to 14% but this is not much.

We omitted Play! and Vaadin (unintentionally) from last year's report, but they both have comfortable midpoints in the distribution.

The lack of large changes in this area can potentially be explained by the fact that changing web frameworks is a somewhat difficult, time-consuming and costly process. There are a lot of legacy, business-critical applications out there, who cannot risk a switch if their application will be broken for a few months due to a huge refactoring.

Any changes in framework usage would therefore occur in new projects where there is freedom in choosing different technologies, but it would seem that new projects have either too little weight or the preferences have not really changed.

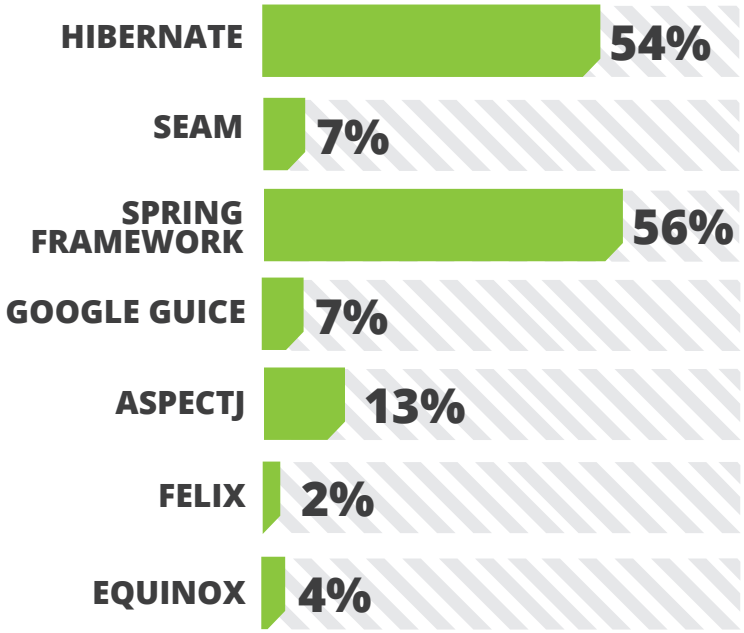


Application **Frameworks**

In the Application Frameworks category, Spring (56%) and Hibernate (54%) are still the most popular. After a big drop comes AspectJ with only 13% of usage. According to the users of this survey other frameworks are not much used.

The % of Spring and Hibernate is almost the same because they are usually used together. And compared to the survey results of the last year (48% and 45%) they are still at the very top. So we expect to see the same results from them in the next year also, and 3rd place will possibly change as well.

Spring and Hibernate are probably so popular because they are relatively easy to use and have a lot of use cases. Like many other actively developed products, they have an active release cycle: the latest version of Hibernate was just released in Q2 2012 weeks ago (Hibernate ORM 4.1.2 on 2012.04.04) and Spring earlier in 2012 (3.1.1 was release on 2012.02.16).



But enough about Spring and Hibernate. Let's take a quick look at AspectJ. The current version is 1.7.0.M1, meaning that 1.7.0.M2 should be around the corner. Last year's survey omitted AspectJ, and now we can see that it is a reasonably well-used framework. We'll be sure to include AspectJ in our next report, and make a better comparison then.

Java **Standards**

Java EE's lesson learned in lightweightness is paying off - both EJB 3.0, CDI and JPA have seen rapid improvements in adoption.

JPA is performing exceptionally well, but that's due to lacking any real competition. JPA is supported by both Hibernate and EclipseLink and is easier to set up than "pure" Hibernate, for example.

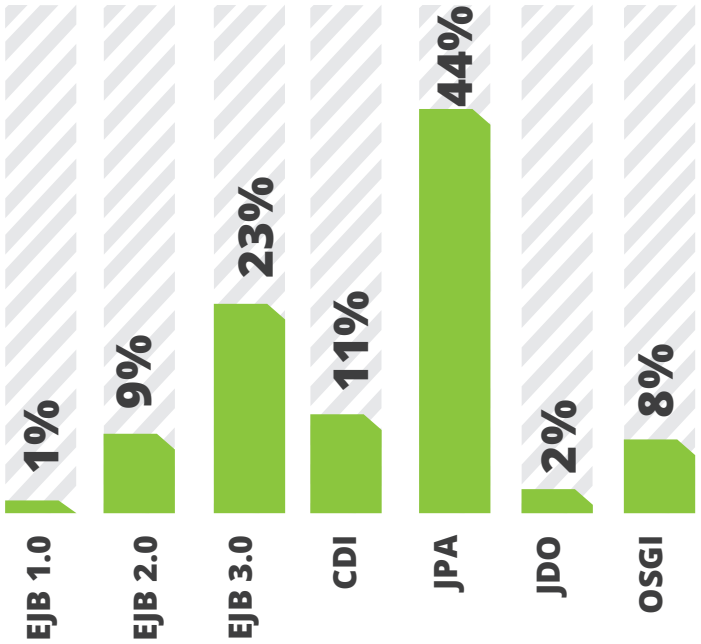
Talking about competition, JDO is light-years behind after lacking a good implementation for years and Spring JDBC templates are too disparate to be compared. In fact, JPA's share might be even bigger, because many developers might be using Hibernate with JPA annotations but without EntityManager.

CDI is aggressively taking ground from Spring in the dependency injection arena, almost doubling its market share.

While EJB 2.0 is steadily going away with its distributed transactions, the continued presence of EJB 1.0 teaches us a lesson that programs we write today will be hanging around as legacy software for many, many years in the future.

We don't have a direct comparison with OSGi adoption in previous years, but it seems like the 10-year-old system has carved out a zone for itself and is not seeing a lot of growth. IDE's and application servers seem to be the ideal use case for it, but for general application development it seems to be remaining a relatively niche solution, probably due to the uncomfortable switch required to bring OSGi into a development team.

It will be interesting to see whether Jigsaw (the Java 8 module system) will make modularity in general more popular and whether it will affect OSGi.



Continuous **Integration Servers**

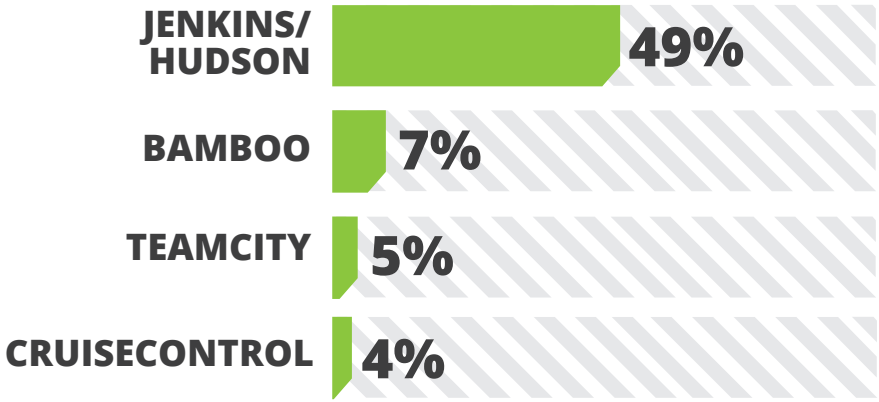
These results show Jenkins/Hudson to be the *de facto* standard method for building a Continuous Integration environment. The reasons behind that are being free, open-source, cross-platform, more or less user-friendly and easy to install.

This picture will not likely to change in the nearest future as the community of Jenkins/Hudson is actively working on fixing bugs, implementing new features and writing new plugins.

Commercial solutions, such as Bamboo (7%) and TeamCity (5%), have their own small niche and are continuing to gain ground in light of the Jenkins/Hudson rift, and the value propositions offered by paid products. Satisfied customers from Atlassian and JetBrains will trust tertiary products from the same companies;

Bamboo integrates very well with Jira, and TeamCity with IntelliJ IDEA, for example..

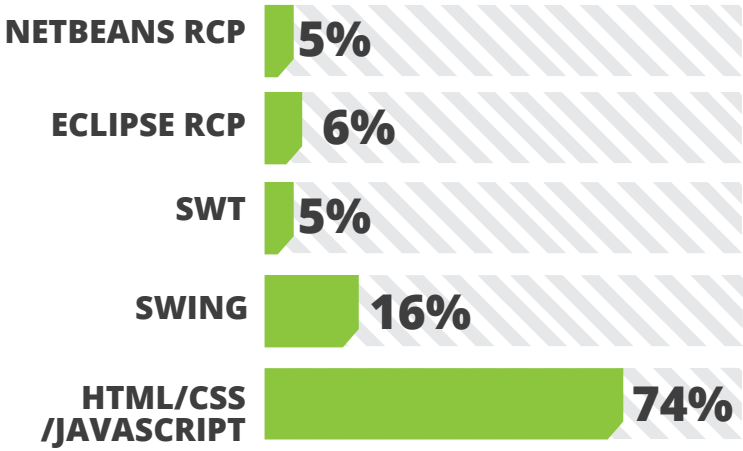
Every CI server has it pros and cons and if you have some doubts which one to choose, we suggest getting started with Jenkins/Hudson as the safe bet.

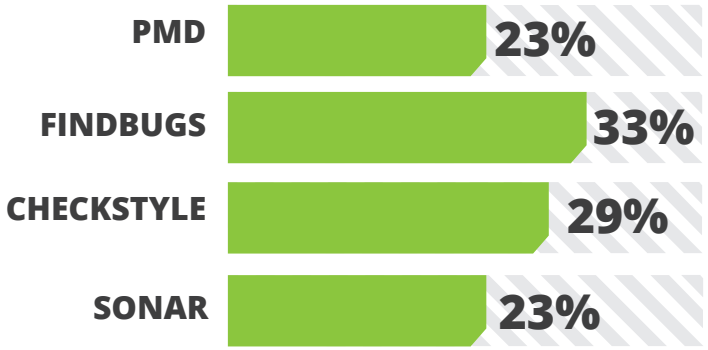


Front End **Technologies**

There are very few surprises in the answers to our front end technology section of the survey. We can tell from these results that Java is still very strongly positioned for web application development and speaks well to Java's continued success as the primary language and environment for enterprise development as the trend of moving applications to the cloud continues.

When we look to the other responses in the section, we see that RCP is becoming increasingly more important and used as a front end technology. If we combine the data points from NetBeans and Eclipse RCPs we see RCP beating SWT and almost overtaking Swing. This speaks to Swing's continued downward usage trend and the growing importance of modularity throughout the application's code base.





Code Quality **Tools**

This is the first year we asked about code quality tools. Findbugs and PMD look for bugs in your code and Checkstyle checks that your source code adheres to your coding rules. Sonar is actually a code quality tools suite, which is somewhat of a different beast, and includes the aforementioned tools, offering the same results but observable as trends over time. In the next report, we also plan to include code coverage tools, like Cobertura and Emma.

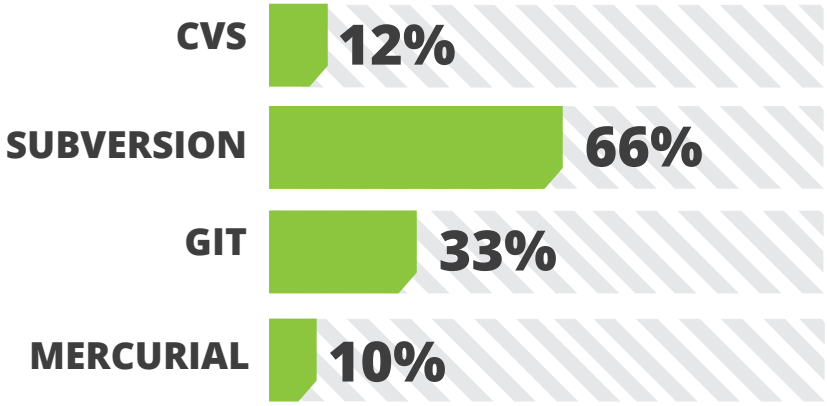
From the results we can't say much more than the Java ecosystem definitely trusts static code analysis and uses tools to find "mistakes that matter" early. The market appears to be spread more or less evenly, with Findbugs, the well-known University of Maryland project, in the lead. Each tool (all these tools are complementary of each other) is used by a nearly quarter of our respondents.

Version Control **Systems**

As has been the case for several years now, Subversion is by far the most-used version control system for Java development. This is generally because of the reasonably smooth transition between CVS (the historical titan of version control) to Subversion.

The most interesting part of the result set is how prevalent Git and Mercurial are becoming. Git is most people's introduction to a distributed version control system. Subversion still has the edge due to the extensive histories people have in their existing subversion repositories. As the tooling for Git improves and the culture around source control moves to a distributed paradigm, it is likely we will continue to see CVS and Subversion slowly giving way to Git and Mercurial.

Keep an eye out for Git



Part II
DEVELOPER TIMESHEET
It's 3pm. Do you know where your dev team is?

How do developers spend their work week? What are they actually DOING all day?

Among lots of other information that we unearthed from the depths of the coding den, we wanted to introduce these 3 interesting findings, and get some running commentary from Lincoln Baxter III, Senior Software Engineer at Red Hat (JBoss Forge), founder of OCPsoft, and opensource author / advocate / speaker:

Finding #1:
Devs spend less time writing code than you might think. The median is 15 hours, programmers spend about 3 hours each work day writing code.

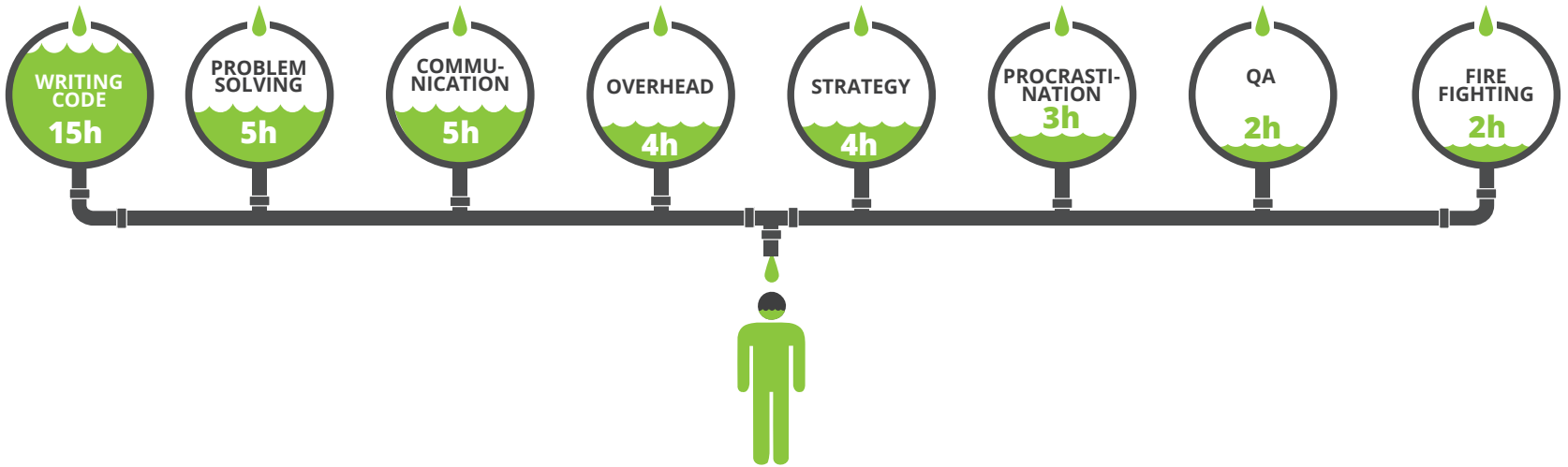
Finding #2:
Devs spend more time on non-development activities than you might think. For each coding hour, devs spend nearly 30min in meetings, reporting, writing emails and dealing with timesheets (7 hours to 15 hours)

Finding #3:
Devs spend more time fighting fires than building solutions

Disclaimer: As with any type of research undertaking, here are a couple of disclaimers: a) We asked respondents to estimate the time they spend doing activities, so 100% accuracy is not guaranteed. b) Not only developers responded, but also architects, QA folks and system admins, so averages may reflect this too.

How do **Developers spend** their work week?

Check out this breakdown of what devs spend their time doing (the top 3 time-consuming activities are marked):



Here are a breakdown of the categories a bit more. We tried to get everything a dev might do during the week, so if you can think of anything we missed, it would be cool to hear about that.

- **Writing Code (programming, coding, hacking away)**
- **Overhead (Building, Deploying, Hardware, Software)**
- **Communication (Meetings, Chats, Teleconferences, etc)**
- **Problem-Solving (Debugging, Profiling, Performance Tuning)**
- **Firefighting (Crashes, Slowdowns, Security, etc)**
- **QA (Manual & Automatic Testing, Code Reviews)**
- **Strategy (Architecture, Refactoring, Thinking)**
- **Process (Bureaucracy, Reporting, Time-keeping, etc)**
- **Procrastination (slashdot.org, reddit.com, Twitter, Facebook)**

We can see that actually writing code is still the dominant activity of the day, but by no means a majority by itself. There is a distinction between “writing code” and “coding”; “Coding” includes writing code + problem solving, QA, strategy etc.

If we assume a 45-hour work week (9 hours per day, incl. lunch and breaks), developers spend 1/3 of their entire work week producing code.

Extended **Interview** with Guest Geek **Lincoln Baxter III**

Linconln Baxter III, Senior Software Engineer at Red Hat (JBoss Forge), founder of OCPsoft, and opensource author / advocate / speaker

ZT: Hey Lincoln, thanks for joining us.
LB3: Glad to be here.
ZT: Can I ask....does it say "The Third" on your passport?
LB3: No, just the Roman Numerals.
ZT: Darn. Anyway, what do you think about Finding #1: Devs spend less time writing code than you think? I mean, according to this, devs spend less than 3 hours each work day writing code?
LB3: To be perfectly honest - and I'm always perfectly honest - I'm not surprised one bit. The biggest drain on productivity is the constant interruptions to our concentration; that can be co-workers, running a build, running tests to check your work, meetings.
It can take up to 25 minutes to regain your focus once you've been distracted from your original task#, and if like me, you consider "writing code" to be the task that developers should be focus on, then the size of other pieces in your pie chart begin to make a lot of sense.
Think of a brain as if it were a computer. There is waste every time a computer shifts from one activity to another (called a context switch,) a very expensive operation. But computers are better at this than we are because once the switch is complete they can immediately resume where they left off; we are not so efficient.
When considering context switching, builds and everything else considered "Overhead" are the biggest distractions from writing code. Even though this piece of the pie is only responsible for 4.63 hours of a developer's week, in reality, the true impact of this problem is much greater.
Once you add in all of the other distractions of the workplace, I'm impressed anyone gets work done at all. Every re-deployment is going to cost you an extra 25 minutes of wasted focus, in addition to the deployment cost itself.
ZT: Alright, on to Finding #2: Devs spend more time on non-development activities than you might think. Based on these numbers, for every 1 hour devs spend writing code, they spend over 30 minutes in meetings, reporting, writing emails and dealing with timesheets (8.25 hours to 14.75 hours). Has it always been like this, or is this something new coming as more IT shops join forces with large enterprise and corporate parents?
LB3: In a desperate attempt to measure and estimate the work that programmers do, we try more and stranger ways to try to gain some kind of understanding.

Think about it - in every other type of engineering field, we perform estimates up front, send out quotes, sign a contract, and work follows.
When writing software, however, try as we might, we still have difficulty with this process. Why? Because writing software today has very little to do with engineering.
The cognitive requirements for programming (software engineering) are much more akin to those of composing music, or painting a picture than they are to building a bridge or installing a drainage culvert. The common misconception about us "geeks" is that we are boring, uncreative and dull, but if that's true then Mozart, Beethoven, and Daft Punk are just as dull.
So why do we insist on measuring software development as if it were an engineering science? We can apply these statistics to bridge building because we have built bridges for thousands of years, and the laws of physics have not changed since then. The laws of computing, however, change daily, and we cannot possibly hope to measure the same way. This is why the Agile software methodology has had so much success, because unlike when building a bridge, the type and amount of work changes frequently in software development. Agile focuses less on measuring how much work on a task has been completed, but instead how much work remains, and how that relates to how much work can be done during the project timeline.
In fact, at OCPsoft we're working on a new open-source agile project management called SocialPM, which is being designed specifically for the purpose of creating as few distractions as are necessary, while helping teams stay organized and keep more accurate schedules in an environment prone to change. (You know, to keep managers happy.)
Traditional estimation techniques, they've got no hope until we have more standard practices for software development. Let's give our guys a break eh? Stop with the time-sheets.

ZT: Cool. Ok, so now to our last finding. Finding #3: Devs spend more time fixing problems than creating solutions. Let's pretend the non-coding, non-communicating part of the dev day could be split into two parts--Firefighting and Building. Firefighting includes emergencies plus problem solving like debugging and performance tuning. Building includes code reviews and tests, plus strategic planning, daydreaming, refactoring and thinking over architecture.If we break down the work day into parts where devs are spending time actively solving problems or working towards creating solutions, devs spend more time putting out fires.

- **Firefighting - Median 7 hours per week**
- **Building - Median 6 hours per week**

If we use science, we can see that Firefighting consumes 16% more time per week than Building. What can you see from this?

LB3: To be quite frank, we're all lazy, and not nearly as smart as we think we are. With the level of complexity in today's software steadily on the rise, with layers upon layers of software depending on software, there's no hope for us to understand every possible condition or scenario, and thus there is a huge lack of real testing in our field.

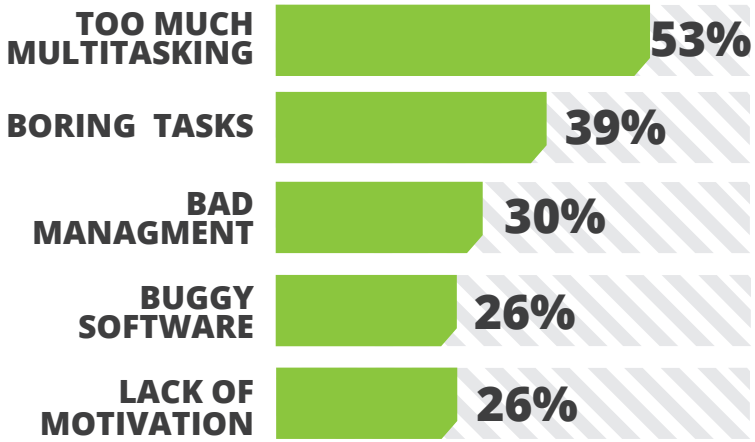
Test driven development has been a hugely overlooked part of our world, not to mention my previous jobs. There are two parts of your job that should be so simple, you don't have to waste any time on them.

#1 - Writing a test.
#2 - Running that test.

We have hugely complex builds that are not properly modularized, and certainly not very testable because how can you test without a database? How can you test without real business logic?If you don't have automated builds and automated test suites that run in a reasonable amount of time (a few seconds or minutes,) then you are going to spend a good amount of your time running builds and trying to write and run tests; not to mention, you may not see the real issue until you deploy the software to a production environment, because mocks can never hope to replicate a real system.
In terms of Java, we're seeing new advances in the field of testing - namely a project from JBoss called Arquillian, which allows us to automate the deployment of real tests to a real environment in which all services are available. The fact that you have access to your database, business services, and more is why "Don't mock me," is the Arquillian slogan.
If there's one thing that surprises me about this statistic, it's that we don't spend more time fighting fires, because without a good automated build and a solid suite of real tests, we're going to have problems.

Part III
DEVELOPER EFFICIENCY
Even the Death Star had a project plan...

What aspects of developer life makes you more/less effcient?



The majority of respondents felt that Too Much Multitasking was the primary reason for not getting work done. Over 1/3 of devs also mentioned that Boring Tasks where responsible for under-efficiency in the cubicle. This begs the question whether or not it is better to get "boring" tasks over and finished as quickly as possible, but we all understand how feeling uninspired to attack your work with enthusiasm will lead to procrastination.

It is fair to conclude that Boring tasks, Bad management and Lack of motivation are indicative of a sizable gap in communication between management and development teams, or, more interestingly, a lack of self-management for organizing one's time, one's mind and one's life.

Extended **Interview** with Guest Geek **Matt Raible**

We asked Matt Raible, Web architecture consultant, frequent speaker and father with a passion for skiing, mountain biking and good beer, to give us some of his insight on the factors that keep developers from finishing their work, so here we go!

ZT: Hey Matt, thanks for joining us. Hey, I heard this rumor that you are mostly Irish-Montanan?

MR: Yep, I grew up in the back woods of Montana with no electricity and I'm mostly Irish. So keep that in mind when asking me how I get things done at work.

ZT: Understood. So what do you think about issues regarding "Too much multitasking"?

MR: I've got a couple of suggestions that work for me:

- Work disconnected. To further facilitate not checking e-mail or reading blogs, I've found that going to a coffee shop w/o connectivity is my most productive environment. They have liquid motivation in the form of coffee, and you can feed your brain with breakfast/lunch or some kind of snack. My most productive days are the ones where I show up at my local Einstein's (bagel shop) at 6 a.m., have two cups of coffee, and work with my headphones on. After the coffee and uber-productivity, I often have an awesome ride to work and barely notice the miles. NOTE: I've found that I'm more productive writing code late at night and authoring articles/books in the early morning.
- Stop reading e-mail, Twitter and Facebook. One of the ways I can tell I'm in uber-productive mode is my unread (or starred) mail piles up and I haven't read any blog posts (or blogged myself) in a couple days.
- Sleep. While working late nights can be productive in the short term, doing it consecutively will burn you out quickly. Getting a good night's sleep can often lead to greater productivity because you're refreshed and ready to go.

ZT: Yeah, sleep is nice. *yawn* .... Sorry, where were we? Ah, "Boring Tasks".

MR: Some tasks are boring and there is nothing really to do about it. I find that music can make a big difference and potentially grow some inspiration where it didn't exist before. If you've got outside projects that you find more interesting, position them at the right time of day.

Listen to music while you work. Some noise-cancelling headphones and your favorite music can do wonders for your productivity. Of course, earbuds work just as well - whatever makes the music sound good. Good music can really help you "get into the groove" of what you're working on, regardless of whether it's

writing or coding.

Work on open source late at night, with a beer on your desk. While I sometimes get the opportunity to work on open source at my day job, I still find that I'm most productive at night. Maybe this is because no one bugs me via e-mail or IM, or maybe it's just because the world is asleep. The strange thing is I often find myself motivated at 3 p.m. for my 11 p.m. workload. However, when I get to 11 p.m., I'm not motivated to work on anything. I've found that cracking open a beer at 11 when I start helps me focus and quit worrying about all the other computer-related tasks I need to do.

ZT: What can you say about "Bad Management"? In your case, you deal with a lot of self-management for your time and resources...what can you tell us that would apply to both contractors and those of us working in teams?

MR: Yeah, what works great for me is to get used to non-standard work hours, and avoiding inefficient wastes of time.

- Work long hours on Monday and Tuesday. This especially applies if you're a contractor. If you can only bill 40 hours per week, working 12-14 hours on Monday can get you an early-departure on Friday. Furthermore, by staying late early in the week, you'll get your productivity ball-rolling early. I've often heard the most productive work-day in a week is Wednesday.
- Avoid meetings at all costs. Find a way to walk out of meetings that are unproductive, don't concern you, or spiral into two co-workers bitching at each other. While meetings in general are a waste of time, some are worse than others. Establish your policy of walking out early on and folks will respect you have stuff to do. Of course, if you aren't a noticeably productive individual, walking out of a meeting can be perceived as simply "not a team player", which isn't a good idea.

ZT: One of the responses was about "Buggy software". Now, this could refer to buggy software that is being used or developed, but I rather think the latter.

MR: Yeah, I would assume they are talking about productivity inhibitors they use professionally. Well, I could make a suggestion that Java developers might enjoy - start using IntelliJ IDEA instead of other IDEs :-)

ZT: We'll actually get to see how popular IntelliJ IDEA is compared to other IDEs in the final Developer Productivity Report 2012 ... finally, Lack of motivation was cited as another factor preventing developers from being more efficient. Any final thoughts on that?

MR: Look, you have to work on something you're passionate about. If you don't like what you're doing for a living, quit. Find a new job as soon as possible. It's not about the money, it's all about happiness. Of course, the best balance is both. It's unlikely you'll ever realize this until you have a job that sucks, but pays well. I think one of the most important catalysts for productivity is to be happy at your job. If you're not happy at work, it's unlikely you're going to be inspired to be a more efficient person. Furthermore, if you like what you do, it's not really "work" is it?

Part IV
DEVELOPER STRESS
Something keeping you up at night?

The good news is that the most common response to What keeps you up at night? was “Nothing. I sleep like a baby”. But it was followed very closely by some aspects of development that warrant a deeper look.

The following 5 stressors were listed as the top things keeping developers up at night.

1. **Making deadlines - 25.00%**
2. **Application performance issues - 24.10%**
3. **Is my code good enough? - 23.70%**
4. **What new stuff do I need to learn? - 23.50%**
5. **Did I do X in the best way? - 22.50%**

Developers are quite concerned that their code is correct, innovative and following best practices. It would seem that training and continuing education is also an important factor that gets a lot of thought. What can companies and teams do to make sure devs are able to feel confident about their code, level of expertise and understanding of best practices?

”Usually I sleep like a baby”

Extended **Interview** with Guest Geek **Martijn Verburg**

To get some valuable perspective on the matter, we've invited Martijn "The Diabolical Developer" Verburg, Java community leader, speaker and CTO at TeamSparq

ZT: Hey Martijn, thanks for being here with us.

MV: It's a pleasure. I was just thinking to myself, "Hmm, I've got sooo much extra time these days, I wonder if anyone wants to interview me!"

ZT: Wow, really?

MV: Nah, I'm totally slammed actually.

ZT: Nice one. So Martijn, as a simple marketing droid I was surprised to see that developers are primarily concerned about Making Deadlines. Isn't that something The Suits should be worrying about more?

MV: Managing deadlines is something that a lot of developers feel that they cannot learn or is out of their control (e.g. Their manager tells them what the deadline is). However, managing deadlines is a skill that can definitely be learned! For example, developers can learn to:

- Scope work into manageable (e.g. 1 day) chunks
- Define what _done_ means (95% is not _done_)
- Factor in contingencies
- Communicate risks and issues to stakeholders
- Learn to prototype ideas to keep the overall project flowing

There are a number of tools to assist you in managing the scope and communication around deadlines, but always remember, "Whatever they tell you, it's a people problem", so developers should look at their communication and expectation setting first.

ZT: I think that developers can definitely take steps to empower themselves a bit more in this process. Can you recommend any specific tools tools or techniques for managing deadlines better?

MV: Absolutely. For example:

- Simple Lean/Kanban style planning with sticky notes and a whiteboard
- Electronic versions of the above (e.g. Atlassian's Greenhopper)
- BDD style tests with an electronic wallboard (therefore the whole business can see exactly where progress is at)
- A publicly shareable issue tracker that tracks timings (e.g. Atlassian's JIRA)

ZT: I'm sure devs will find that helpful. Next, why do you think Performance Issues would rank so highly on the list of developer stress?

MV: Performance and performance tuning is terrifying for most developers because they have no idea where to start. Modern software applications are so complex that it can feel like finding a needle in a haystack. However, performance and performance tuning

is actually a _science_, not an art and definitely not guesswork. Kirk Pepperdine (one of Java's foremost expert in this field) always hammers home the point "Measure, don't guess".
By following the sort of scientific methodology that Kirk and others like him teach, you can systematically track down and fix performance issues as well as learning to bake performance in right from the beginning. There is a host of tooling that can assist in this area as well, but it's the methodology that's truly important. I can highly recommend Kirk's course (www.kodewerk.com - he runs courses worldwide, not just in Crete, so drop him a line).

ZT: Have you ever been asked if you think your Code is Beautiful (or simply "good enough")?

MV: This is a question that all developers secretly fear. The fact is you can't definitively state what good code is! The Softwarecraftsmanship folks will beg to differ here ;-). Define what Clean Code is! At the end of the day there are many differing opinions on what defines 'good code'. However, I'd suggest that there are some clear traits of what is commonly perceived as good code:

- It's broken up into small classes/ methods/blocks each that perform one thing and one thing well
- Some level of interface is used so alternative implementations can be used
- The code is easy to test against
- You can ascertain by the naming used in the code, what the code is used for, e.g. it's problem domain
- Common language pitfalls are avoided - type1, the sorts of things that static code analysers warn you against
- Common language pitfalls are avoided - type2, the sorts of things you pick up in Josh Bloch's "Effective Java" title.
- There's more, the list is fairly long.

So my advice is to try to follow the better practices out there (note I avoid using 'best practices', horribly overloaded term), make sure that you're using the static code analysis tools, try to practice TDD and above all else get a 2nd pair of eyes on the code, preferably via Pair programming to begin with.

ZT: Is it fair to suggest that when devs worry about their code being good enough, it might speak to their next concern about Learning New Stuff?

MV: There's always the fear of being left behind, can you as a developer guess what's going to be big next? If your technology CV is out of date you might struggle to find that next job!
Take a deep breathe and relax. Most new technologies are simply short lived fads and it's impossible to keep up with everything!
There's no way a developer can stay up to date with the latest Java libraries, learn Scala, pick up Clojure, hack some iOS, and then deploy all that to the cloud on a NoSQL distributed grid environment. See what I mean?
The trick is to identify trends. Cloud is a trend, so you should learn about the principles behind it, but don't sweat if you haven't learned to deploy to the 5+ different Java cloud providers out there today. Functional programming is a trend, so learn _why_ (hint - Multi-core processors and concurrency) it is and see if it's something that you need to learn

about now or whether you can wait a few years. The same goes for any new craze you read about or hear at a conference. At the end of the day, our industry spends a lot of time re-inventing the wheel, and we have memories like goldfish :-). For example, some of the older hands are certainly chuckling away at this new functional fad, they were doing it over 20 years ago...

ZT: I think it's fair to say that any professional will be concerned about doing X in the "Best Practices" way. You mentioned above that you dislike how "best practices" is poorly used. As a long-time coder yourself, how do developers feel about this?

MV: This is definitely related to developer concerns over "beautiful code", but I'm going to assume this is more of a fear at the macro design level. Developers can fear that if they chose the wrong web framework or picked the wrong app server or designed the n-tier architecture wrong that they're doomed.

A common problem here is that developers aren't standing on the shoulders of Giants, most problems are not actually that unique and there is a wealth of free advice on sites like programmers.stackexchange.com, on mailing lists, in books, at conferences and of course at your local user groups.

Another common problem is in the prototyping space, not enough developers prototype the risky parts of their projects soon enough. Are you worried for example that JBoss might not support AMQP? Well, spend a day upfront proving that it does or doesn't!

THAT'S ALL FOR NOW.

CHECK OUT ZEROTURNAROUND.COM/BLOG FOR MORE CONTENT FROM THE ZEROTURNAROUND TEAM.

ZEROTURNAROUND