# Transitioning to Scala

Advice from a developer who helped rebuild Walmart.ca with Scala and Play

From late 2011 to early 2014 I was a lead developer at Nurun Toronto, a digital agency focused on e-commerce solutions. Facing a pipeline of new projects with nothing but Spring in one hand and Java in the other, I realized that it was time to quickly explore alternatives.

Working at an agency puts things in perspective. Fucking up a contractual deadline because “Java sucks” isn’t going to impress clients. Using the same tools and techniques we used to build applications back in 2004 won’t help us build applications in 2014. In 2004, restarting an app server to test a single line of code was perfectly acceptable—WebSphere’s average *120 second* restart time was considered a good opportunity to grab a coffee while pondering the mysteries of a 1,200 line XML configuration file. Today that kind of waste will put an agency or a developer out of business.

Our applications are no longer simple forms sitting in front of database tables, they're invaluable tools that help us get shit done in our daily lives.

## Why the Typesafe stack?

After implementing a past project with Ruby on Rails I knew what I didn't want; I didn't want to use another dynamically typed, interpreted language or another stateful web framework. I wanted a statically typed language that compiled to Java bytecode and came with a healthy ecosystem of tools and a stateless web framework for scalability. I also knew that our corporate clients would require the peace of mind only possible with professional support contracts through a reputable vendor.

This left me with a few choices to explore, the most intuitive being the Typesafe stack. It had everything we needed.

What started as a proof-of-concept micro e-commerce site to internally demonstrate the simplicity of Scala code and developer productivity of Play eventually gained traction and became the foundation of Walmart Canada's new e-commerce platform.

## Why is Scala perceived to be complex?

Scala is *flexible*. Flexibility comes at the price of simplicity, but in the right hands Scala is a very elegant language that is much more than just a "better Java".

The biggest challenges of transitioning to Scala — or *anything* new — are rarely technical. Talented programmers can learn a new syntax, new concepts, and a new IDE. Rather, change often brings out the weaknesses in other areas like process and culture.

*tl;dr — it's all about people.*

The rest of this article is not a Scala programming tutorial. There are already plenty of those, and I'm not the most qualified Scala developer in the world to teach you the latest tricks from the deepest pools of advanced Scala.

What follows is a collection of advice geared towards developers, team leads, and managers who are thinking of transitioning their teams to Scala. The advice is based on my own personal experience as a team lead on an enterprise

Scala project.

## Advice for developers and managers thinking of making the switch to Scala

*Scala. Better than a hot cup of joe!*

### 1. Understand the language features

Every new Scala developer, team lead, and manager should read Martin Odersky's Scala levels guide.

After a year and a half of full-time Scala development and one major enterprise Scala project under my belt, I'm not afraid to rank myself as a level A2.5/L1.5 Scala developer according to Martin's guide. I've used techniques defined in A3/L3, but I simply haven't had the need to use the majority of them on the relatively straightforward task of building web applications. I've yet to use the cake pattern or learn about higher-kinded types. This doesn't make me a bad programmer or bring up any intense symptoms of impostor syndrome, it just reflects that I have a limited amount of time in my life and I need to focus on learning what will give me the most bang for my buck. I also play drums, guitar, take two dance lessons per week, drink a lot of coffee, and go out on dates. Time is precious.

On the Walmart.ca project, with the exception of combinator parsers and folds, we stuck to techniques from the shallow end of the level guide. Even with "shallow" Scala we produced a significantly better platform than the predecessor it replaced. We didn't run into any implementation headaches. The code we wrote is infinitely more maintainable than the code it replaced. It runs significantly faster in production. It scaled perfectly fine on both Black Friday and Boxing Day, something many Java-based e-commerce platforms can't claim.

Isn't that what counts?

So don’t make the mistake of dismissing simple Scala as lesser Scala. Look at all the tricks you can master at levels A1 and A2:

- Simple closures
- Collections with map, filter, etc
- Pattern matching
- Trait composition
- Recursion
- XML literals

This is way more than Java with a few extra goodies. This is a new way to write expressive, maintainable software.

Some of the techniques at A3 are still fairly easy to master—and pretty important for those using Akka or other parallel computing libraries—but not insurmountably hard or requiring a degree in math to understand:

- Folds
- Streams and other lazy data structures
- Actors
- Combinator parsers

A pleasant side effect of mastering these techniques is that you will become a better programmer in **all** of the languages you work with. I'm a much better JavaScript developer now that I've had a chance to really master closures and other techniques that are a part of both Scala and JS.

## 2. Take your time

A lot of Scala developers come from Java looking for immediate gratification but don't fully appreciate that Scala is a completely different language. Everything new needs practice to master. Scala is no different.

The good news is that A2/L1 Scala developers are fully qualified to work on typical Scala applications while even providing mentorship for junior developers. Not every Scala developer needs to understand advanced purely-functional data structures, type theory, or higher-kinded types.

Remember that Scala is a programming language for professional developers building next-generation applications. It will take time to learn and even more time to master.

## 3. Don't be afraid to study

If you're a Java developer, I strongly recommend exploring Scala learning materials in the following order:

- Read Scala for the Impatient. It's a good starting point for imperative developers, especially if you *do the exercises*. I still comb through my original answers and refer to them when I get stuck on something.
- Read Programming in Scala by Martin Odersky, Lex Spoon, and Bill Venners. It's a more detailed read than *Scala for the Impatient,* so it's nice to go through after having a broader view of the language features.
- If possible, take Coursera's Functional Programming Principles in Scala course. Coursera uses Scala themselves!
- Explore the Typesafe Activator templates. Online documentation and learning materials for Scala are sparse compared to other languages, but there's no better way to learn than hacking on someone else's code—especially if the code is written by talented developers like James Ward.
- If possible, take Coursera's Principles of Reactive Programming course. This is a great resource for developers who would like to explore the use of Scala and Akka for processing large volumes of data.
- Attend Scala meet-ups and learn what other working programmers are using Scala for!
- Read Functional Programming Patterns in Scala and

Clojure. After seeing first hand how much more legible my functional style code looks compared to my old imperative style code, I'm thirsty to learn more even more functional languages. Haskell is a bit too much for me, but I've begun teaching myself Clojure. My Scala code is becoming leaner and meaner by the second.

## 4. Take what you read online with a grain of salt

Incredibly smart developers such as Tony Morris, the creator of Scalaz, come from a different world than I do; the world of Haskell, functional programming, and mathematics.

He argues against function signatures such as:

```
def reverse[A]: List[A] => List[A]
```

In favour of function signatures such as:

```
def <-:[A, B](f: A => B): List[A] => List[B]
```

*The next time you see a function named <-: and you think to yourself, "OMG, how unreadable, what am I going to do now!?", ask yourself if there are other tools — perhaps more* *robust than those familiar — to comprehend the code. What is its type? What are its algebraic properties? Are there parametric properties to exploit?*

That's the beauty of Scala. Tony is right. I am right. There's no wrong, there's simply a matter of personal and team preference. I prefer ***reverse*** over <-**:** any day, but developers such as Tony will strive for mathematical purity and truth just as I will strive for simplicity and readability. Our styles will likely always differ. Tony writes libraries, I use libraries to build applications, and we both use Scala to do so. I don't mind using the occasional *var* and coming back later to clean it up, but such is a travesty to some.

However, suppose your team speaks a variety of different natural languages, and not just a single language such as English or French or German? Symbolic function names may make perfect sense over English verbs in a situation like this, even if they are only aliases of natural language verbs. As a Canadian I've worked on applications with both English and French verbs used throughout, and I can assure you that it was even less pretty than our flappy heads and beady eyes. *Sacré bleu!*

That's why un-opinionated languages such as Scala are so beautiful. You are free to apply your own style, the language doesn't get in the way. I love that defining <-: is *possible*.

## 5. Leverage professional services if within your budget

I was lucky enough to learn from true expert developers such as Nilanjan Raychaudhuri and Roland Kuhn from Typesafe, who Nurun brought in for design reviews, code reviews, and pair programming. They provided invaluable assistance at multiple phases of the project which really boosted our confidence level as we learned a new style of programming.

Not only were we learning functional style Scala, but we were also learning reactive programming concepts. We were also ramping up on both Play and Akka. With all of the new technologies at play—no pun intended—it was a huge boost to have Typesafe's assistance throughout the project. It gave us the confidence that we were always on the right track.

Typesafe's email support was also out-of-this-world responsive. A support subscription is well worth the expense if within your budget.

## 6. Embrace diversity

The Scala community is truly the most diverse collection of programmers in the world. Some are former Java developers like myself, others come from the world of academia. Some are self-taught programmers, others hold doctorates. Some are trying to solve business problems on a tight budget, others are trying to push the boundaries of computing. Some are writing applications, others are writing libraries.

Understanding and respecting everyone in the community is important to becoming a good developer. It may seem as if a simple question on StackOverflow leads to baffling replies that require years of category theory study to understand, but keep in mind that Scala is a new language and the community is still finding its voice. As more non-academic programmers move into the world of Scala and answer more questions, discussions may become a little more focused on the *application* of concepts rather than the *theory* behind concepts.

If you ever become discouraged, look at the source code of libraries such as Finagle by Twitter or Smoke by mDialog. Finagle and Smoke are excellent examples of practical Scala implementations used in production environments today. Not all Scala is over-the-top complex.

## 7. Set realistic expectations

New Scala developers coming from Java should accept the fact that they will not learn advanced, functional Scala overnight. Nor is advanced functional Scala required to be successful with typical business application programming.

New Scala developers coming from a functional background will have an easier time adjusting as Scala naturally promotes functional style. However, experienced functional programmers will have to tailor their style if working on a team with mostly imperative programmers.

It's all about team balance and making sure the code you write is maintainable by the people who will be responsible for it. The most elegant code in the world is useless if it's not maintainable.

## 8. Pair programming and code reviews are mandatory

Pair programming is almost essential to make sure Scala code doesn't become too divergent from the average skill set and style of the team as a whole. The last thing you want is code so advanced that mere mortals can't touch it, or a single functional programmer going off on a tangent by rewriting perfectly working imperative code before the team as a whole is ready for that leap. Experimentation is important, but that's what Git is for. Fork baby, fork.

*Don't let this happen to your team.*

Because of the flexibility of Scala, it's easy to slip over the edge of complexity and into unfamiliar territory as developers start to unlock the expressive power of Scala and explore new language features.

Culture is always important on a development team, but it's extra important for a new Scala team to move together in tandem when first learning.

## 9. Keep it simple

Scala is new, so people are still learning what language features to use and what language features to avoid. Until Douglas Crockford masters Scala and writes *Scala: The Good Parts*, development teams are left to figure out the value proposition of each language feature for themselves. There's no right and no wrong, just trial and error. Don't be afraid to question what doesn't make sense.

When the Reflection API was first introduced in Java, every Java developer wanted to flex their intellectual prowess and use reflection for **everything**. This lead to a portion of a codebase I was working on at the time to be a complete maintenance nightmare for no reason other than a single rogue developer going postal and overusing a feature they didn't understand. It's better to start off slowly and write clean, simple Scala before dipping your toes into every obscure language feature. I'd rather work with clean imperative style Scala than a mishmash of poorly implemented advanced techniques any day.

*Avoid the deep end of Scala until you're ready. Take your time.*

Just like good music, good code is elegant and sparse. Good food is made with few high quality ingredients. Nobody wants to eat a plate full of mush with every spice imaginable, so why write code like that? A confident level A1 Scala developer will write easier to maintain code than someone arbitrarily writing level A3/L3 code before they understand what they're doing and why they're doing it.

## 10. Call out ugly code

It's possible to write extremely ugly Scala, just like it's also possible to write extremely ugly Java, extremely ugly Perl, and extremely ugly English.

There is one major difference between ugly Java and ugly Scala, however.

Ugly Scala may be ugly in many different ways. It may be ugly imperative Scala, or it may be ugly functional Scala, or it may be an ugly mishmash of both. It may be ugly in ways that you don't understand yet because it's written in a style that's still unfamiliar. Scala is new, so it's not yet possible to instantly spot anti-patterns like we can do in more mature languages like Java. This can lead to a team of developers mistaking ugly code as beautiful code and beautiful code as ugly code. Developers may begin to emulate ugly patterns as they learn, which leads to even uglier code in the future. The vicious feedback loop begins.

It's better to avoid this altogether rather than try to fix it later.

If you see a brilliant developer write ugly Scala, call it out. Ask questions. Don't assume that because you're new to a language you don't have anything to offer. The worst case scenario is that you're wrong and the code you're questioning is elegant, but now you'll understand *why* it's elegant.

## 11. Scala is only one part of the puzzle

The way web applications are developed is much different today than it was a decade ago. I'm currently working on a Scala, Play, AngularJS, and MongoDB app. The bulk of my work is client-side. In the past year I've written more Angular than Scala, which isn't a bad thing, it's just reality.

The beauty of Scala is that I can write clean, stable, well performing server-side code, while avoiding the ugly boilerplate of Java and the brittleness of a dynamic interpreted language like Ruby. I can spend my time writing client-side code with the confidence that the server-side logic I implemented with Scala is bulletproof and ready to rock when I need it.

As much as I would like to become the grand-master of all things Scala, I'm a working programmer who needs to stay sharp on a wide variety of technologies; HTML5, SASS, AngularJS, RequireJS, SQL, MongoDB, etc, etc, etc, etc.

Even if you don't have time to master every aspect of a single language, Scala is a great technology to dive into. Reactive programming will be the de-facto style of development over the next decade and I believe Scala will be at the forefront of this paradigm shift. The performance gains and scalability benefits of reactive applications are impossible to ignore.

Coast-to-coast JSON is replacing bulky XML. REST is murdering overly-complicated SOA patterns. Mobile is replacing the desktop. Transforming massive amounts of data into actual knowledge is made possible with the performance of Akka without investing in ridiculous amounts of hardware.

Scala is only one part of the puzzle, but it's at the heart of many other puzzle pieces that are coming together to create a new breed of applications.

## 12. Learning Scala will make you a better programmer

It's rare that a corporate developer has to ***really*** stretch themselves. When's the last time you learned a totally different approach to software development?

The last major shift — and the only major shift I've witnessed in my career — was the migration from procedural languages to OO languages. I was lucky enough in 1998 to work on one of the very first Java applications in a Canadian bank while I interned at CIBC. Most of the developers I worked with were ex-COBOL and C programmers who took a leap of faith to try something different. For all the heat Java gets today, at the time Java was very practical when we had to deploy thick clients on both Windows and OS/2.

Working alongside programmers with 20 or 30 years of experience — some of whom **actually *programmed with punch cards*** — was a valuable learning experience and made me realize that I wouldn't stick with one style of programming forever. As I learned Java I also had to maintain JCL. I had to dig into old COBOL and 360 Assembly code for months before coming back to Java. It kept things in perspective. JCL and COBOL were not sexy, but they did the job they were designed to do. My time as an intern was spent exploring everything from Smalltalk to Excel — Excel being my first experience with functional programming!

Scala catches a lot of unwarranted flack. Rather than stretch themselves, some developers with short attention spans lash out and run away to more familiar languages. That’s fine. The problem is that they leave a trail of online discouragement in their wake, which is unfair to Scala as a language and makes interested developers question the value of the language.

If you read something discouraging about Scala, try to learn about who wrote it. They may have different requirements than you do. Don’t be easily swayed, anything challenging is bound to discourage some people. Explore the success stories of Scala, which are becoming more and more common.

## Conclusion

Scala is not only a technical investment, but also an investment in culture. The rewards are well worth the investment, and shouldn’t you be doing this *anyways*?

If you’re working on a new project that needs to be scalable, reliable, and easy to maintain, or you simply want to stretch yourself as a programmer, Scala is definitely worth it. With some common sense it’s not as difficult a transition as it seems.

Remember that Scala is ready for the enterprise — and already in the enterprise — ***now***!

WRITTEN BY

**Kevin Webber**

I’m a freelance Scala developer from Toronto, Canada. I do it all for the cookies. http://kevinwebber.ca