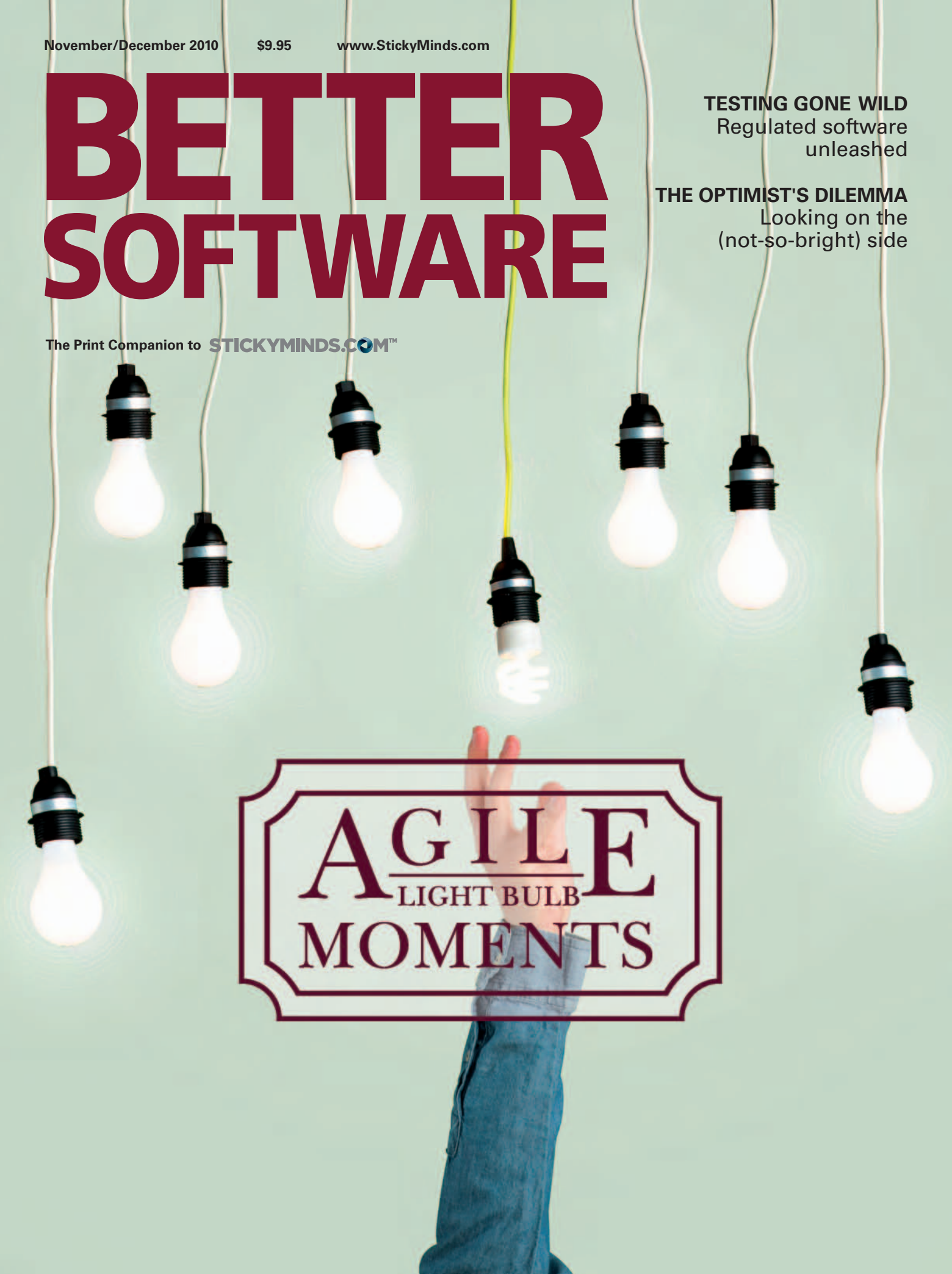
November/December 2010 $9.95 www.StickyMinds.com

# BETTER SOFTWARE

The Print Companion to STICKYMINDS.COM™

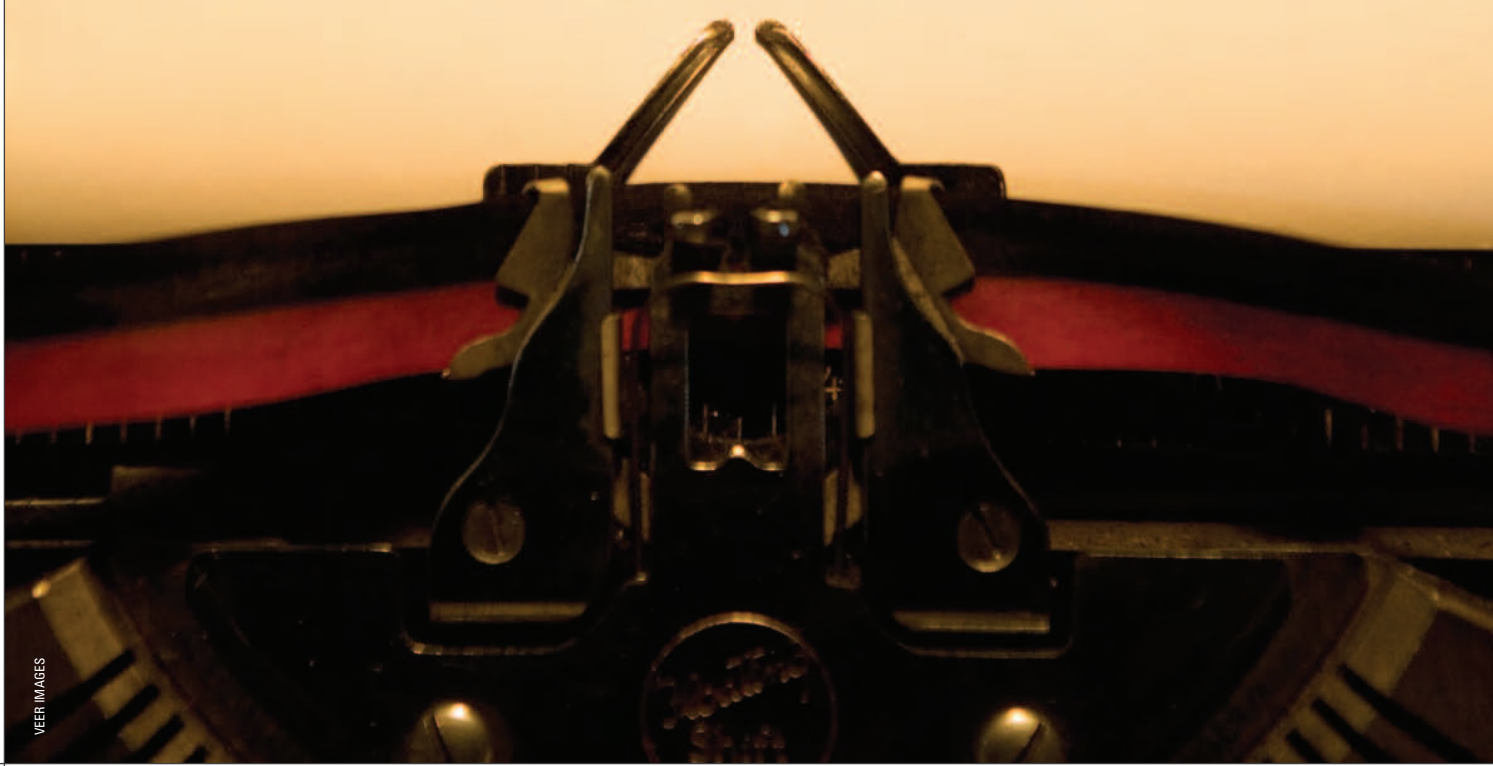
**TESTING GONE WILD**
Regulated software unleashed

**THE OPTIMIST'S DILEMMA**
Looking on the (not-so-bright) side



# An Introduction to Scala

by Daniel Wellman



VEER IMAGES

Scala is a programming language designed to be concise, safe, and compatible. Programs written in Scala run on the Java Virtual Machine and can reuse existing Java libraries and code. Scala is already used in production for business-critical systems; it powers several services at Twitter [1] including message queues, the streaming API, and people search function. Foursquare's main and mobile websites are written in Scala [2]. In addition, Scala is used in projects at companies such as LinkedIn, Novell, Sony Pictures Imageworks, and Siemens. Martin Odersky, Scala's creator, has a strong language design background: He wrote Sun's Java 1.3 compiler and cocreated Generics for Java.

Scala refines the object-oriented features of Java and borrows functional programming techniques from languages including ML and Haskell. This yields code sizes that are typically reduced by a factor of two to three when compared to equivalent Java code [3] and draws aesthetic comparisons to other dynamic languages [4] like Ruby. But Scala code is fast—with performance on par with Java [5] and faster than Ruby.

One of Scala's key strengths is its excellent compatibility with Java. Scala's compiler compiles Scala source files to Java class files (bytecode), and its runtime libraries are straight Java JARs. Scala code can call any existing Java code you have, so you can reuse all that Java code you've written. This means you can write code in Scala and run it anywhere you would run Java code—from desktop Swing applications to web applications powered by servers such as Jetty or Glassfish.

## Functional Programming

The functional programming style is an alternative to the imperative style often used in object-oriented languages. Functional programming languages model their computations in terms of side effect-free functions, like mathematical functions—the output of a function depends only upon its inputs. If you call a function with some input value, it will yield the same output no matter how many times you call the function. Compare this to an object-oriented language, where calling a method on an object may give you a different result on every invocation if an object's state may change (for example, a GUI window object's `getPosition()` method will return different values as the window is moved around the screen).

Functional languages are written in terms of immutable values; once a variable is assigned, it can never change. Think of final variables in Java—those are immutable values. Here's a sample assignment in Scala:

```
val theAnswer = 42
```

To declare an immutable value in Scala, you use the `val` keyword. Any attempt to reassign the value will result in a compiler error.

Furthermore, functions are first-class values just like numbers and strings; they can be assigned to variables and passed as arguments to functions. Here is how to find the even numbers between one and eight:

```
val numbers = List(1, 2, 3, 4, 5, 6, 7, 8)
numbers.filter { x => ( x % 2 ) == 0 }
// Result is List(2, 4, 6, 8)
```

This code creates a list containing the values one through eight and assigns it to the variable "numbers." The "filter" function on List is invoked, which evaluates the list and returns a new list containing only the items for which the specified function evaluates to true. In this case, we specify a function (between the curly brackets) that takes a variable "x" and returns a Boolean indicating if x modulo 2 is zero (another way of asking "Is x even?")

Functional programs can be simpler to understand than programs with mutable state, since a function's result depends only upon its inputs. If you've tried to understand how a method on an object works when the result depends on what happened to the object before the call, you've found a problem that depends upon mutable state.

This simplicity is helpful when applied to multithreaded concurrent programming. In Java, multithreaded programs are written using memory locks and synchronization, a technique that is extremely easy to get wrong and that can lead to race conditions requiring hair-pulling debugging sessions. Immutable state makes concurrent processing much simpler since you avoid many concurrency problems altogether (see the book *Java Concurrency in Practice* [6]).

Of course, programs often need mutable state at some point, and Scala supports this programming model as well. Scala enables you to design programs that have mutable, encapsulated state in objects and rely on the functional parts of the language for your computations and parallel processing.

## Scala—a Better Java

Scala is also an object-oriented language and improves upon many features of Java. In Scala, everything is an object—even numbers, characters, and Booleans—which makes for simpler code with fewer special cases. (Note the compiler replaces these objects with primitives, which gives you the best of both worlds: objects for simpler code, yet primitives for faster performance.) Scala has a powerful type system that enables type-safe code in many cases where Java would have required casting. And just like Java, Scala will catch type mismatch errors at compile time.

Scala code is less verbose than Java. Consider the code in Java to create a map of some HTTP result codes to their status messages:

```
// Java
Map<Int, String> statusMsgs =
    new HashMap<Int, String>();
myMap.put(200, "OK");
myMap.put(404, "Not Found");
```

In Scala, it's simply:

```
// Scala
val statusMsgs = Map(200 -> "OK",
                     404 -> "Not Found")
```

While Java requires us to specify the type of the map's keys and values on both the left and right sides of the expression, Scala looks at the creation on the right side and determines the variable's type. This is known as type inference, and it means that many type declarations can be omitted, which reduces the amount of code and clutter per line. Additionally, note how Scala lets you initialize the map's contents in one line, which is a nice touch. In Scala, Map is a library class like Java's HashMap.

Another feature that enhances Scala's object-oriented power is support for traits. Traits are similar to Java interfaces, which define a set of method signatures that a class will implement. However, traits may also contain method implementations, which is not possible with Java interfaces. And like Java interfaces, classes may implement many traits. Multiple traits may be stacked together in one class to assemble rich behavior with minimal code, but Java is limited to extending at most one base class to reuse any default behavior and requires that the programmer implement the remainder of the interfaces.

For example, the Scala library provides the Ordered trait, which is used to compare objects using semantics like greater than or less than. If a class implements the Ordered trait, the implementer must define how objects are compared, just like Java's Comparable interface. However, the Ordered trait provides default implementations of the methods >, >=, <, and <=, which the implementing class now gains "for free." (Note that Scala methods may use symbols in their names, unlike Java.)

## Why Learn Scala Now?

Scala is gaining momentum. The availability of books is a good indicator of market interest in the language. Presentations on Scala are becoming frequent at conferences. In April, Scala passed 1,000 questions available on the popular question site Stack Overflow.

And improvements are in the works. As of this writing, the latest stable version of Scala is 2.8.0, released in July 2010. Version 2.8 offers several improvements including: an improved collections library; support for nested Java annotations, which is especially useful if you want to use annotations for Java Persistence Applications API; new language features, including named and default function arguments; and hooks added to the compiler to make it easier to build Scala IDE plug-ins, which means better tool support.

## The Good and the Bad

Scala is an advanced language, but it's not without weaknesses. There have been some annoying problems with point releases of Scala that require you to recompile your code and use libraries recompiled with the latest version. However, the development team has stated that it will be focusing on binary compatible releases starting with version 2.8, which may help. Scala is a flexible, powerful language, which means that not only does it enable you to create beautiful, sophisticated code but it also won't prevent you from creating a snarled, unreadable mess.

Scala is developed at a university—not funded commercially by a big company like Java and Sun/Oracle—so its resources are more limited. There have been occasional compiler bugs, and the API documentation can be a bit thin compared to its Java counterpart.

That said, the Scala community is friendly and helpful, whether it's the active Scala mailing lists [7] or an online IRC chat channel. In fact, Scala's creator, Martin Odersky, actively participates in the mailing list, which reflects the close-knit nature of the community.

## How Can I Get Started?

Download Scala from the Scala website at (www.scala-lang.org.) Once you have installed Scala, you can get an interactive command prompt (also known as a read-eval-print loop or REPL) by typing "scala" at your command prompt. Several books about Scala are now available, plus many blogs on the subject (see the StickyNotes for links).

An IDE can help when learning a new language, and Scala has plug-ins for all the major IDEs (Eclipse, NetBeans, and IDEA). These plug-ins are still in their early development phases so, while you can expect syntax highlighting and some code completion, there aren't many refactoring features, and you can expect to find some problems such as the editor not marking all errors in the code (though the compiler will catch them).

One useful way to start practicing Scala is to write tests in Scala for your existing Java code. You can write tests just as you would in Java using libraries like JUnit or TestNG but in Scala's succinct syntax. Or to try out some of the more expressive features of Scala, you could use a Scala testing library like ScalaTest [8] or Specs [9].

Let me suggest one technique to avoid: Don't try to learn Scala by first learning Lift, a popular Scala web application framework. Lift takes a unique approach to web applications, which can be quite daunting to understand if you're new to Scala or functional programming. As an alternative starter project, consider the "99 Problems in Scala" [10] web page that features several exercises that demonstrate the functional programming style. **{end}**

dan@danielwellman.com


For more on the following topics go to www.StickyMinds.com/bettersoftware.

- References
- Further reading