

# Adopting Scala: Lean Architecture and Clean Code

## Chiradip Mandal

Adopting Scala in a production environment is a larger decision than simply learning a new programming language. It's critical to understand the business value Scala brings, how it simplifies architecture, and how it transforms the workforce.

We are mostly talking about Scala as the ecosystem in this book rather than treating as a language in isolation. Scala ecosystem comprises frameworks, libraries and tools that help architects and developers to use Scala in enterprise settings with all accessories required.

This book simplifies the adoption of Scala by addressing multiple factors like ROI, architectural benefits, time to market and artful coding. The readers will be able to unlearn obsolete concepts and apply the new techniques, styles, and patterns in new way to better solve their daily design and coding problems while positively impacting the bottom line.

Many architects fail to embrace Scala's new architectural paradigm, including concurrent capabilities, patterns, coding style, functional aspects, and composing components etc. This book will make **architects** and **developers** more productive and efficient in applying Scala to solve practical daily design and coding problems.

Finally, to make Scala practical, one needs to be aware of its rich ecosystem - namely frameworks, libraries, and build tools. This book is not designed to teach a lot about them but will bring them in as appropriate.

## 1 Adopting Scala – delving into Scala ecosystem

### 1.1 Why adopt Scala?

- 1.1.1 Why Every Java shop should consider switching to Scala ASAP with zero loss and significant gain
- 1.1.2 Key Benefits – Scala as a language
- 1.1.3 Key Benefits – Scala as the ecosystem
- 1.1.4 Business case – how it really transforms business
- 1.1.5 Architecture – Scala simplifies complex architecture
- 1.1.6 Workforce – efficiency, productivity, quality and energized employees

### 1.2 How Scala fits into an existing Java/JEE ecosystem

- 1.2.1 Pushing Scala into Java ecosystem without friction
- 1.2.2 Making Scala work with Java based frameworks and libraries
- 1.2.3 Development Tool Support
- 1.3 Scala brings Functional capability into Java Ecosystem – a huge boost in productivity, simplicity, and new possibilities
- 1.4 Scala and other new languages – when Scala is more appropriate than others
- 1.5 Cost of adoption – you can make it almost free
- 1.6 Risk assessment – no other risk than changing the mindset
- 1.7 First step to adopt Scala in your enterprise
  - 1.7.1 Convince yourself, your team, and your bosses
  - 1.7.2 Introduce Scala into ongoing Java project right now – its easy, and 100% non-intrusive
- 1.8 Who is using Scala?
- 1.9 Summary

## **2 Scala ecosystem makes Architecture simple – leaner and cleaner**

- 2.1 Introduction to Scala (as the) ecosystem
- 2.2 Distributed Application made easy
  - 2.2.1 Actor System – distribute at will, be wise though
  - 2.2.2 Low-level Message based contracts as opposed to API based contracts – Don't think JMS
- 2.3 Break free from Application Servers – this means simplicity and speed
- 2.4 Lets talk about traditional JEE enterprise Architecture
- 2.5 Lets see how Scala can simplify the JEE architecture
  - 2.5.1 Lets see why it does not need application server in most places
  - 2.5.2 Scalability comes naturally with Scala ecosystem and not with Application Servers
- 2.6 Let is crash - Let the system crash and recover quickly

## **3 Mini chapter on Getting started with Scala based Architecture and Prototype**

- 3.1 Identify the type of the architecture
- 3.2 Choose the right framework to use
- 3.3 Divide the system into decoupled components – Scala really adds new paradigm in decoupling
- 3.4 Create the distribution strategy
- 3.5 Choose a good new build tool
- 3.6 Write the prototype

## **4 Integration**

- 4.1 The Java and non-Java world
- 4.2 Messaging Systems

- 4.2.1 AMQP
    - 4.2.2 JMS
    - 4.2.3 ZeroMQ
  - 4.3 Enterprise Integration Patterns
    - 4.3.1 Camel
    - 4.3.2 Spring Integration
  - 4.4 OSGi and Scala
  - 4.5 Summary
- 5 The concurrency puzzle –Actors and Akka**
  - 5.1 Scala and Akka makes concurrent architecture easy
  - 5.2 The Actor Model
  - 5.3 Introducing Akka
  - 5.4 Styles of using Scala Actor Model to solve key concurrency problems
  - 5.5 Building an Elastic Architecture using Actors
  - 5.6 A case study
  - 5.7 Summary
- 6 Building Scalable Systems with Scala & Akka**
  - 6.1 Why you need to scale
  - 6.2 Design styles of Scala based systems
  - 6.3 Departure from Application Server dominated world
  - 6.4 Enterprise Architecture revisited without App Server
    - 6.4.1 Raw performance with less computing resources
  - 6.5 The benefits of this new architectural style – ROI, TTM, PnP, Extensibility, and Manageability
  - 6.6 Fitting it together in the ecosystem
  - 6.7 Frameworks to consider
  - 6.8 Summary
- 7 Big Data in simple (read Scala) way**
  - 7.1 What's different about Big Data?
    - 7.1.1 Big Data is distributed
    - 7.1.2 Big Data demands massive scale
    - 7.1.3 Moving data is more expensive than moving execution
    - 7.1.4 Processing Huge Data needs advanced data-structure and collections
    - 7.1.5 Raw Performance is in demand in Big Data processing
  - 7.2 Scala meets Big Data
  - 7.3 Scala and Some Big Data systems
    - 7.3.1 Hadoop and Scala
    - 7.3.2 Spark
  - 7.4 Complimentary technologies to Scala for Big Data
  - 7.5 Business benefits of using Big Data in Scala way
  - 7.6 What to avoid and what to embrace
  - 7.7 Summary
- 8 Spark – the Big Data in Scala way**
  - 8.1 What is Spark?

- 8.2 Advantages of using Spark
- 8.3 Use cases and performance comparison
- 8.4 Resources to further explore Spark
- 8.5 Summary

## **9 Mixing with other languages**

- 9.1 Polyglot Architecture
- 9.2 Mixing Scala with Java
- 9.3 Mixing Scala with JavaScript/NodeJS
- 9.4 Mixing Scala with Clojure
- 9.5 XML processing
- 9.6 JSON processing
- 9.7 Summary

## **10 Testing with Scala**

- 10.1 How testing Scala is different
- 10.2 Scala Test Frameworks
- 10.3 Introducing Scala Test
- 10.4 Test Driven Development with FunSuite
- 10.5 Business Driven Development with FunSpec
- 10.6 Functional, Integration and Acceptance testing with FeatureSpec
- 10.7 Summary

## **11 Build**

- 11.1 Bad old Maven
- 11.2 Simple Build Tool (SBT) – really simplify the build process
- 11.3 Gradle
- 11.4 Leiningen
- 11.5 Summary

## **12 Design Styles and Patterns**

- 12.1 Composition vs. Inheritance
- 12.2 Template
- 12.3 Adapter
- 12.4 Decorator
- 12.5 Filter/Dispatch
- 12.6 LoadBalancing
- 12.7 IoC – Cake patterns
- 12.8 Audit/Logging
- 12.9 How these styles and patterns impacts Architecture and Efficiency
- 12.10 Summary

## **Appendices**

### **13 Frameworks**

- 13.1 Play framework and Play Mini
- 13.2 Lift
- 13.3 Spray
- 13.4 Scalatra

13.5 Finagle

13.6 Scalaz

## **14 Handling Data**

14.1 Scala Collections

14.2 Handling XML

14.3 Handling JSON

14.4 ORM tools

14.5 Slick

14.6 Mapper

## **15 Scala powertools**

15.1 Type enrichment

15.2 Function as value

15.3 Future

15.4 Special collections - Either, Some, None

15.5 Covariance and contra-variance

**About the author:** Chiradip is a Technologist at Chiradip.com, Technologist & Product Architect at Cisco, managing a large Scala project. Architected many successful commercial products in very quick succession. Technology evangelist – instrumental in promoting Scala, Clojure, Erlang and other modern languages, hands-on using various technologies including Scala, Akka, NodeJS, active in technology communities. Profile link: <http://www.chiradip.com/chiradip-a-brief-profile.html>