HOW WE PORTED OUR APPLICATION FROM JAVA TO SCALA

Øyvind Raddum Berg

EARLY TIMELINE PENGER.NO

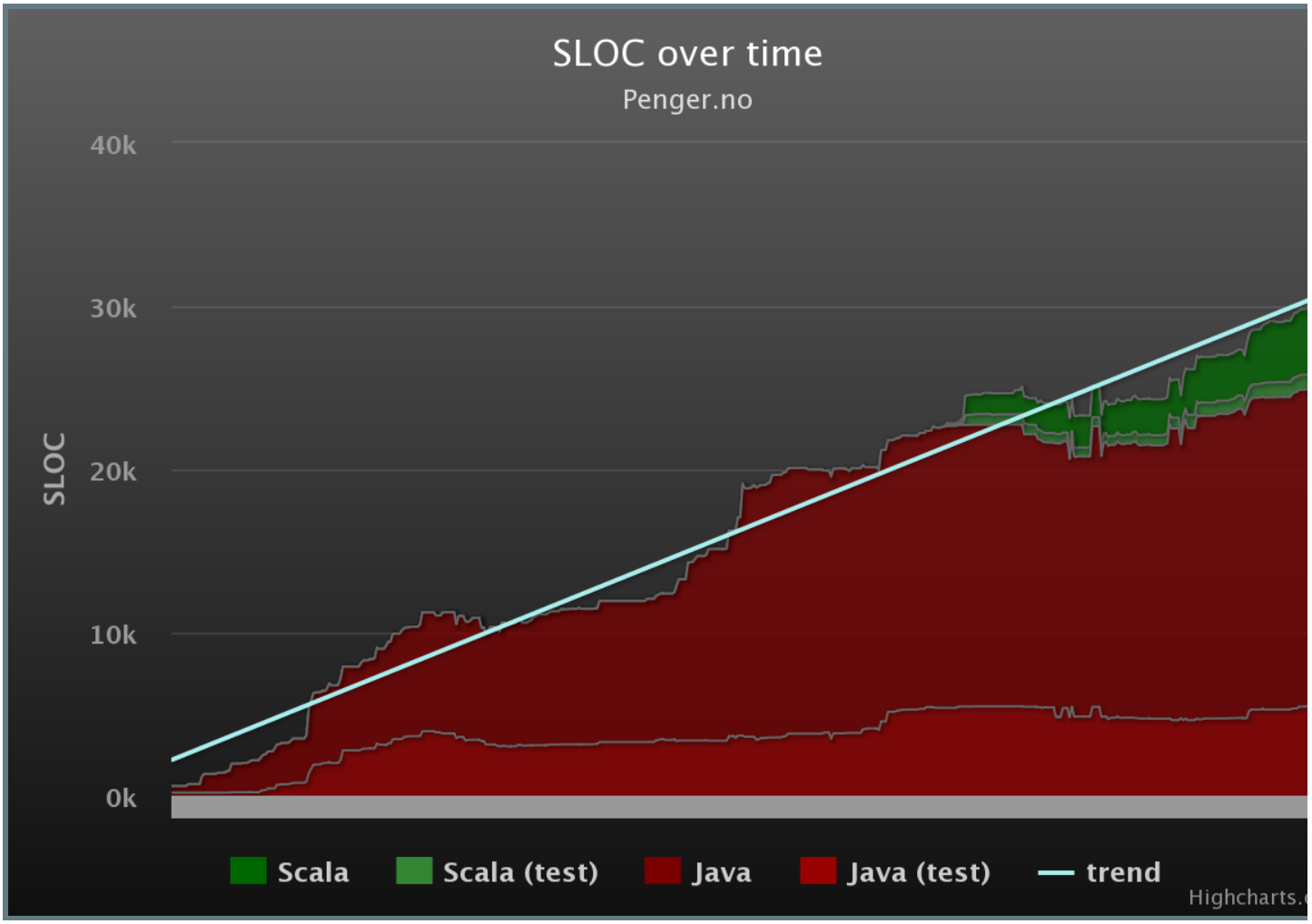
autumn 2010 start of mortgage project (*in Java*)
autumn 2011 start of insurance project (*in Scala*)

SITUATION EARLY 2012...

- code base growing quickly
- unhappy with technology

CODEBASE SIZE AND COMPLEXITY GROWING QUICKLY

initial 18 months of project (jan 2011-june 2012)



UNHAPPY WITH TECHNOLOGY

FRAMEWORK HELL:

- limited/leaky abstractions
- want to understand whats going on
- runtime failures
- debugging

ENVY

- prime motivating factor

HAD TESTED THE SCALA WATERS

- insurance app
- backoffice application already rewritten
- expertise within team

WANTED TO OPTIMIZE FOR:

- developer happiness
- peace of mind/trust in code
- similar technology between projects

PORTING STRATEGY:

refactor Java codebase
introduce optional semantics

Java	=> Scala
Java collections	=> Scala collections
checked exceptions	=> Either/Try
Hibernate	=> Slick
Spring	=> cake pattern
Spring MVC	=> Unfiltered

REFACTOR JAVA CODEBASE

- limit/contain mutability
- embrace java 8 stream api or guava
- avoid multiple returns
- short, static functions
- composition over inheritance
- value classes

JAVA.UTIL.OPTIONAL<T>

compile-time guarantees and easy porting, not elegance

MECHANICALLY CONVERT

SOURCE GRAPH

- Scala calling Java 👍🏼
- Java calling Scala 👎🏼
- start converting controllers, then continue downstack
- keep all inter-module APIs «Java-friendly» while you can

INTELLIJ <3

EXAMPLE

```
public class Foo<T> {
    private final T param;

    public Foo(T param) {
        this.param = param;
    }
}
```

automatically converted (doesnt compile, in this case):

```
class Foo {
  def this(param: T) {
    this()
    this.param = param
  }

  private final val param: T = null
}
```

what you probably want:

```
case class Foo[T](param: T)
```

PORT JAVA COLLECTIONS

CONVERTING BETWEEN HIERARCHIES

explicitly (preferred)

```
import scala.collection.JavaConverters._

val javaMap: util.Map[String, Int] =
    immutable.Map("foo" -> 2, "bar" -> 42).asJava
```

implicitly

```
import scala.collection.JavaConversions._

val javaMap: util.Map[String, Int] =
    immutable.Map("foo" -> 2, "bar" -> 42)
```

CHECKED EXCEPTIONS => EITHER[A, B] OR TRY[T]

rewriting exposes inconsistencies

DESPRINGIFY

REPLACEMENT FUNCTIONALITY

- spring comes with... everything
- most functionality we need exists in scala stdlib, or is easy to write yourself
- took a few dependencies, like commons-fileupload, and commons-er

APP WIRING

- started rewiring components early because of slick
- two apps in one, with somewhat intertwining dependencies
- *ugh*...

HAVE SPRING DO THE ULTIMATE WIRING

SHARE CONFIG AND DATABASE CONNECTION

```
@Configuration
class SpringBeans {
  @Bean @Autowired
  def concreteApp(ds: javax.sql.DataSource, cfg: Config): ConcreteApp =
    ...
}
```

HAVE SPRING DO THE ULTIMATE WIRING

SHARE CONFIG AND DATABASE CONNECTION

```
new ConcreteApp with DataSourceDbConnectionComponent {
  lazy val datasource = ds
  lazy val conf       = cfg
}
```

HAVE SPRING DO THE ULTIMATE WIRING

SHARE CONFIG AND DATABASE CONNECTION

```
@Configuration
class SpringBeans {
  @Bean @Autowired
  def concreteApp(ds: javax.sql.DataSource, cfg: Config): ConcreteApp = {

    new ConcreteApp with DataSourceDbConnectionComponent {
      lazy val datasource = ds
      lazy val conf       = cfg
    }

  }
}
```

PORTING TIMELINE

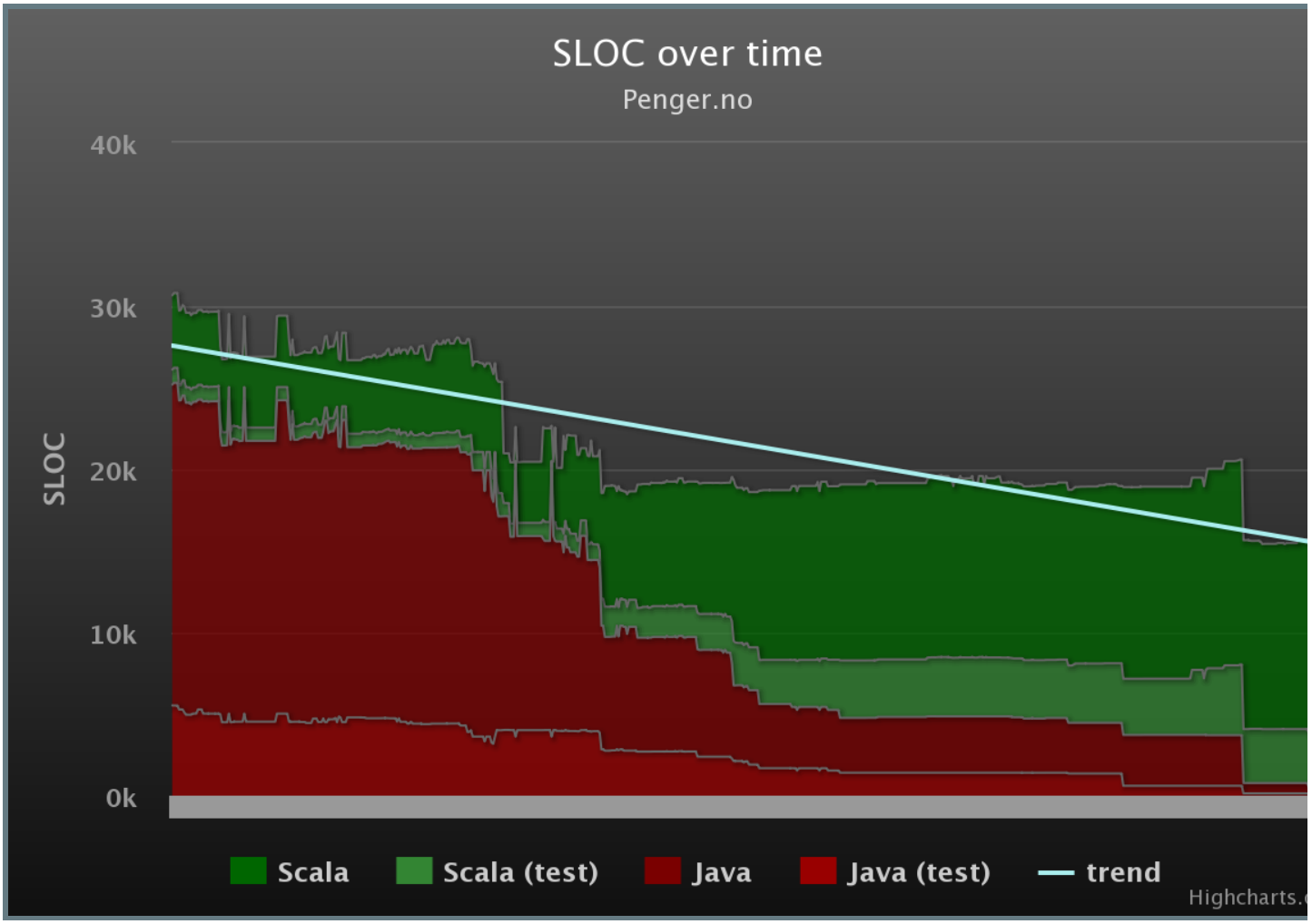
summer 2012	Started refactoring Java code
november 2012	~~maven~~ => sbt
december 2012	Started porting to Scala
april 2013	~~javax.validation~~
june 2013	~~Hibernate~~
june 2013	~~Spring in service layer~~
january 2014	~~Spring MVC~~

TIME USAGE

- six months of preparatory refactoring
- one year of on and off porting
- no dedicated time until december 2013
- two months stop-the-world development to code new frontend, rew
  http layer and wrap up porting effort

CODEBASE EVOLUTION

june 2012 - april 2014



CONCLUSION

- 👍developer happiness
- 👍peace of mind/trust in code
- 👍similar technology between projects