Scala in Practice

3 years later…

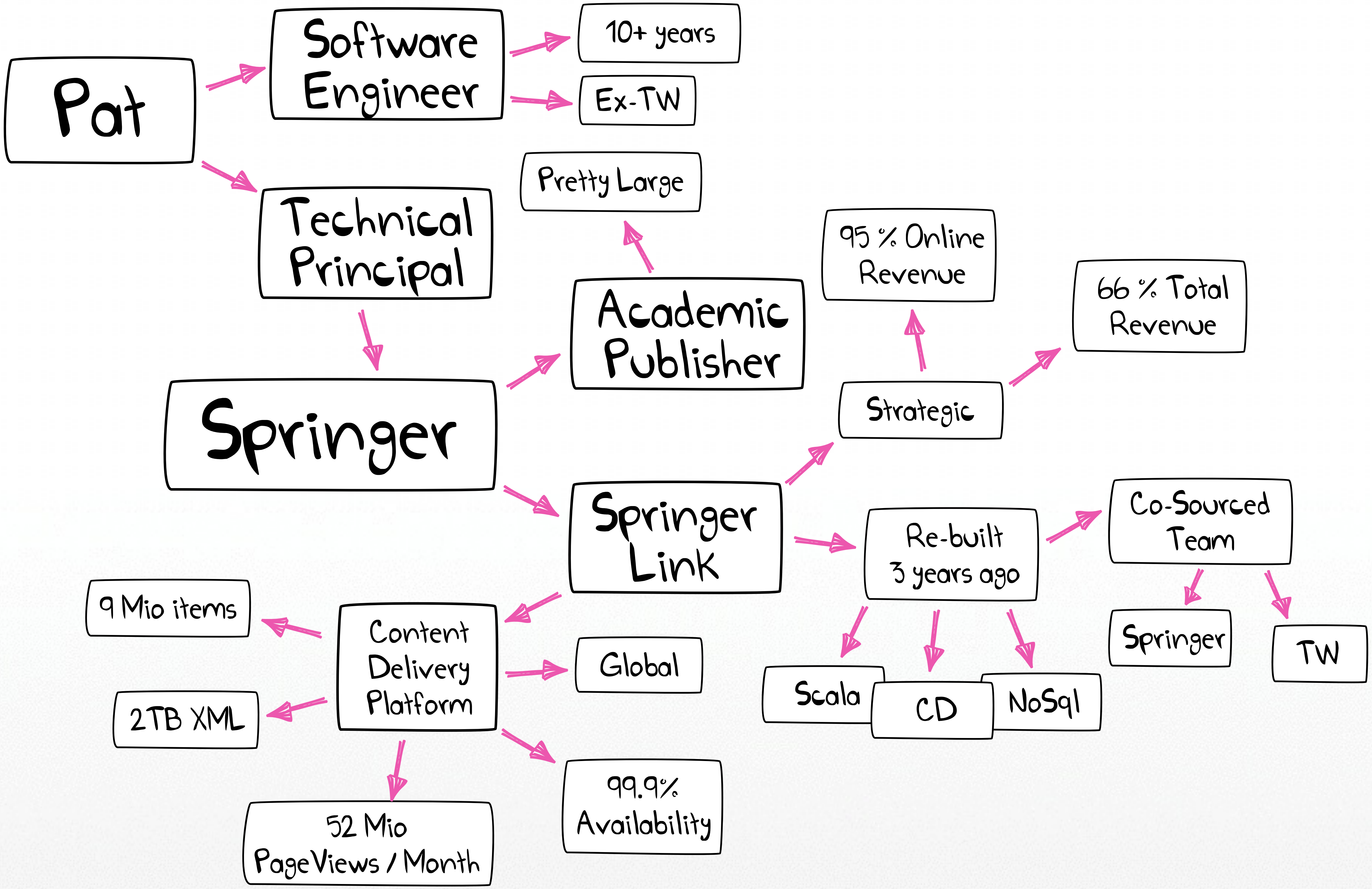
@patforna

patric.fornasier@springer.com

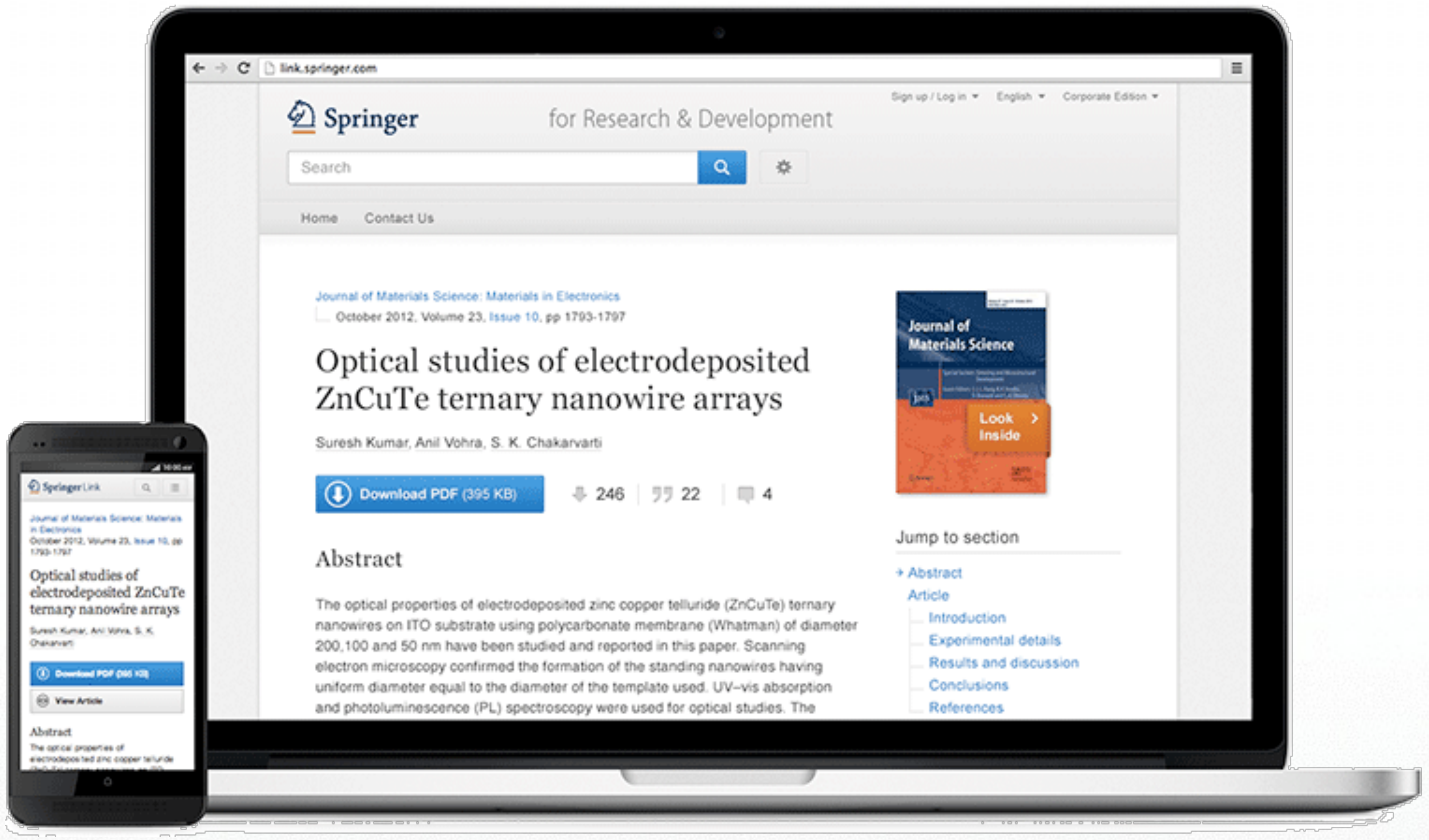
Context

Context

Springer



SpringerLink



link.springer.com

Springer for Research & Development

Sign up / Log in English Corporate Edition

Search

Home Contact Us

Journal of Materials Science: Materials in Electronics
October 2012, Volume 23, Issue 10, pp 1793-1797

Optical studies of electrodeposited ZnCuTe ternary nanowire arrays

Suresh Kumar, Anil Vohra, S. K. Chakarvarti

Download PDF (395 KB) 246 22 4

Abstract

The optical properties of electrodeposited zinc copper telluride (ZnCuTe) ternary nanowires on ITO substrate using polycarbonate membrane (Whatman) of diameter 200,100 and 50 nm have been studied and reported in this paper. Scanning electron microscopy confirmed the formation of the standing nanowires having uniform diameter equal to the diameter of the template used. UV–vis absorption and photoluminescence (PL) spectroscopy were used for optical studies. The

Journal of Materials Science

Look Inside

Jump to section

- Abstract
- Article
  - Introduction
  - Experimental details
  - Results and discussion
  - Conclusions
  - References

SpringerLink

Journal of Materials Science: Materials in Electronics
October 2012, Volume 23, Issue 10, pp 1793-1797

Optical studies of electrodeposited ZnCuTe ternary nanowire arrays

Suresh Kumar, Anil Vohra, S. K. Chakarvarti

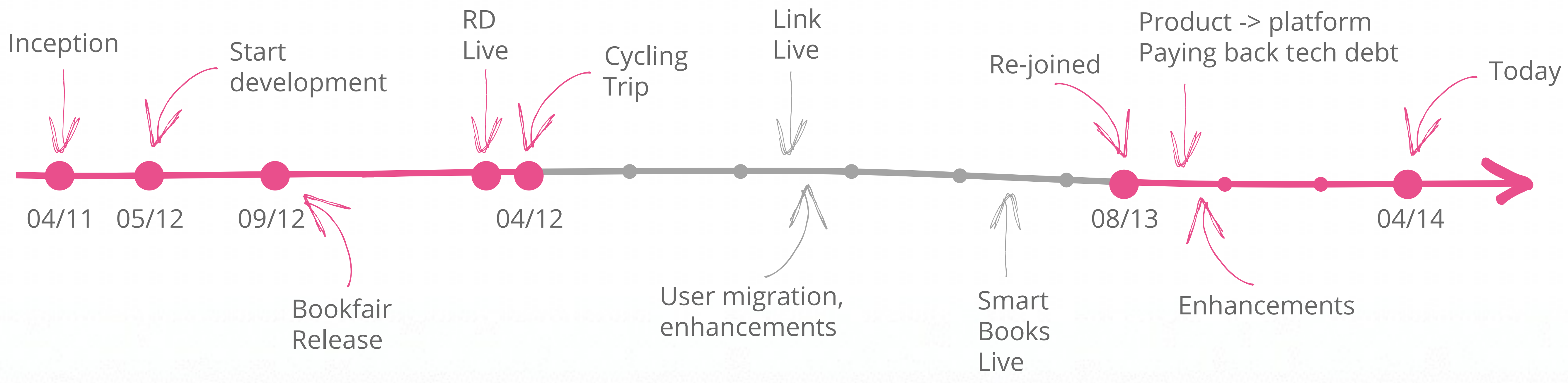
Download PDF (395 KB)

View Article

Abstract

The optical properties of electrodeposited zinc copper telluride

Timeline


Inception
Start development
RD Live
Cycling Trip
Link Live
Re-joined
Product -> platform Paying back tech debt
Today
04/11
05/12
09/12
04/12
08/13
04/14
Bookfair Release
User migration, enhancements
Smart Books Live
Enhancements

Looking back: why scala?

- Increase productivity
- Be more attractive employer
- Team decision

Looking back: good vs bad

Good

- Functional programming
- Terse syntax
- JVM ecosystem
- Gentle learning curve
- DSL friendly syntax
- Motivated team

Bad

- Tool support
- Compilation times
- Language complexity #moreRope

Fast-forward

Fast-forward (Aug 2013)

- 2.5 years into project
- 1.5 years of weekly live releases
- 100k LOC
- >10k commits
- >90 committers

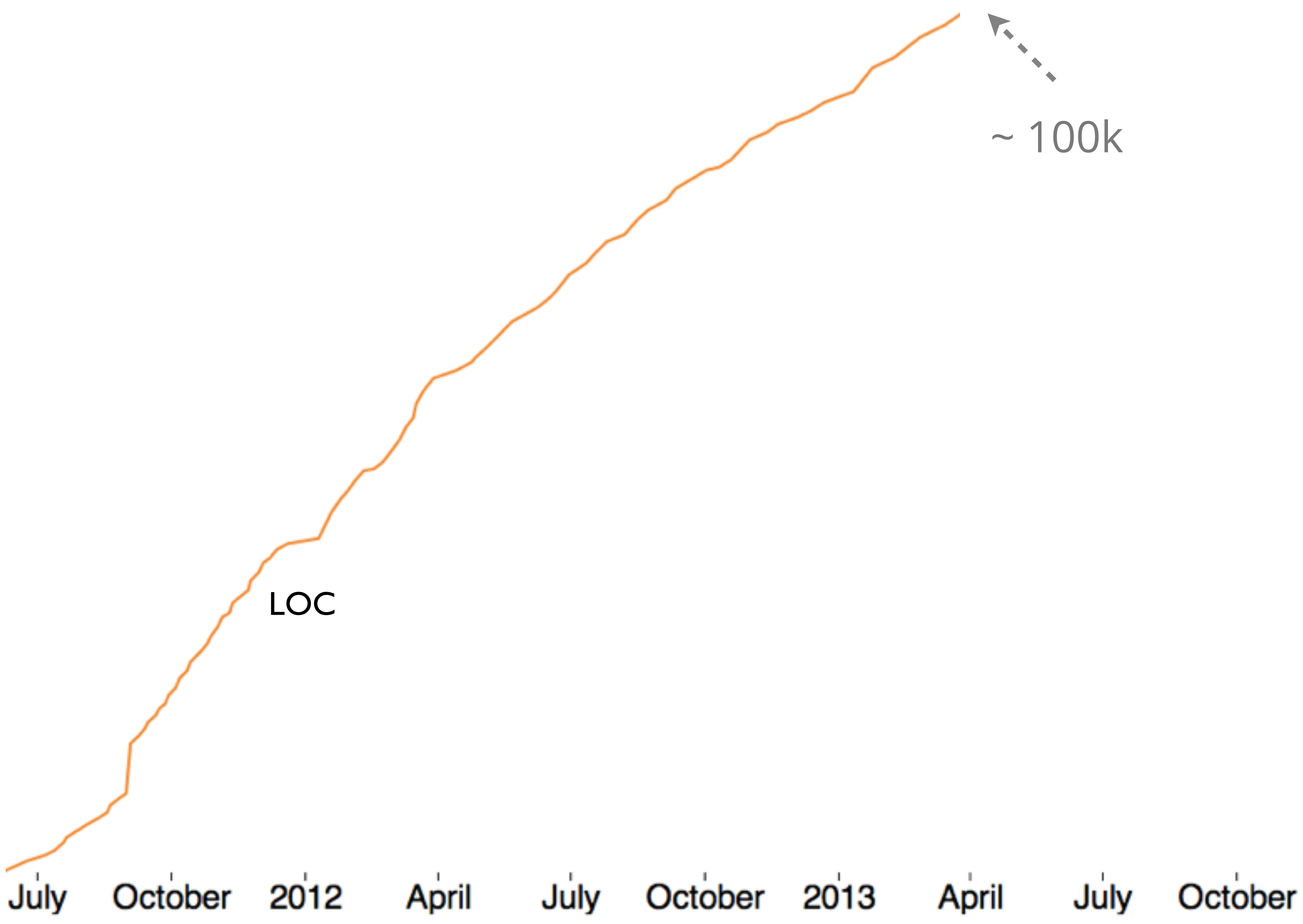
Fast-forward (Aug 2013)

- 2.5 years into project
- 1.5 years of weekly live releases
- 100k LOC
- >10k commits
- >90 committers

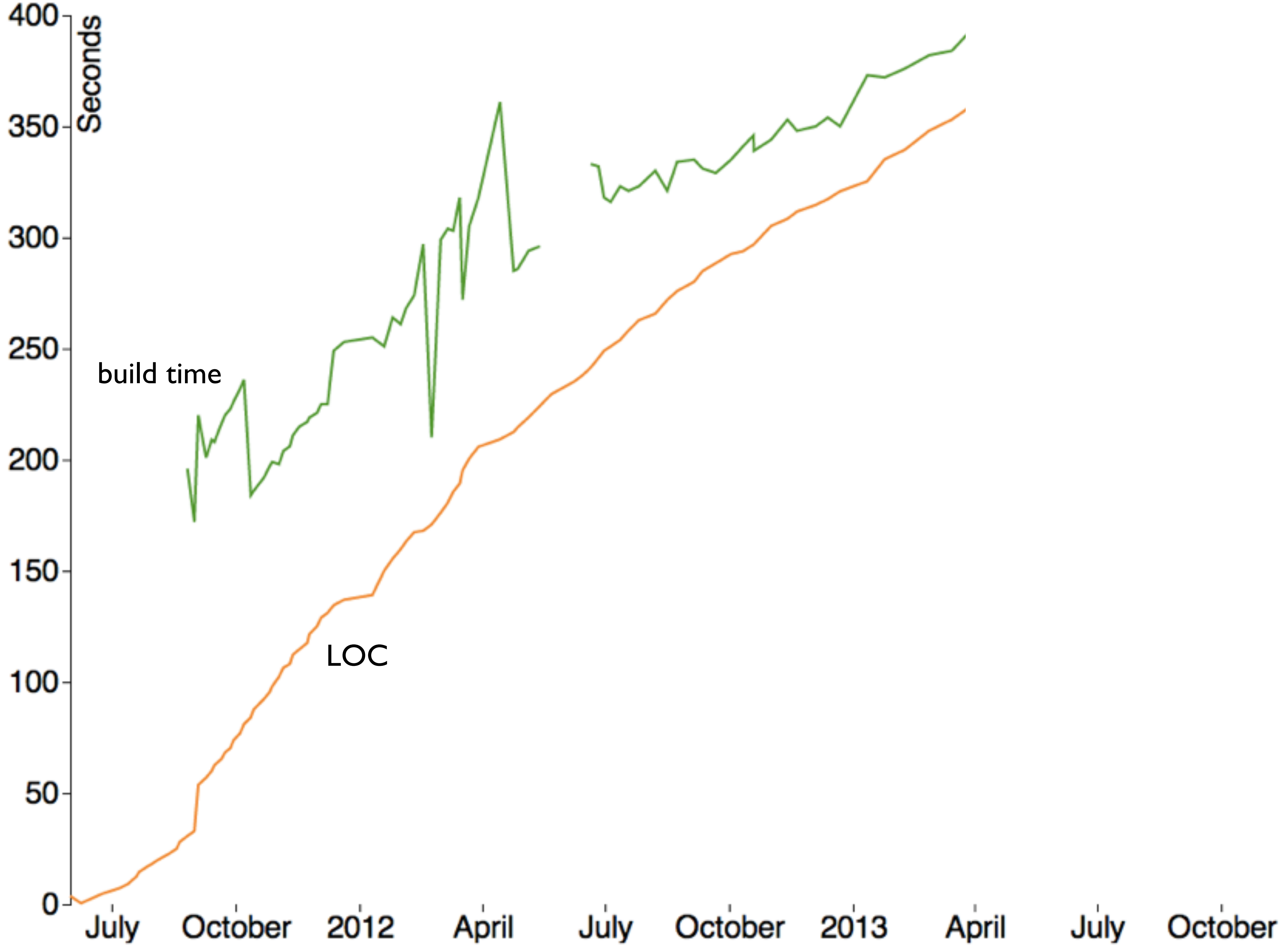
- Poor feedback loops *
- Lots of accidental complexity*

* not all related to Scala - to be fair

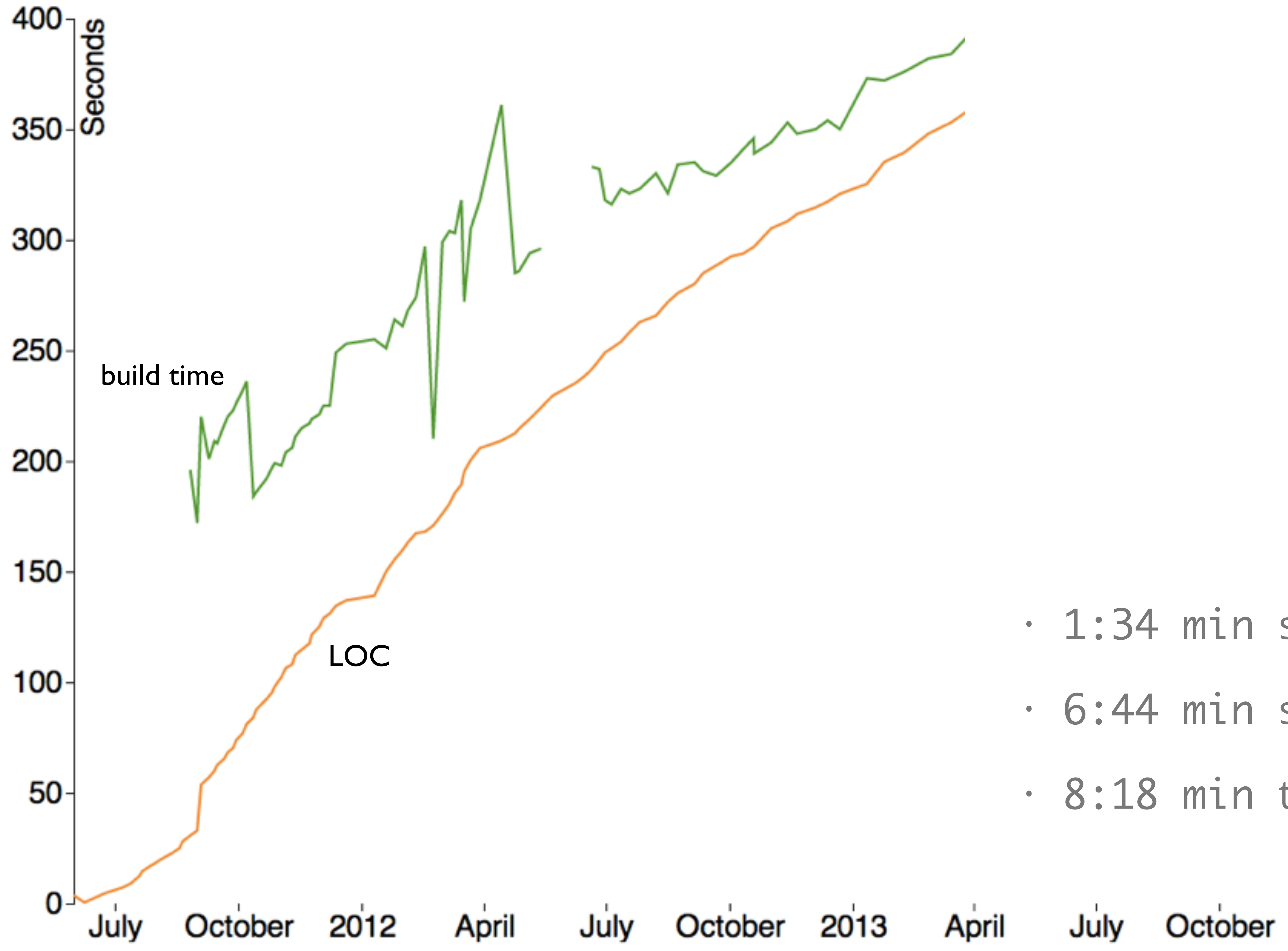
Trend (2 years)



Trend (2 years)



Trend (2 years)



- `1:34 min` src/main
- `6:44 min` src/test
- `8:18 min` total

What did we do?

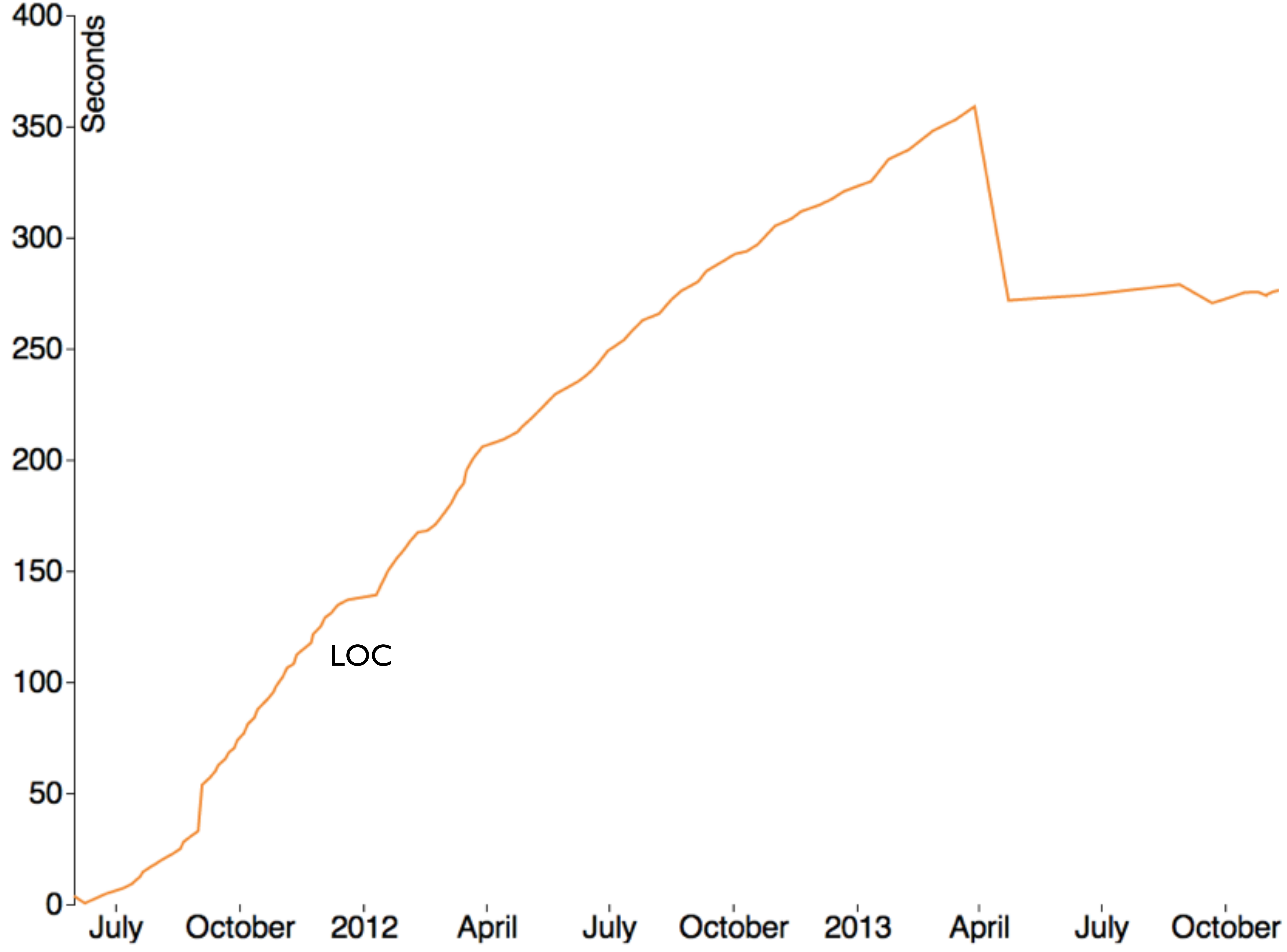
What did we do

- Reduced build time
- Improved feedback loops
- Reduced accidental complexity

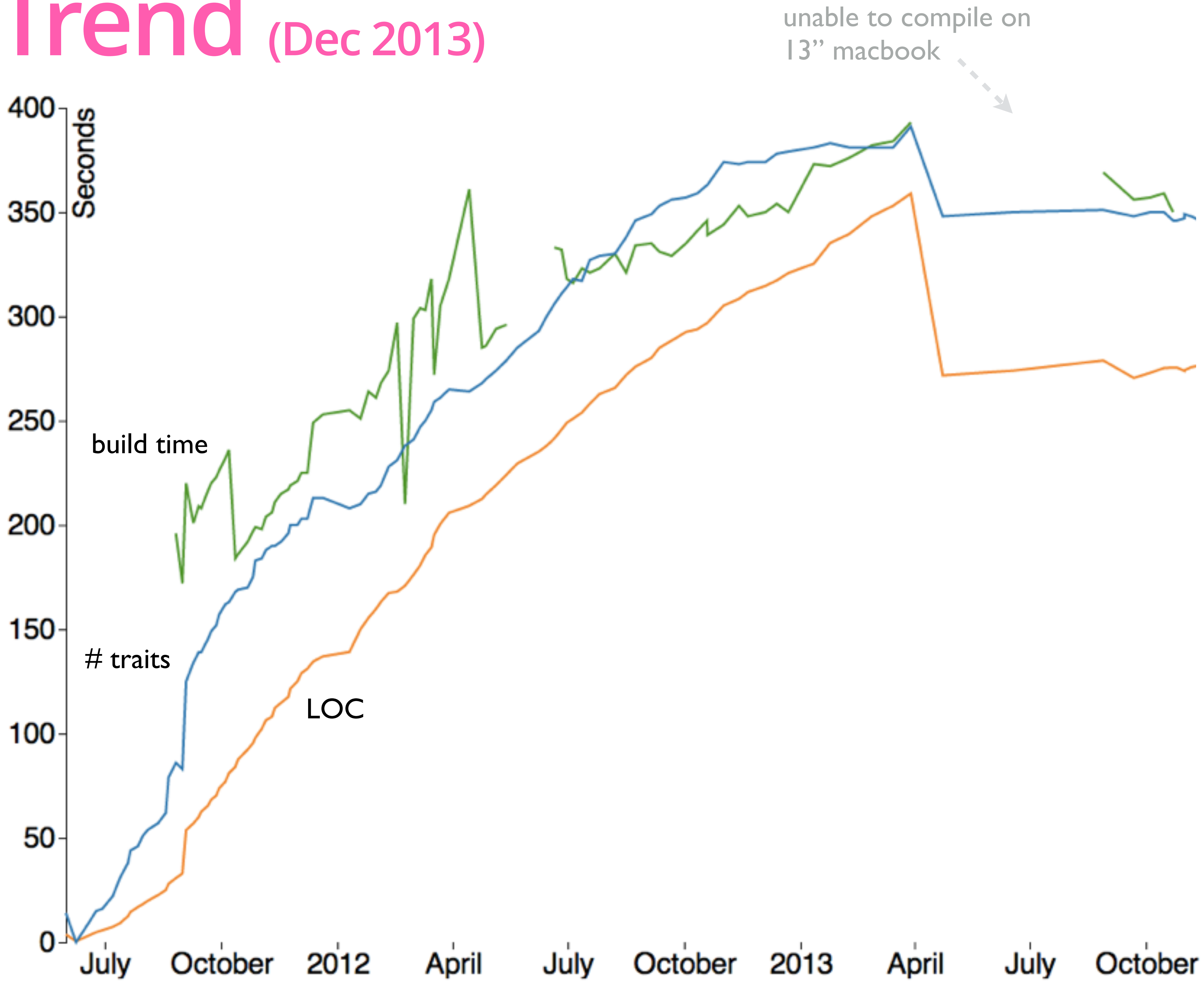
Build time

- Reduced size of codebase (broke off vertical slices, pulled out APIs, pulled out libraries, removed unused features, removed low-value tests, etc.)
- Reduced usage of certain language features (esp. traits and implicits)

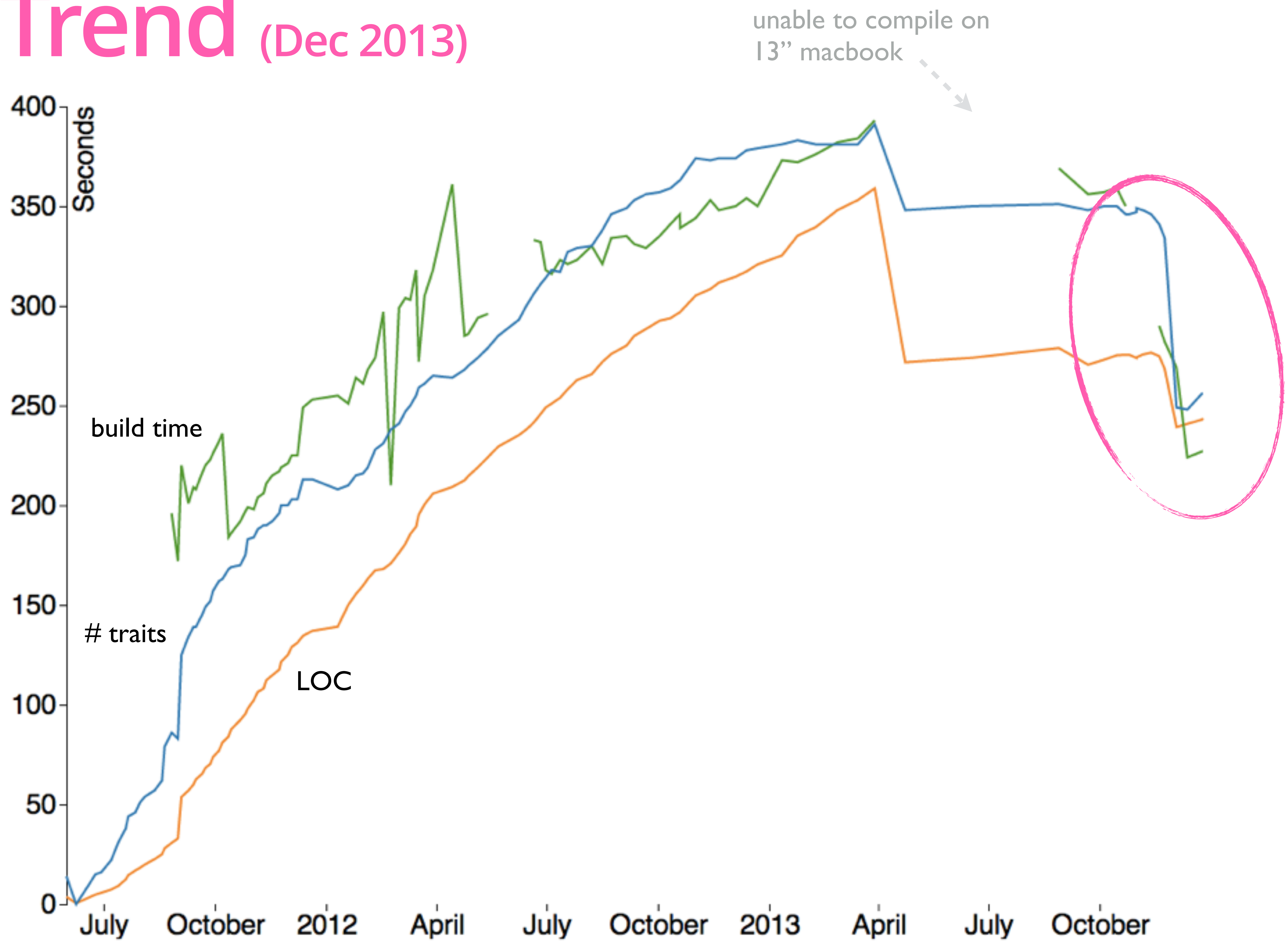
Trend (Dec 2013)



Trend (Dec 2013)



Trend (Dec 2013)



The problem with traits

- Will re-compile on every class the trait is mixed in
- Slows down dev-build cycle
- Will result in byte code bloat
- Will compile *a lot* slower

For faster compile times:

- Use pure traits
- Use old-school composition for code re-use
- Use pure functions via imports (e.g. import Foo._)
- If unavoidable, use inheritance for code re-use

Build time

Build time

- `1:34 min` src/main
- `6:44 min` src/test
- `8:18 min` total

Build time

- `1:34 min` src/main
- `6:44 min` src/test
- `8:18 min` total

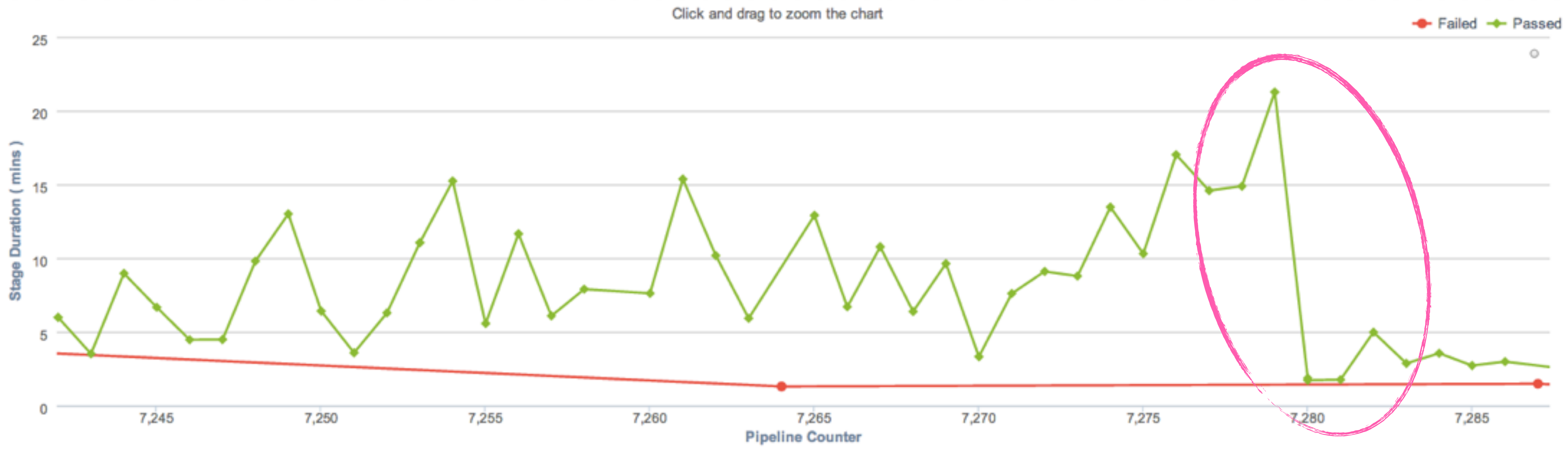
- `0:24 min` src/main
- `3:11 min` src/test
- `3:35 min` total

Build time (on CI server)

- Incremental compilation on CI
- Only one dedicated CI agent
- Physical build servers
- CPUs with higher clock speed

Build time (on CI server)

- Incremental compilation on CI
- Only one dedicated CI agent
- Physical build servers
- CPUs with higher clock speed



Complexity

Complexity

- There’s still a lot of code in our codebase that is hard to read
- It seems to be very easy to shoot yourself in the foot with Scala
- Scala *is* complex (and that’s why scalac will never be as fast as javac)

Complexity

- There's still a lot of code in our codebase that is hard to read
- It seems to be very easy to shoot yourself in the foot with Scala
- Scala *is* complex (and that's why scalac will never be as fast as javac)

```
Invariant/covariant/contravariant types (T, +T and -T)
Refined types (new Foo {...})
Structural types (x: {def y: Int})
Path dependant types (a.B)
Specialized types (@specialized)
Self types (this =>)
Projection types (A#B)
Existential types (M[_])
```

```
                       Type bounds (<:, >:)
        Type constraints (=:=, <:< and <%<)
                     Type members (type T)
                 Type aliases (type T = Int)
               Type classes ( (implicit ...) )
                          View bounds (<%)
                  Higher kinded types (* => *)
    F-Bounded type polymorphism (M[T <: M[T]])
```

http://nurkiewicz.github.io/talks/2014/scalar/#/16

Not opinionated

- Many ways to do the same thing
- Coding conventions help, but only so much*

*For example: http://twitter.github.io/effectivescala/

Not opinionated

- Many ways to do the same thing
- Coding conventions help, but only so much*

```
def foo() = "foo"
def bar = "bar"

foo
foo()
bar
bar()            // won't compile
```

*For example: http://twitter.github.io/effectivescala/

Not opinionated

- Many ways to do the same thing
- Coding conventions help, but only so much*

```
def foo() = "foo"
def bar = "bar"

foo
foo()
bar
bar()               // won't compile


def baz(x: String) = x
“x”.charAt(0)
“x” charAt(0)     // won't compile
“x”.charAt 0      // won't compile
“x” charAt 0
baz("x")
baz “x"             // won't compile
```

*For example: http://twitter.github.io/effectivescala/

Not opinionated

- Many ways to do the same thing
- Coding conventions help, but only so much*

```
def foo() = "foo"
def bar = "bar"

foo
foo()
bar
bar()              // won't compile


def baz(x: String) = x
“x”.charAt(0)
“x” charAt(0)    // won't compile
“x”.charAt 0     // won't compile
“x” charAt 0
baz("x")
baz “x"            // won't compile
```

```
list.foreach { x => println(x) }
list.foreach ( x => println(x) )
list.foreach { println(_) }
list.foreach ( println(_) )
list foreach { x => println(x) }
list foreach ( x => println(x) )
list foreach { println(_) }
list foreach ( println(_) )
```

* For example: http://twitter.github.io/effectivescala/

Not opinionated

- Many ways to do the same thing
- Coding conventions help, but only so much*

```
def foo() = "foo"
def bar = "bar"

foo
foo()
bar
bar()              // won't compile


def baz(x: String) = x
“x”.charAt(0)
“x” charAt(0)      // won't compile
“x”.charAt 0       // won't compile
“x” charAt 0
baz("x")
baz “x"            // won't compile
```

```
list.foreach { x => println(x) }
list.foreach ( x => println(x) )
list.foreach { println(_) }
list.foreach ( println(_) )
list foreach { x => println(x) }
list foreach ( x => println(x) )
list foreach { println(_) }
list foreach ( println(_) )

if (foo) "x" else "y"

foo match {
 case true => "x"
 case _ => "y"
}
```

* For example: http://twitter.github.io/effectivescala/

Surprises

http://dan.bodar.com/2013/12/04/wat-scala/

Surprises

```
List(1, 2, 3).toSet
```

http://dan.bodar.com/2013/12/04/wat-scala/

Surprises

```
List(1, 2, 3).toSet
scala.collection.immutable.Set[Int] = Set(1, 2, 3)
```

http://dan.bodar.com/2013/12/04/wat-scala/

Surprises

```
List(1, 2, 3).toSet
scala.collection.immutable.Set[Int] = Set(1, 2, 3)

List(1, 2, 3).toSet()
```

http://dan.bodar.com/2013/12/04/wat-scala/

Surprises

```
List(1, 2, 3).toSet
scala.collection.immutable.Set[Int] = Set(1, 2, 3)

List(1, 2, 3).toSet()
Boolean = false
```

http://dan.bodar.com/2013/12/04/wat-scala/

Implicits

- Can make it very hard to read code
- Tool support is very bad
- Impacts compilation time
- Surprising behaviour (esp. when used with overloaded methods or optional params)

Tooling

- Tool support is still very basic
- Makes it hard to continuously refactor (which means people are less likely to do it)

```
def handle(response: HttpResponse, request: HttpRequest)
```

Tooling

- Tool support is still very basic
- Makes it hard to continuously refactor (which means people are less likely to do it)

```
def handle(response: HttpResponse, request: HttpRequest)
```

no luck with “change signature”
refactoring support

Trait entanglements

- Makes it difficult to reason about behaviour

Trait entanglements

- Makes it difficult to reason about behaviour

```
trait A {
  def foo = "a"
}
```

Trait entanglements

- Makes it difficult to reason about behaviour

```
trait A {
  def foo = "a"
}

trait B extends A {
  override def foo = "b"
}
```

Trait entanglements

- Makes it difficult to reason about behaviour

```
trait A {
  def foo = "a"
}

trait B extends A {
  override def foo = "b"
}
```

```
class C extends A with B
new C().foo
```

Trait entanglements

- Makes it difficult to reason about behaviour

```
trait A {
  def foo = "a"
}

trait B extends A {
  override def foo = "b"
}
```

```
class C extends A with B
new C().foo

"b"
```

Trait entanglements

- Makes it difficult to reason about behaviour

```
trait A {
  def foo = "a"
}

trait B extends A {
  override def foo = "b"
}
```

```
class C extends A with B
new C().foo

"b"


class D extends B with A
new D().foo
```

Trait entanglements

- Makes it difficult to reason about behaviour

```scala
trait A {
  def foo = "a"
}

trait B extends A {
  override def foo = "b"
}
```

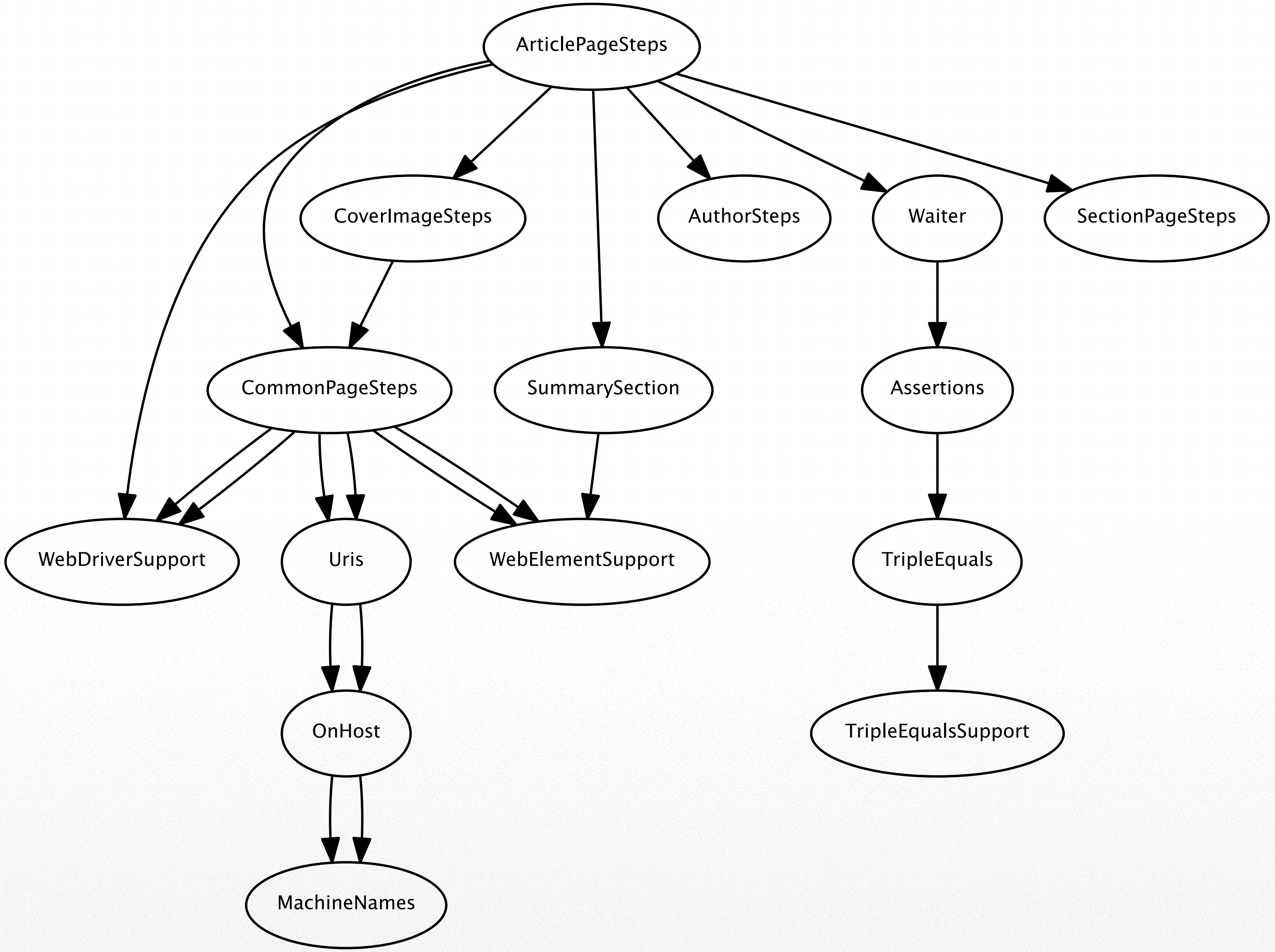
```scala
class C extends A with B
new C().foo
```

```
"b"
```

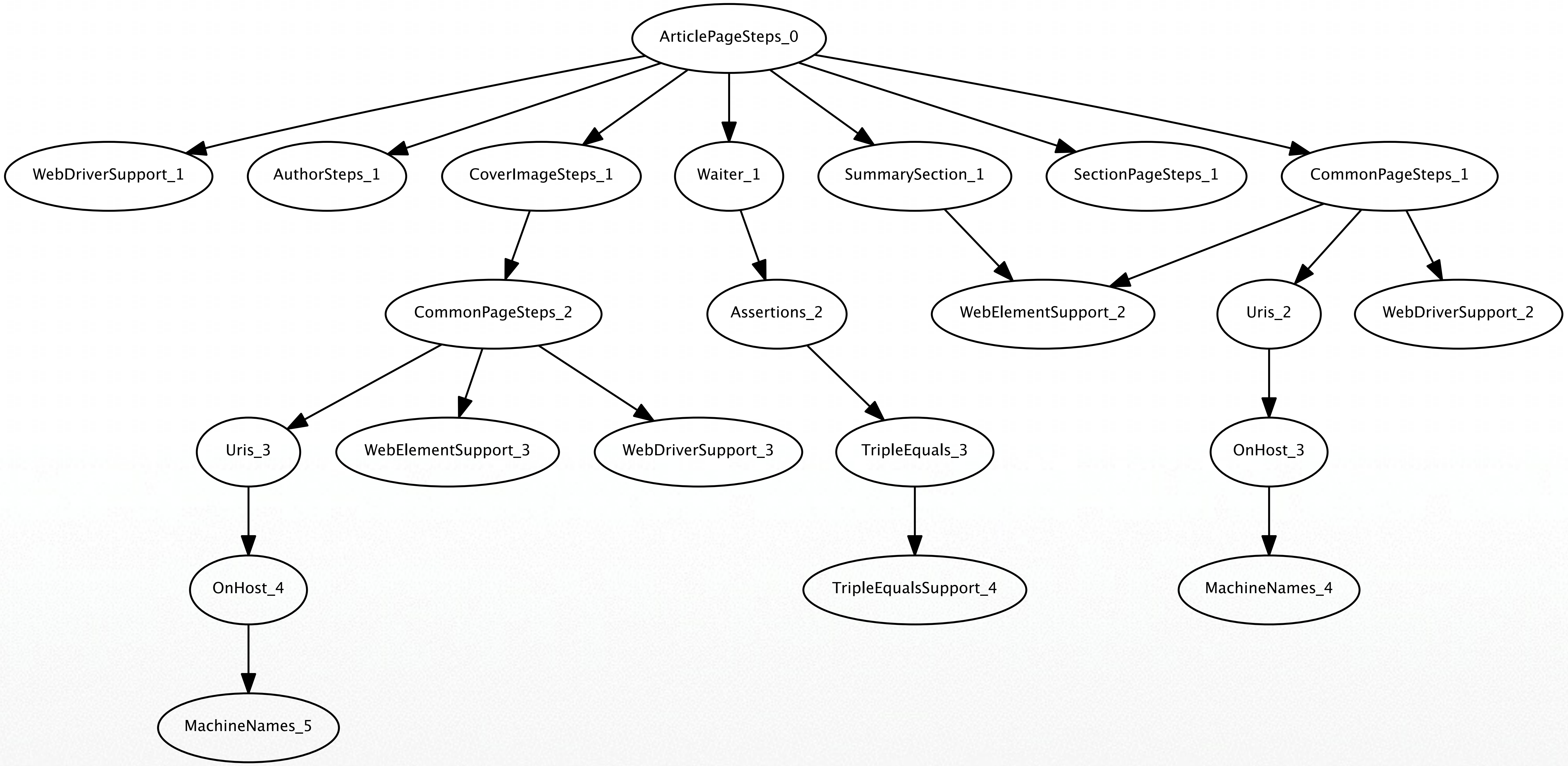
```scala
class D extends B with A
new D().foo
```

```
"b"
```

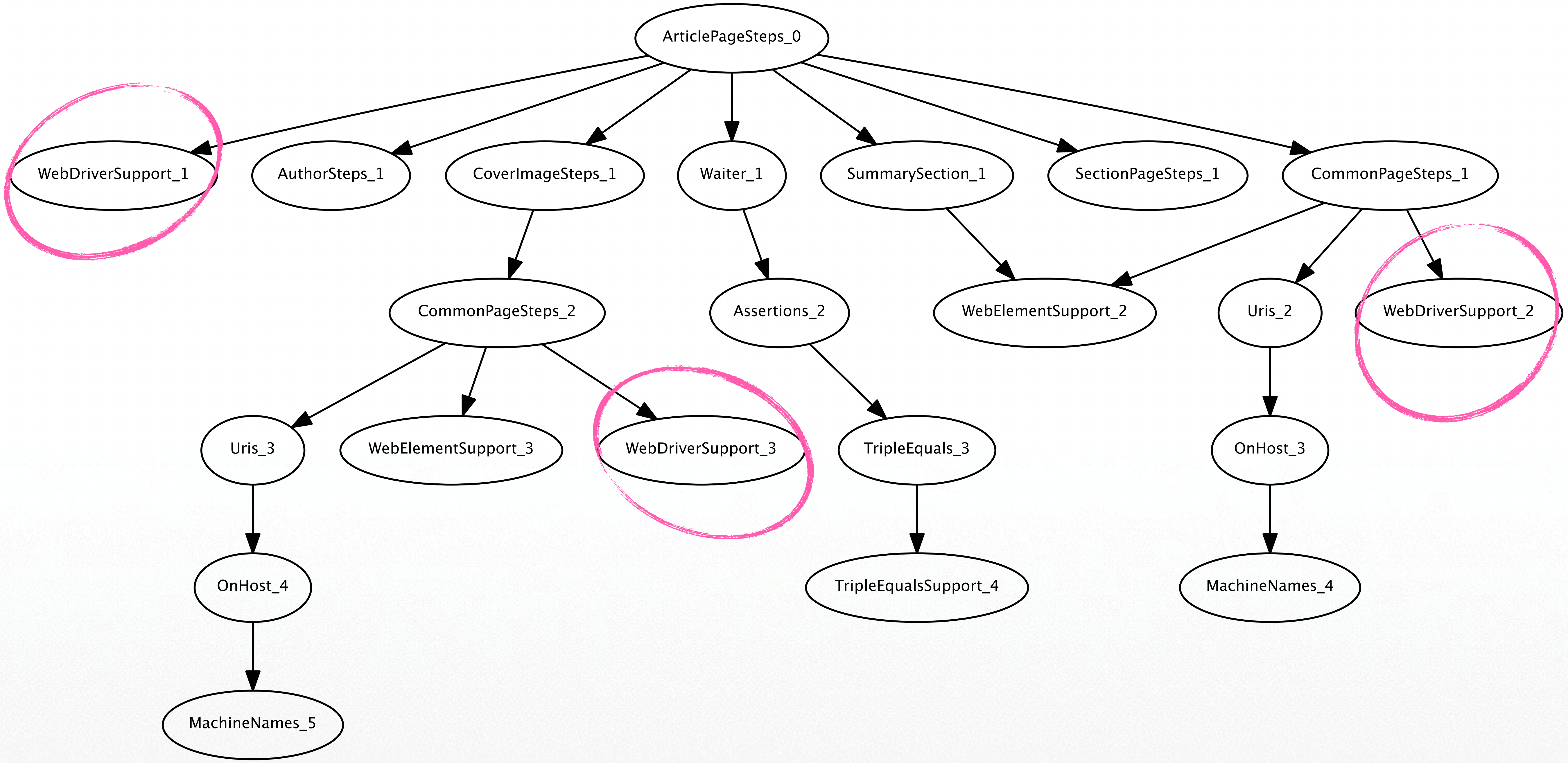
Trait entanglements (2)



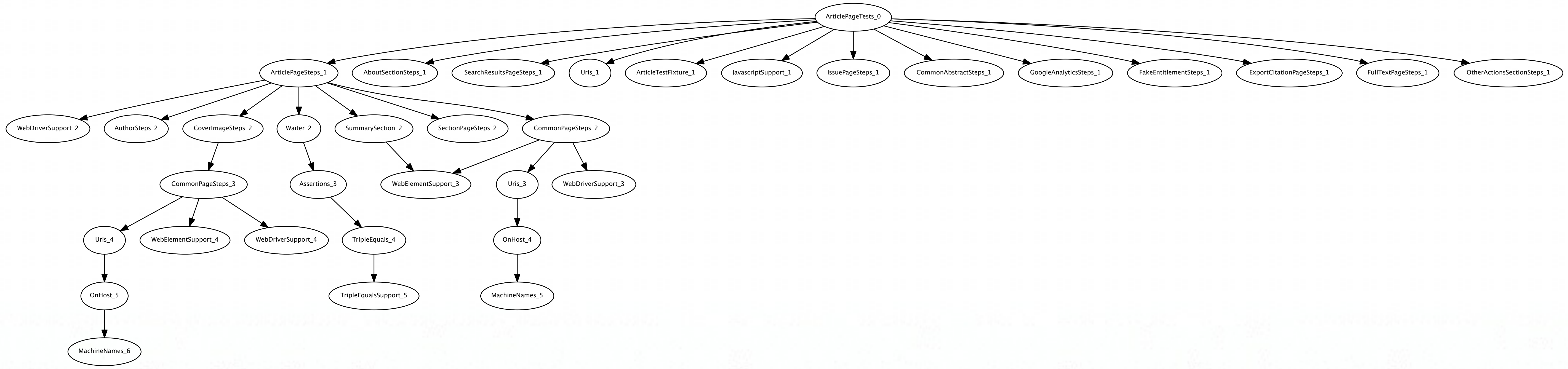
Trait entanglements (3)



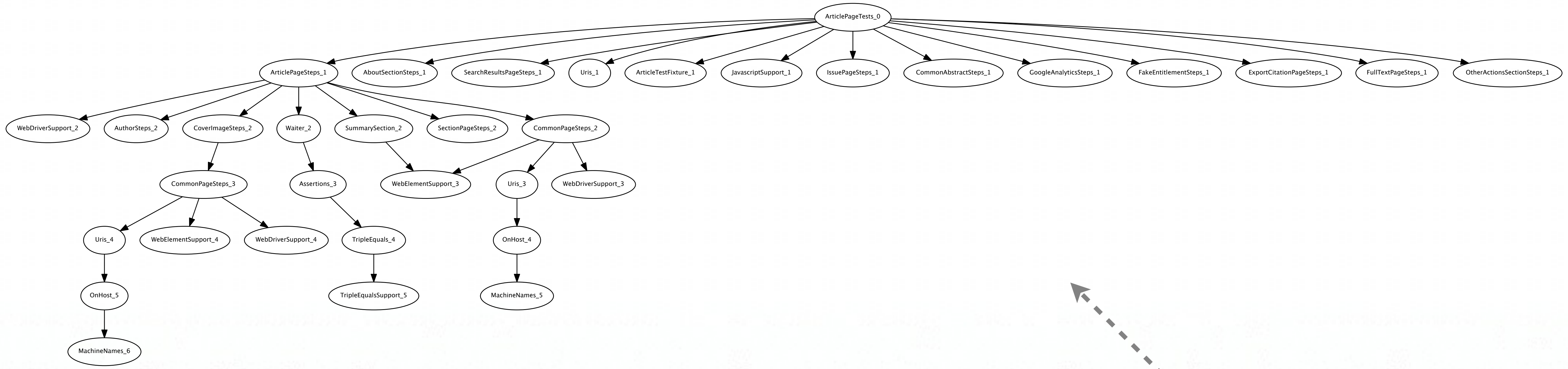
Trait entanglements (3)



Trait entanglements (4)



Trait entanglements (4)



Imagine many more circle here

So, what’s next?

Today

- We’ve delivered successfully using Scala
- Don’t think we’re more productive (pure gut feeling, though)
- We try to stick to the good parts (conventions, functional programming, pattern matching, etc.)
- Complexity, slow compilation and lack of tool support are real problems

The future

- No urgency to move away from Scala or re-write existing systems
- Java 8 is an alternative
- Smaller teams and apps will probably lead to more polyglotism (and less Scala)

Thanks

http://joinit.springer.com

@patforna
patric.fornasier@springer.com