Toward a Safer Scala

Detect errors earlier with static analysis
tinyurl.com/sd15lint
What is Static Analysis?

- Scrutinizing code without executing it
- Not exactly the compiler’s job
Why Static Analysis?

- `scalac` enables some error-prone code
  - Some Scala features demo well, work poorly
  - More on that soon!
Why Static Analysis?

- scalac enables some error-prone code
- Coding standards
  - Without enforcement, only suggestions
  - Executable best practices, rather than Markdown
  - Choose the subset of Scala your team uses
Why Static Analysis?

- scalac enables some error-prone code
- Coding standards
- Code review
  - Automate it
  - Works when expert or OSS project leader is out
What You Deserve

- Better compiler
  - Inferring Any is the root of much WTF
  - Invalid string interpolation references
What You Deserve

● Better compiler
● More hygienic code
  ○ Two spaces, no tabs, no discussion
  ○ Limits on line, function, and file length
What You Deserve

- Better compiler
- More hygienic code
- Higher quality code
  - When I said no nulls, I meant no nulls
  - Regret every var
What You Deserve

- Better compiler
- More hygienic code
- Higher quality code
- Don’t ship bugs (which are easily detected)
  - Runtime exploding `String.format()`
  - What is the head of an empty List?
Static Analysis Options

- IDE-based analysis
- Web UIs
- scalac switches
- FindBugs
- Scalastyle
- Abide
- WartRemover
- Linter
- Scapegoat
- Artima SuperSafe

Thank you, tool authors!
IDE-based Options

Inconsistencies

If not in release build process, it doesn't exist.

Condolences if you build releases with your IDE.
Web-UI-based analysis

- Doesn’t fit my biases
- Outside compile loop
- Separate from release process
- Relatively immature analysis
- Nice reporting with trending
- Consider for codebases with mixture of languages
Criteria

- **Signal to Noise**
  - ROI for errors avoided
  - False positives

- **Suppression**
  - Possible
  - Limited Scope

- **Learning curve**
  - Build integration
  - Documentation
3) X.scala:

```scala
object X {
  val a = b
  val b = 42
}
scalac X.scala
[Compiles fine]
scala -e "println(X.a)"
```
3) X.scala:

```scala
object X {
  val a = b
  val b = 42
}

scalac X.scala
[Compiles fine]

scala -e "println(X.a)"
0

Go home Scala, you're drunk!
```
Reference uninitialized value

Good: Scala 2.11 warns

Bad: Only warns on change or clean

Ugly: Can still ship it
Seth Tisue says:

“I believe almost any well-managed project should be using -Xfatal-warnings”

in SI-7529

I agree with Seth. Is your project well-managed?
scalac switches

Recommendation: Enable fatal warnings

scalacOptions  += Seq(
  "-Xfatal-warnings"
)
Set(1).equals()
Adapted Arguments

Set(1).equals()  
Compiles Always false
List(1,2,3).toSet()
Adapted Arguments

List(1,2,3).toSet()
List(1,2,3).toSet.apply(()
Adapted Arguments

Good: Deprecated around 2.10

Bad: Deprecation is just a warning

Ugly: Compiler reports
"re-run with -deprecation for details"
scalac switches

Recommendation: Enable `-deprecation`

```scala
scalacOptions +++= Seq(
  "-deprecation",
  "-Xfatal-warnings"
)
```
Caveat

deprecation applies to libraries, too.

Upgrading libraries can break build.
logger.error(
    "Failed to open %s. Error: %d"
    .format(file)
)
Wrong number of args to format()

```python
logger.error(
    "Failed to open %s. Error: %d"
    .format(file)
)
```

Runtime Exception
logger.error的优点

Failed to open $file.

Error: $IoError

"Failed to open $file." +
"Error: $IoError"
Oh, you wanted to interpolate that?

```java
logger.error(
    "Failed to open $file."
    +
    "Error: $IoError"
)
```

Mo’ money. Mo’ problems.
logger.error(
    "Failed to open $file."
    +
    "Error: $IoError"
)

Missing interpolator
scalac switches

Recommendation: Enable -Xlint

```
scalacOptions +++= Seq(
    "-Xlint",
    "-deprecation",
    "-Xfatal-warnings"
)
```
Incremental -Xlint

Scala 2.11.4+ allows selective -Xlint

Run `scalac -Xlint:help` for details

Incremental approach key to existing projects
sealed trait ADT

object A extends ADT

object B extends ADT

final val C = 3

val adt = Seq(A, B, C)
sealed trait ADT
object A extends ADT
object B extends ADT
final val C = 3
val adt = Seq(A,B,C)

Inferred Seq[Any]
val a = Seq(1, 2)
val a = Seq("3", "4")
val crossProduct = a.flatMap { x =>
    b.flatMap { y => Seq(x, y) }
}
val a = Seq(1, 2)
val a = Seq("3", "4")
val crossProduct = a.flatMap { x => b.flatMap { y => Seq(x, y) } }

Inferred Seq[Seq[& Any]]
Inferring Any With -Xlint

A type was inferred to be `Any`; this may indicate a programming error.

You’re right, scalac! Thank you!
scalac switches

Recommendation: No seriously enable `-Xlint`

```
scalacOptions +++= Seq(
  "-Xlint",
  "-deprecation",
  "-Xfatal-warnings"
)
```
scalac switches

- Signal/Noise
- Suppression
- Learning Curve
Cross Building

Stephen Compall pointed out complexities when targeting multiple versions of scala.

Solution: Only add the switch to latest version.
FindBugs

- Venerable Java static analysis tool
- findbugs4sbt plugs it into sbt
- High false positive rate for Scala
  - Baffled by synthetic code
FindBugs

Signal/Noise
Suppression
Learning Curve
Scalastyle

- Oriented toward “style”
- Uses Scalariform’s parser
  - Not a scalac plugin
- sbt plugin for easy integration
  - Executed via scalastyle command
  - Can be tied to compile or test commands
- Used by Martin O’s Coursera classes
- Mature, fairly well-documented project
Incrementally configured via XML

<scalastyle>
  <check level="error" class="NullChecker" enabled="false">
    <check level="error" class="FileLengthChecker" enabled="true">
      <parameters>
        <parameter name="maxFileLength"><![CDATA[1500]]></parameter>
      </parameters>
    </check>
  </check>
</scalastyle>
Incrementally configured via XML

<scalastyle>
<check level="error" class="NullChecker" enabled="true">
<check level="error" class="FileLengthChecker" enabled="true">
<parameters>
<parameter name="maxFileLength"><![CDATA[1500]]></parameter>
</parameters>
</check>
</check>
</scalastyle>
Incrementally configured via XML

<scalastyle>
  <check level="error" class="NullChecker" enabled="true">
  </check>
  <check level="error" class="FileLengthChecker" enabled="true">
    <parameters>
      <parameter name="maxFileLength"><![CDATA[750]]></parameter>
    </parameters>
  </check>
</scalastyle>
Suppress via comments

// scalastyle:off null
TerribleJavaApi.madeMe(null)
// scalastyle:on null

Consistent with Rule 0.1
Suppress via comments

Bad.api(null)  // scalastyle:ignore null
1. Executable coding standard
2. Ban “Better Java”
3. Encourage Good Practices
Executable coding standard

- Line, file, and function length
- Cyclomatic complexity
- Argument list length
- Functions per type
- Indentation
- Magic numbers
- Identifier names match regex
Ban “Better Java”

- Ban null
- Ban var
- Ban return
- Ban while
- Ban clone() method
Encourage Good Practices

- Explicit return type for public methods
- Don’t ship `???.` and `println`
- Require `equals/hashCode` if other defined
- Any `equals` must override `Object.equals`
- Disallow structural types
  - Type lambdas require suppression
- Simplify boolean expressions
Scalastyle Extensibility

- Simple: Regexes to find banned patterns
- Complex: Custom rules are possible but, cumbersome
Scalastyle

- Rule set feels broad and unfocused
- Implements rules for both sides of controversies
  - DisallowSpaceAfterTokenChecker
  - EnsureSingleSpaceAfterTokenChecker
Scalastyle

Signal/Noise

Suppression

Learning Curve
If your coding standard isn’t automatically and uniformly applied, you don’t have a coding standard.
Vehicle for passive-aggressive misanthropy
 Scalariform ❤❤❤❤

- Configurably format code
- Consistent formatting regardless of editor
- Preclude petty code review comments
- Triggered manually or on every compile
- sbt and editor integrations
Abide

- New project
- Unreleased 0.1-SNAPSHOT
- Must build from source. Artifacts not online.
- github.com/scala/scala-abide
  - Driven by Typesafe
    - Iulian Dragos
    - Adriaan Moors
- Documentation is work in progress
Rules have mainstream flavor

- Only bans `var` in public context
- Detects when a `var` instead of `val`
- Unused locals detection. WANT!
  - Local vars, defs (False positives)
  - Private class members
  - Method arguments
- Akka specific rules
  - Avoid capturing unintended context
- Implementing `scalac -Xlint` and `-Y?`
Abide

- Signal/Noise
- Suppression
- Learning Curve
val testUsers = Seq(
  ("Flintstone, Fred", 42),
  ("Rubble", "Betty", 35)
)
val testUsers = Seq(
  ("Flintstone, Fred", 42),
  ("Rubble", "Betty", 35)
)
val testUsers = Seq(
  ("Flintstone, Fred", 42),
  ("Rubble", "Betty", 35)
)

Inferred Seq[Product with Serializable]
WartRemover

- Brian McKenna
- Accurately and well documented
- Easy sbt integration
- Extensibility is reasonably well documented
wartremoverExcluded  +=  Seq(
    baseDirectory.value  /  "File.scala",
   )
1. Avoid Problematic Inference
2. Ban “Better Java”
3. Partial Methods which throw
Avoid Problematic Inference

- Product
- Serializable
- Nothing
- Any
Ban “Better Java”

- Ban null
- Ban var
- Ban return
- Ban throw
Partial Methods which throw

- `List.head`, `.tail`, `.last`, `.reduce`
- `Option.get`
- `scala.util.Either.get`
My team calls them “YOLO methods”
Why tolerate methods which blow up if you catch them in the wrong mood?
wartremoverErrors += Warts.all
wartremoverErrors ++= Seq(
  Wart.Any,
  Wart.Serializable,
  Wart.Product
)
WartRemover

- Signal/Noise
- Suppression
- Learning Curve
WartRemover

- Focused and opinionated
- Recommendation: Best suited to
  1. Greenfield, or
  2. Already clean projects, or
  3. Writing principled Scala
- Slow running: May need separate build target
def f(s: String) = {
  s.split(':') match {
    case Array(k, v) => k -> v
    case a => error("bad: " + a)
  }
}
def f(s: String) = {
  s.split(':') match {
    case Array(k, v) => k -> v
    case a => error("bad: " + a)
  }
  Array.toString()
}
bad:  [Ljava.lang.String;@6f89341e
def g(l: List) = {
    if (l.length == 0) 0
    else l.reduce(...)
}
```python
def g(l: List) = {
    if (l.length == 0) 0
    else l.reduce(...) 
}
```

List.length is O(n)
Use isEmpty()
Linter

- Many years of many forks
  - Jorge Ortiz: com.foursquare.lint in 2012
  - Matic Potočnik (@HairyFotr): current fork
- Documentation is middling
- scalac plugin without sbt plugin
- Configured via scalac switches
Implements 50% more rules than Scalastyle

- Guess what they do based on name or reading unit tests
- Tend toward more traditional linter-style
- Easy to be overwhelmed
Linter Found

- `String.format()` call with mismatched args
- Unused function argument
- `aList.length == 0` <- use `.isEmpty`
- Using string interpolation unnecessarily
- Calling `.toString` on Array
- Close `scala.io.Source`
- Inferring Nothing
Linter Also Found

- Casting instead of `.toByte`
- Calling `.toSeq` on a `Seq`
- Calling `.toString` on a `String`
- Calling `.toList` on a `List`
- Calling `.toMap` on a `Map`
- Use `JavaConversions`, not `JavaConverters`
Linter

- Incomplete docs
- Definitely has value; identified real bugs
- No false positives
- Suppression via comments.*
- Relatively slow running
- Wish: short list of highest ROI checks
- Wish: sbt plugin
Linter

Signal/Noise

Suppression

Learning Curve
seq.find(_ > 42).isDefined
seq.find(_ > 42).isDefined
seq.exists(_ > 42)
seq.exists(_ == "the one")
seq.exists(_ == "the one")
seq.contains("the one")
Scapegoat

- Stephen Samuel
- A bit over a year old
- Development has tailed off since August
- sbt plugin makes it easy to get started
- Fairly good docs. A little behind the code.
- File level, path regex based suppression
WartRemover
++
Linter
++
Scalastyle
Lonely sealed class
Match instead of partial function
Duplicated import
length == 0
var could be a val
Unused method parameter
Use .contains(y) instead of .exists(_ == y)
What I dislike

- Overwhelming. Enables 100+ rules by default
- Somewhat unclear how to disable rule
- Bans `final case class`
- Bans wildcard imports. Bedazzling?
- At least 5 different classes of false positive
Scapegoat

- Signal/Noise
- Suppression
- Learning Curve
Array(1) == Array(1)
Array(1) == Array(1)

Compiles
Always false
Artima SuperSafe™

- Commercial offering
- $60/user/year
- From Artima, who make ScalaTest
- 1.0.0 published in January
Artima SuperSafe™

- Small rule set so far
- Rules focus on problems of equality
- Also rule for suspicious inference
- Emphasis on no false positives
  - I found one. Artima published update next day.
- No support for suppression
- Relatively minimal impact on build time
Artima SuperSafe

- Signal/Noise
- Suppression
- Learning Curve
Recommendation Summary

**Adopt:**
- scalac switches
- Scalastyle

**Consider:**
- WartRemover
- Linter

**Watch:**
- SuperSafe
- Abide
- Scapegoat
<table>
<thead>
<tr>
<th>Tool</th>
<th>S/N</th>
<th>Sup</th>
<th>LC</th>
</tr>
</thead>
<tbody>
<tr>
<td>scalac switches</td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Red" /></td>
<td><img src="#" alt="Yellow" /></td>
</tr>
<tr>
<td>Scalastyle</td>
<td><img src="#" alt="Yellow" /></td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Yellow" /></td>
</tr>
<tr>
<td>Abide</td>
<td><img src="#" alt="Unknown" /></td>
<td><img src="#" alt="Unknown" /></td>
<td><img src="#" alt="Unknown" /></td>
</tr>
<tr>
<td>WartRemover</td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Yellow" /></td>
<td><img src="#" alt="Green" /></td>
</tr>
<tr>
<td>Linter</td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Yellow" /></td>
<td><img src="#" alt="Red" /></td>
</tr>
<tr>
<td>Scapegoat</td>
<td><img src="#" alt="Yellow" /></td>
<td><img src="#" alt="Yellow" /></td>
<td><img src="#" alt="Yellow" /></td>
</tr>
<tr>
<td>Artima SuperSafe</td>
<td><img src="#" alt="Yellow" /></td>
<td><img src="#" alt="Red" /></td>
<td><img src="#" alt="Yellow" /></td>
</tr>
</tbody>
</table>
1. scalac switches
   a. Adapted Rob Norris’ post

2. Scalastyle coding standard
   a. Such as alexandru’s list

3. lint build target
   a. WartRemover inference and partial
   b. Linter
   c. Slow scalac switches
Skeleton Project with Static Analysis
tinyurl.com/lint-skel
Existing Project

Start small; tighten screws

1. Start with all rules disabled
2. Make linter gate in CI
3. Pick a rule team agrees on
4. Clean up code or suppress
5. Enable rule as error
6. Goto 3
Broken Windows

“Tip 4: Don’t Live with Broken Windows

Fix each as soon as it is discovered.”
TL;DR

scalacOptions += Seq(
  "-Xlint",
  "-deprecation",
  "-Xfatal-warnings"
)
Questions?

Slides: tinyurl.com/sd15lint