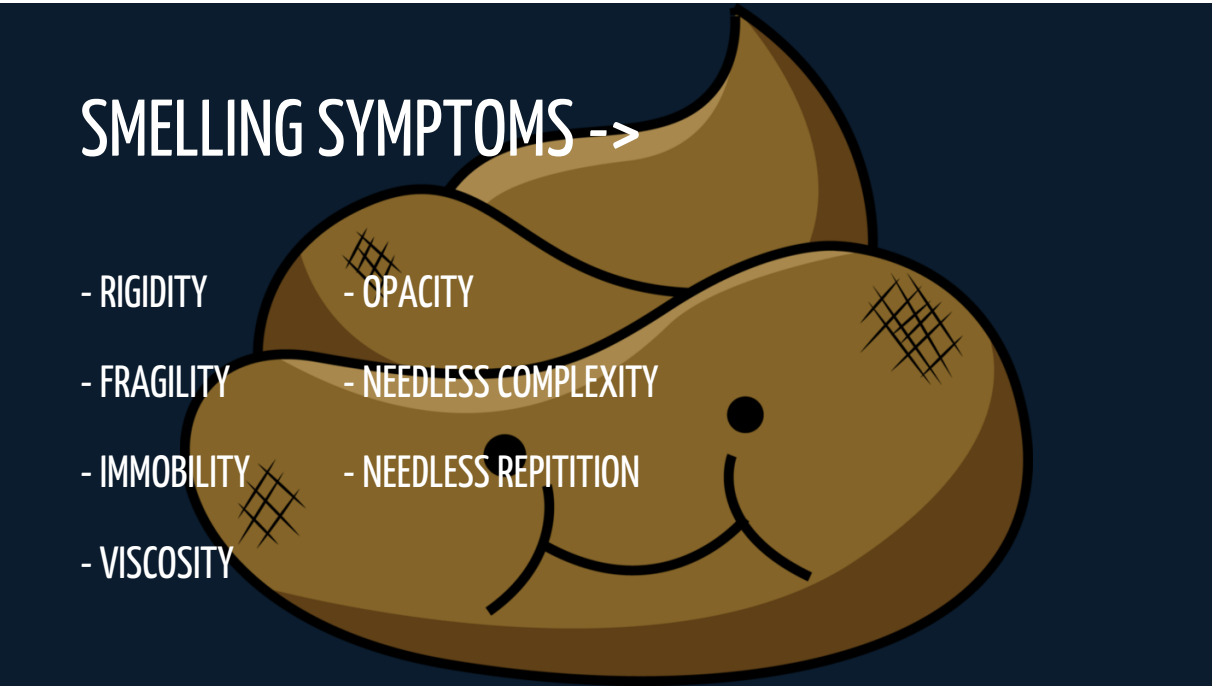

ARCHITECTING WELL-STRUCTURED JAVA APPLICATIONS
Eduards Sizovs
@eduardsi


MOST APPS BEGIN LIFE SMALL AND NEAT.

TIME GOES BY...

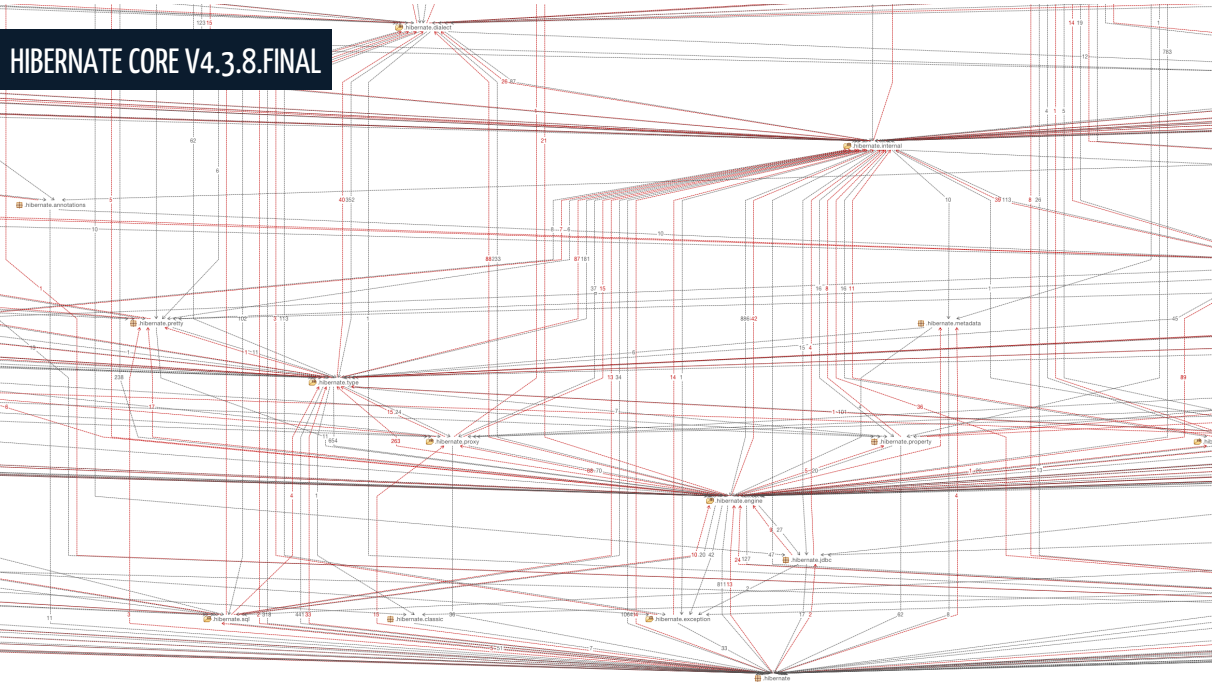

HELLO. I AM YOUR ROTTING ENTERPRISE APP.

SMELLING SYMPTOMS ->

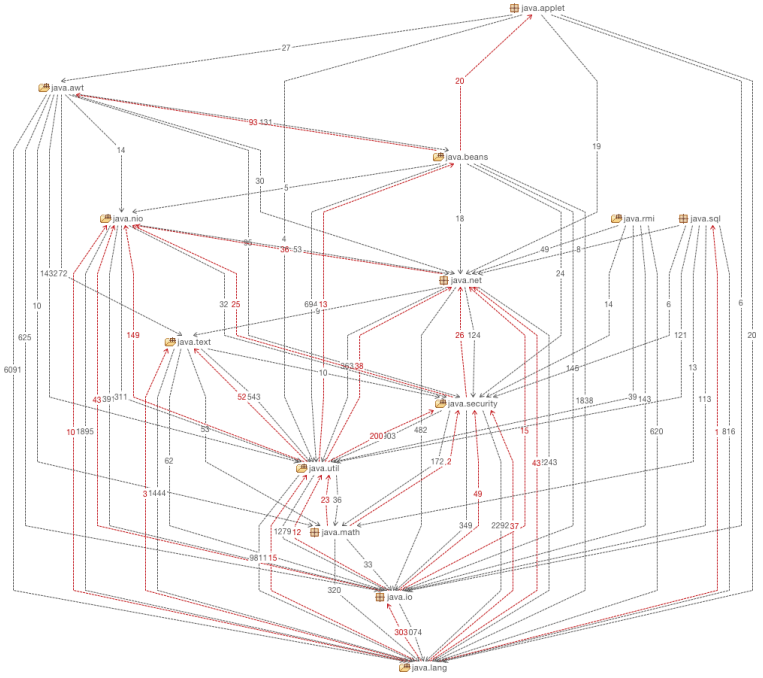


- RIGIDITY
- FRAGILITY
- IMMOBILITY
- VISCOSITY
- OPACITY
- NEEDLESS COMPLEXITY
- NEEDLESS REPITITION

HIBERNATE CORE V4.3.8.FINAL



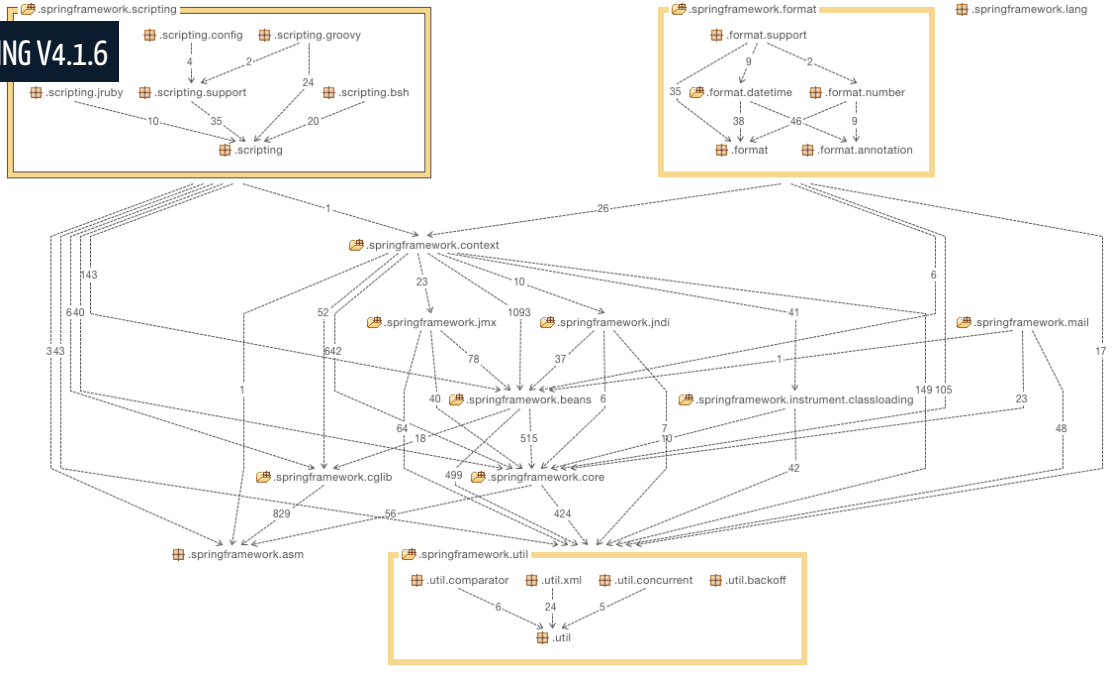
JDK V1.7.0_51


java.applet
java.awt
java.beans
java.io
java.lang
java.math
java.net
java.nio
java.rmi
java.sql
java.text
java.security
java.util

A JDK CODE BASE IS DEEPLY INTERCONNECTED AT BOTH THE API AND THE IMPLEMENTATION LEVELS, HAVING BEEN BUILT OVER MANY YEARS PRIMARILY IN THE STYLE OF A MONOLITHIC SOFTWARE SYSTEM. WE'VE SPENT CONSIDERABLE EFFORT ELIMINATING OR AT LEAST SIMPLIFYING AS MANY API AND IMPLEMENTATION DEPENDENCES AS POSSIBLE, SO THAT BOTH THE PLATFORM AND ITS IMPLEMENTATIONS CAN BE PRESENTED AS A COHERENT SET OF INTERDEPENDENT MODULES, BUT SOME PARTICULARLY THORNY CASES REMAIN.

(C) MARK REINHOLDS, CHIEF ARCHITECT OF THE JAVA PLATFORM

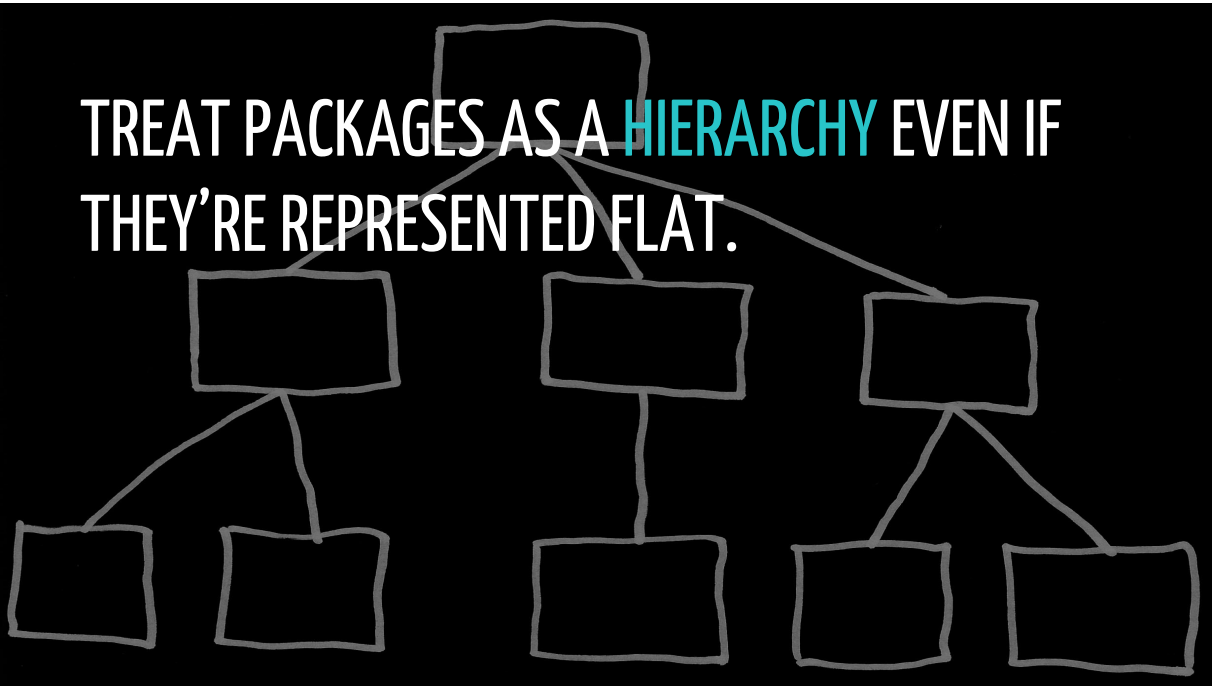
SPRING V4.1.6


.springframework.scripting
.scripting.config
.scripting.groovy
.scripting.jruby
.scripting.support
.scripting.bsh
.scripting
.springframework.format
.format.support
.format.datetime
.format.number
.format
.format.annotation
.springframework.lang
.springframework.context
.springframework.jmx
.springframework.jndi
.springframework.mail
.springframework.beans
.springframework.instrument.classloading
.springframework.cglib
.springframework.core
.springframework.asm
.springframework.util
.util.comparator
.util.xml
.util.concurrent
.util.backoff
.util

PRINCIPLES.


PACKAGE IS THE FIRST-CLASS CITIZEN AND
KEY ELEMENT OF LOGICAL DESIGN.
THE
DEPAUL
BOX
COMPANY

TREAT PACKAGES AS A HIERARCHY EVEN IF THEY'RE REPRESENTED FLAT.



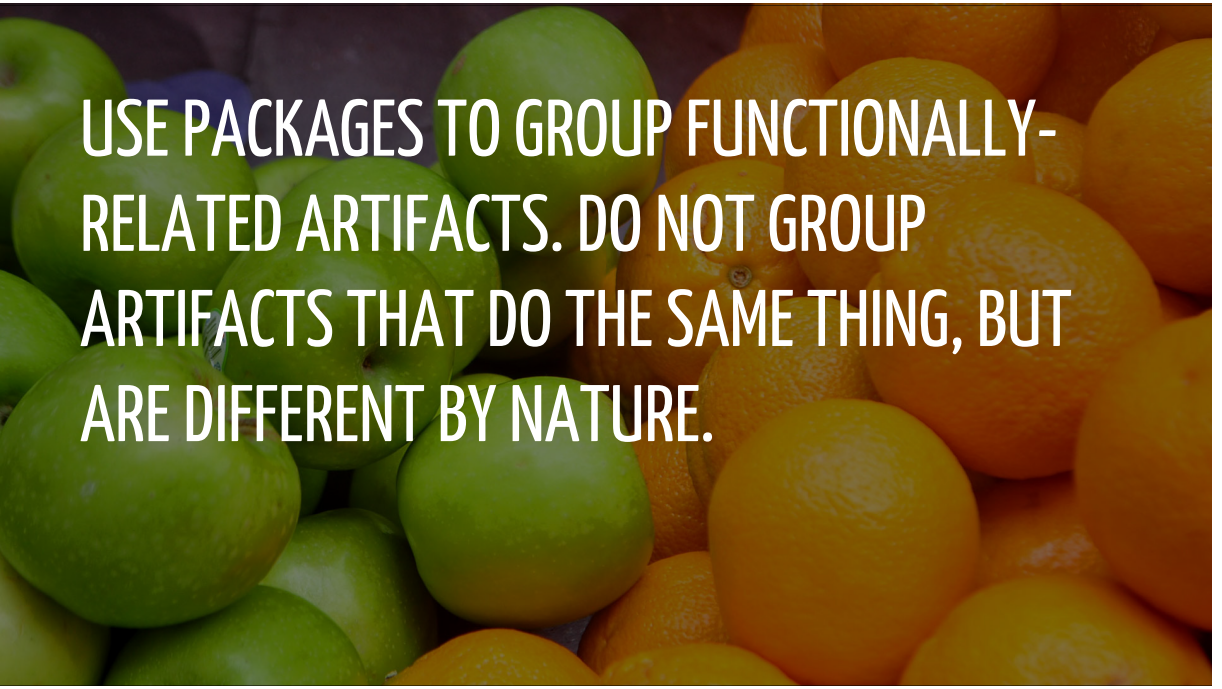
```
io.shwitter.user
io.shwitter.user.registration
io.shwitter.user.profile
io.shwitter.timeline
```

```
io.shwitter.user                (part of)
io.shwitter.user.registration
io.shwitter.user.profile
io.shwitter.timeline
```

```
io.shwitter.user
io.shwitter.user.registration  (part of)
io.shwitter.user.profile
io.shwitter.timeline
```

```
io.shwitter.user
io.shwitter.user.registration
io.shwitter.user.profile        (part of)
io.shwitter.timeline
```

```
io.shwitter.user
io.shwitter.user.registration
io.shwitter.user.profile
io.shwitter.timeline                (part of)
```


USE PACKAGES TO GROUP FUNCTIONALLY-RELATED ARTIFACTS. DO NOT GROUP ARTIFACTS THAT DO THE SAME THING, BUT ARE DIFFERENT BY NATURE.

```
io.shwitter.controller
io.shwitter.dao
io.shwitter.domain
io.shwitter.services
io.shwitter.exceptions
```

```
io.shwitter.controller
io.shwitter.dao
io.shwitter.domain
io.shwitter.services
io.shwitter.exceptions

UserController, TimelineController...
```

```
io.shwitter.controller
io.shwitter.dao
io.shwitter.domain
io.shwitter.services
io.shwitter.exceptions

UserDAO, TimelineDAO...
```

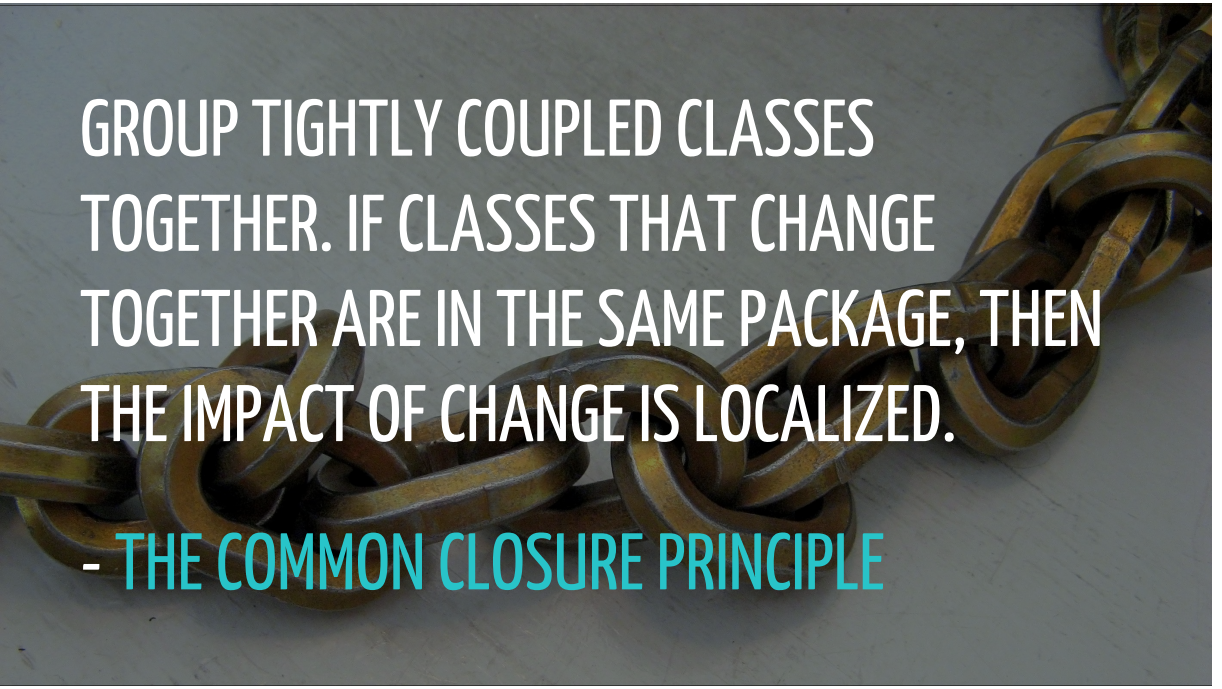
```
io.shwitter.controller
io.shwitter.dao
io.shwitter.domain
io.shwitter.services
io.shwitter.exceptions

User, Timeline...
```

```
io.shwitter.controller
io.shwitter.dao
io.shwitter.domain
io.shwitter.services
io.shwitter.exceptions

RegistrationService, TimelineService...
```

```
io.shwitter.controller
io.shwitter.dao
io.shwitter.domain
io.shwitter.services
io.shwitter.exceptions
```

```
io.shwitter.controller
io.shwitter.dao
io.shwitter.domain
io.shwitter.services
io.shwitter.exceptions

io.shwitter.user
io.shwitter.user.registration
io.shwitter.user.profile
io.shwitter.timeline
```

```
io.shwitter.controller
io.shwitter.dao
io.shwitter.domain
io.shwitter.services
io.shwitter.exceptions

io.shwitter.user
io.shwitter.user.registration
io.shwitter.user.profile
io.shwitter.timeline

User, UserDAO, UserController...
```


GROUP TIGHTLY COUPLED CLASSES TOGETHER. IF CLASSES THAT CHANGE TOGETHER ARE IN THE SAME PACKAGE, THEN THE IMPACT OF CHANGE IS LOCALIZED.
- THE COMMON CLOSURE PRINCIPLE


MAKE SURE ARTIFACTS DO NOT ESCAPE.


MAKE PACKAGES HIGHLY COHESIVE BY FOLLOWING SINGLE RESPONSIBILITY PRINCIPLE.
CAN
BOTTLE


KEEP PACKAGES LOOSELY COUPLED, IDEALLY
– COMPLETELY INDEPENDENT. REFLECTION
DOESN'T COUNT.

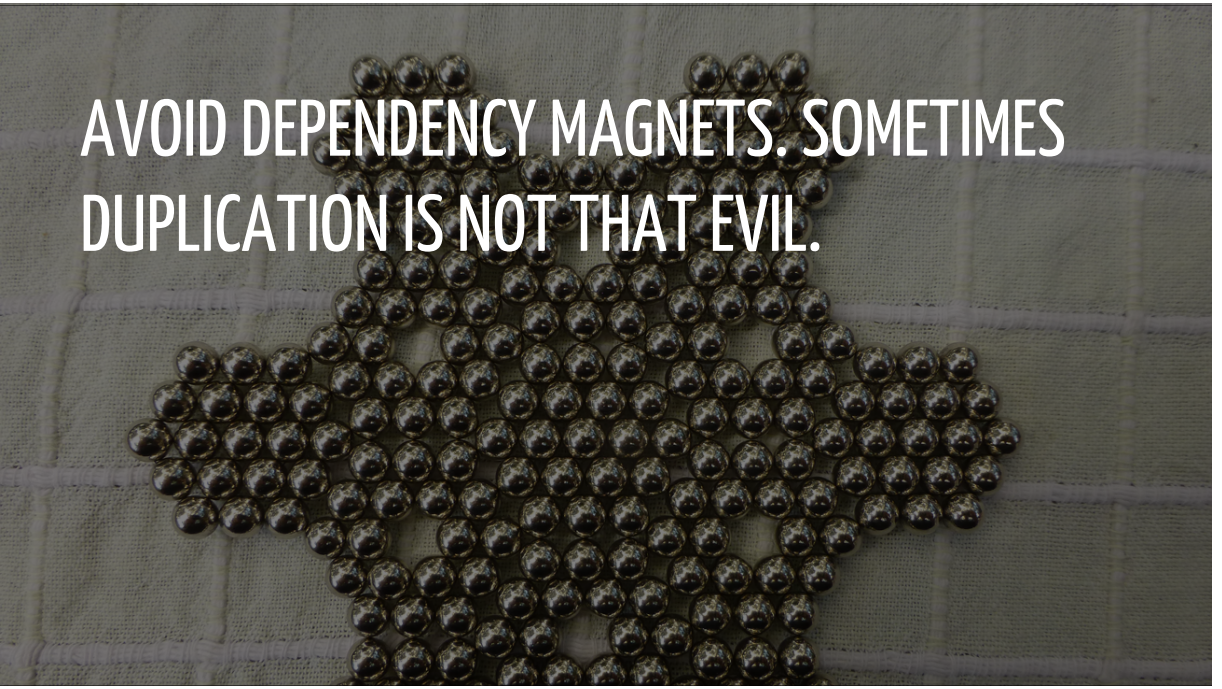
```
package io.shwitter.user

@Entity
class User {
  // name, password etc.
}
```

```
package io.shwitter.timeline

@Entity
class Timeline {
  @OneToOne User user;
}
```

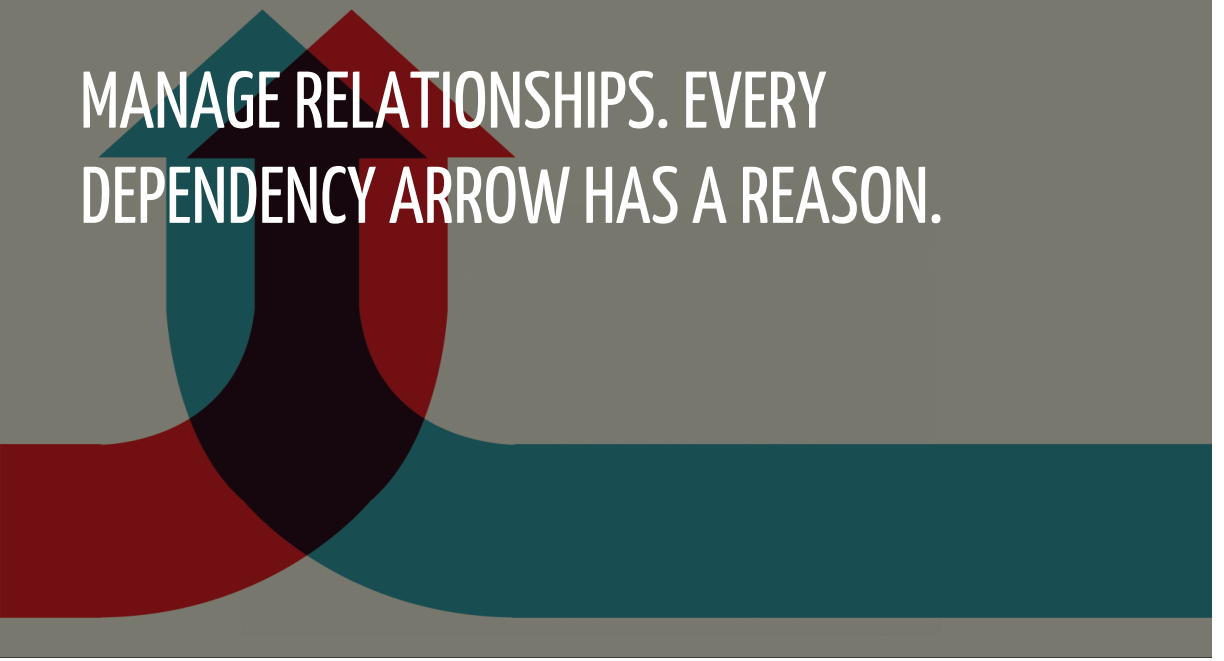
```
package io.shwitter.user

@Embeddable
class UserId {
  Long value;
}
```

```
package io.shwitter.timeline

@Entity
class Timeline {
  @Embedded UserId userId;
}
```

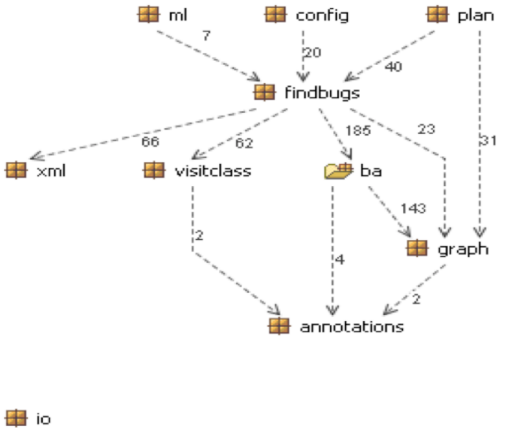

PROVIDE SLIM PACKAGE INTERFACE AND HIDE
IMPLEMENTATION DETAILS.


AVOID DEPENDENCY MAGNETS. SOMETIMES
DUPLICATION IS NOT THAT EVIL.

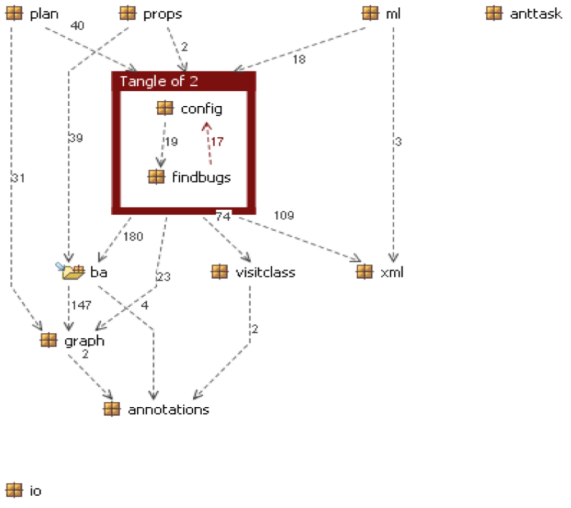
MANAGE RELATIONSHIPS. EVERY DEPENDENCY ARROW HAS A REASON.



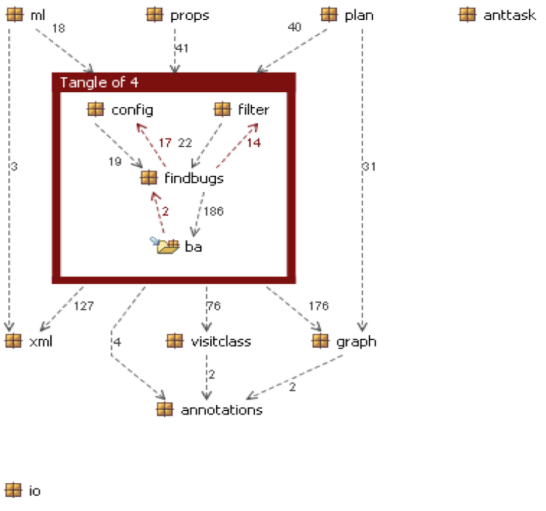
FINDBUGS V1.0 - A GREAT START



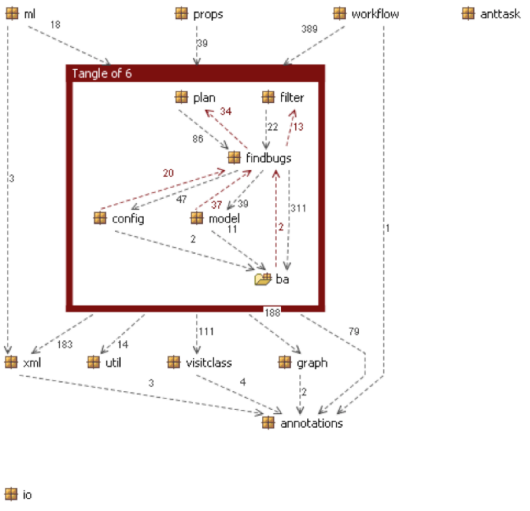
FINDBUGS V1.1 – IMPERFECTION CREEPS IN



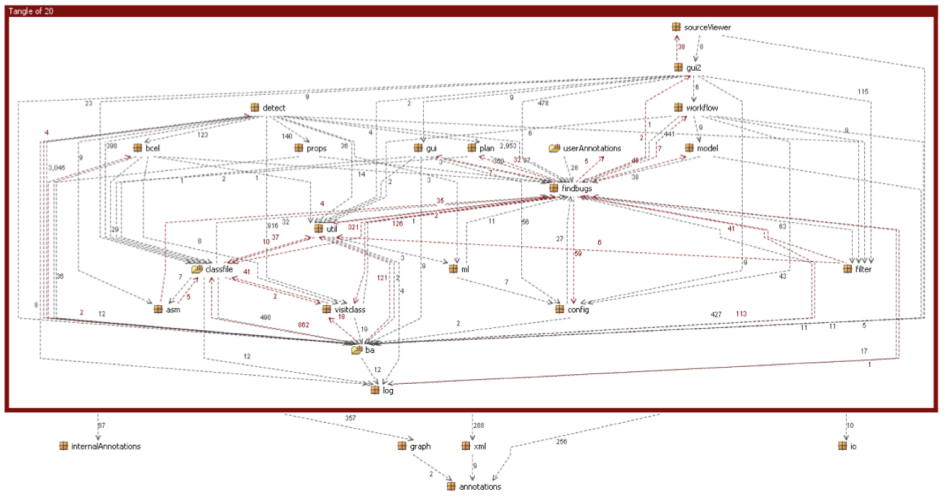
FINDBUGS V1.2 – IMPERFECTION TAKES HOLD



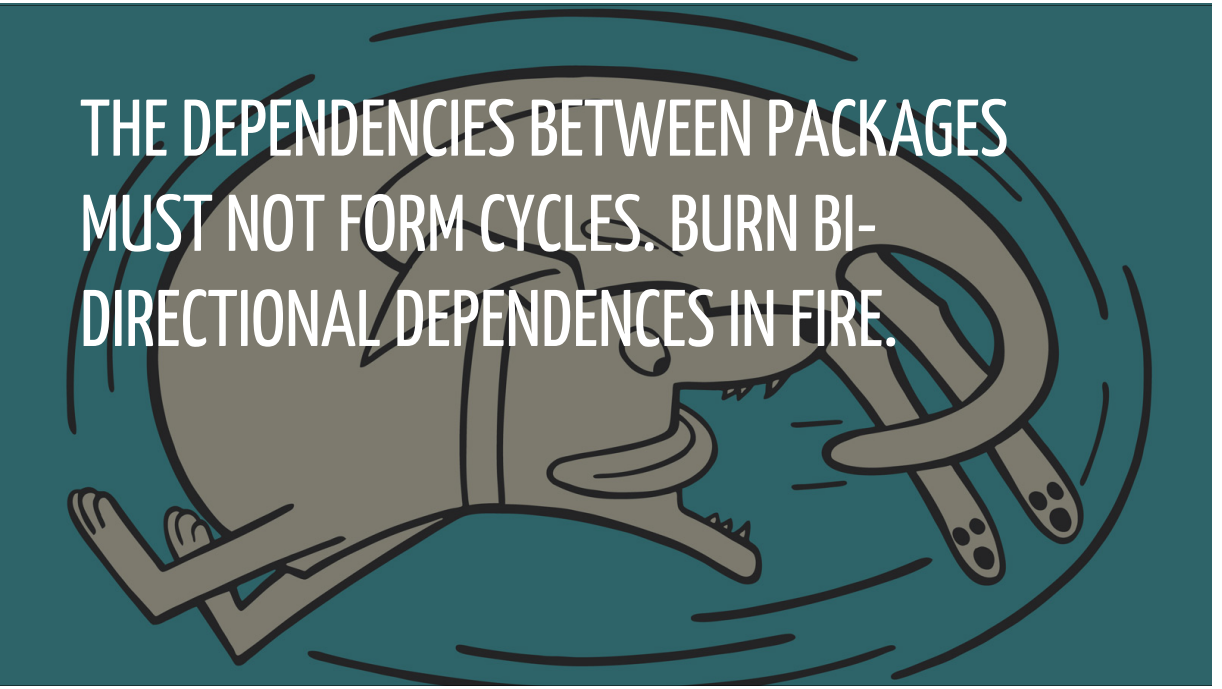
FINDBUGS V1.3 – CHAOS BEGINS



FINDBUGS V1.4 – EXPLOSION



THE DEPENDENCIES BETWEEN PACKAGES MUST NOT FORM CYCLES. BURN BI-DIRECTIONAL DEPENDENCES IN FIRE.

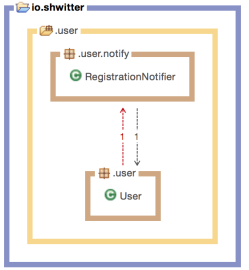


HOW?

MERGING

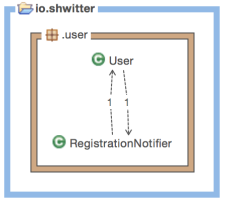
```
package io.shwitter.user
class User {
  void register(RegistrationNotifier notifier) {}
}
```

```
package io.shwitter.user.notify
class RegistrationNotifier {
  void notify(User user) {}
}
```



MERGING - REPACKAGING

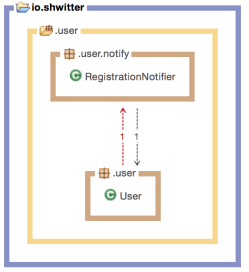
```
package io.shwitter.user
class User {
  void register(RegistrationNotifier notifier) {}
}

class RegistrationNotifier {
  void notify(User user) {}
}
```



DEPENDENCY INVERSION

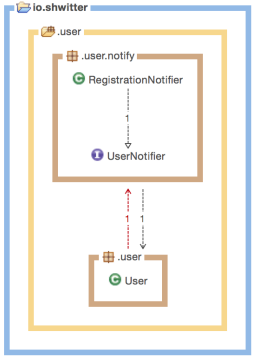
```
package io.shwitter.user
class User {
  void register(RegistrationNotifier notifier) {}
}
```

```
package io.shwitter.user.notify
class RegistrationNotifier {
  void notify(User user) {}
}
```



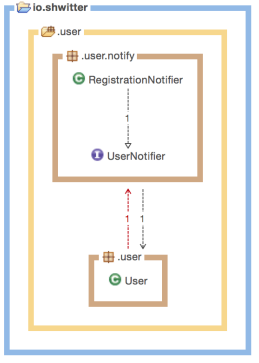
DEPENDENCY INVERSION - REFACTORING STEP 1

```
package io.shwitter.user
class User {
  void register(RegistrationNotifier notifier) {}
}


package io.shwitter.user.notify
class RegistrationNotifier implements UserNotifier {
  void notify(User user) {}
}
interface UserNotifier {
  void notify(User user) {}
}
```



DEPENDENCY INVERSION - REFACTORING STEP 2

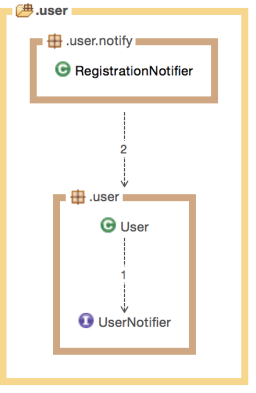
```
package io.shwitter.user
class User {
  void register(UserNotifier notifier) {}
}


package io.shwitter.user.notify
class RegistrationNotifier implements UserNotifier {
  void notify(User user) {}
}
interface UserNotifier {
  void notify(User user) {}
}
```



DEPENDENCY INVERSION - REFACTORING STEP 3

```
package io.shwitter.user
class User {
  void register(UserNotifier notifier) {}
}
interface UserNotifier {
  void notify(User user) {}
}
```

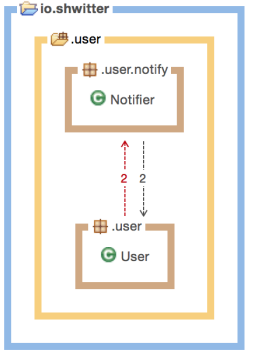
```
package io.shwitter.user.notify
class RegistrationNotifier implements UserNotifier {
  void notify(User user) {}
}
```



ESCALATION

```
package io.shwitter.user
class User {
  void register(Notifier n) { n.notify(this); }
}
```

```
package io.shwitter.user.notify
class Notifier {
  void notify(User user) { sendEmailTo(user.email()); }
}
```

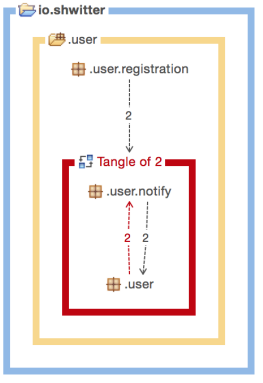


ESCALATION - REFACTORING STEP 1

```
package io.shwitter.user
class User {
  void register(Notifier n) { n.notify(this); }
}
```

```
package io.shwitter.user.notify
class Notifier {
  void notify(User user) { sendEmailTo(user.email()); }
}
```

```
package io.shwitter.user.registration
class Registrator {
  void register(User user, Notifier n) {}
}
```

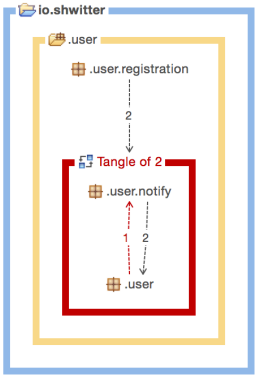


ESCALATION - REFACTORING STEP 2

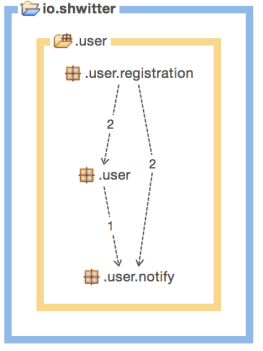
```
package io.shwitter.user
class User {
  void register() { }
}
```

```
package io.shwitter.user.notify
class Notifier {
  void notify(User user) { sendEmailTo(user.email()); }
}
```

```
package io.shwitter.user.registration
class Registrator {
  void register(User user, Notifier n) { n.notify(user); }
}
```



ESCALATION - REFACTORING STEP 3




```
package io.shwitter.user
class User {
  void register() { }
}
```

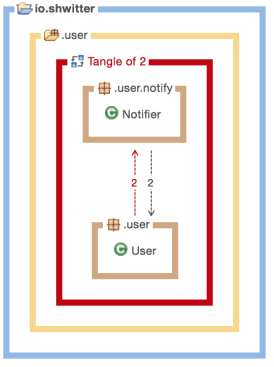
```
package io.shwitter.user.notify
class Notifier {
  void notify(String emailAddress) { sendEmailTo(emailAddress); }
}
```

```
package io.shwitter.user.registration
class Registrator {
  void register(User user, Notifier n) { n.notify(user.email()); }
}
```

DEMOTION

```
package io.shwitter.user
class User {
  void register(Notifier n) { n.notify(this); }
}
```

```
package io.shwitter.user.notify
class Notifier {
  void notify(User user) { sendEmailTo(user.email()); }
}
```

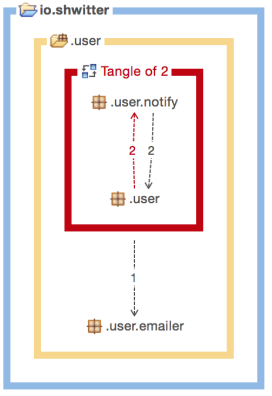


DEMOTION - REFACTORING STEP 1

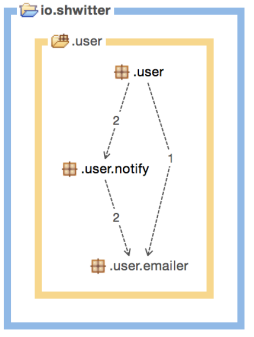
```
package io.shwitter.user
class User implements EmailHolder {
  void register(Notifier n) { n.notify(this); }
}
```

```
package io.shwitter.user.notify
class Notifier {
  void notify(User user) { sendEmailTo(user.email()); }
}
```

```
package io.shwitter.emailer
interface EmailHolder  {
  String email();
}
```



DEMOTION - REFACTORING STEP 2




```
package io.shwitter.user
class User implements EmailHolder {
  void register(Notifier n) { n.notify(this); }
}
```

```
package io.shwitter.user.notify
class Notifier {
  void notify(EmailHolder emailHolder) { sendEmailTo(emailHolder.email(
}
```

```
package io.shwitter.emailer
interface EmailHolder  {
  String email();
}
```

TOOLS

Macker

What's New in Macker 0.4.x

New features

Version 0.4.2:

- Added an Ant task for report formatting, with built-in support for HTML/CSS output, and extensibility through XSLT. (more)

Version 0.4:

- Filters, a new construct, open the door to customizable pattern behavior. The first set of filters match based on supertypes and basic class attributes. The next version of Macker will include more sophisticated filters. (more)
- Rules now have severity levels -- you can generate warnings without failing the Ant build, or include debug messages which are hidden during a normal run. (more)
- Macker can now generate XML reports. (more)
- Message rules can print arbitrary messages, optionally with error or warning status. (more)
- The special from/to variables are now available to patterns in an access rule, in addition to the message. Two new variables give just the from/to package name. (more) Two new patterns match the from/to classes. (more)
- The Ant task can now set a property when rule checking fails, and supports limiting the number of errors / messages printed to the console. (more)

Changes to existing features

- The `regex` attribute is now deprecated in favor of `class`, which will make future features more readable. (Only the name has changed; both do the same thing.)
- For each now also matches against classes one reference out from the primary classes, instead of just the primary classes themselves. This lets you check against modules which are not part of the primary classes, but, for example, in a jar from a different project.
- Verbose output shows classes in sorted order.

Implementation changes

- Macker now uses Jakarta BCEL for class file parsing. This speeds things up a hair, and eliminates the warning messages JClasslib used to generate.
- Class analysis now is vastly more detailed: it includes superclass, interfaces, and access modifiers, and internally distinguishes different types of signature-level references. This richer analysis will manifest itself in richer patterns and rules in future versions of Macker. (Right now, it probably just cancels out the speed benefits of BCEL.)
- The internal event callback mechanism is better structured (inching closer to IDE integration).
- Macker now forces validation against its internal DTD.
- There's a small but steadily growing regression test suite.

MACKER EXAMPLE

```
<?xml version="1.0"?>
<macker>
    <ruleset name="Simple example">
        <access-rule>
            <deny>
                <from class="**Print*" />
                <to class="java.**" />
            </deny>
        </access-rule>
    </ruleset>
</macker>
```

4. Dependency Definition File

Classycle's Dependency Checker uses a dependency definition file (file type `.ddf`)
a syntax as explained below) which define sets of classes and independency relati

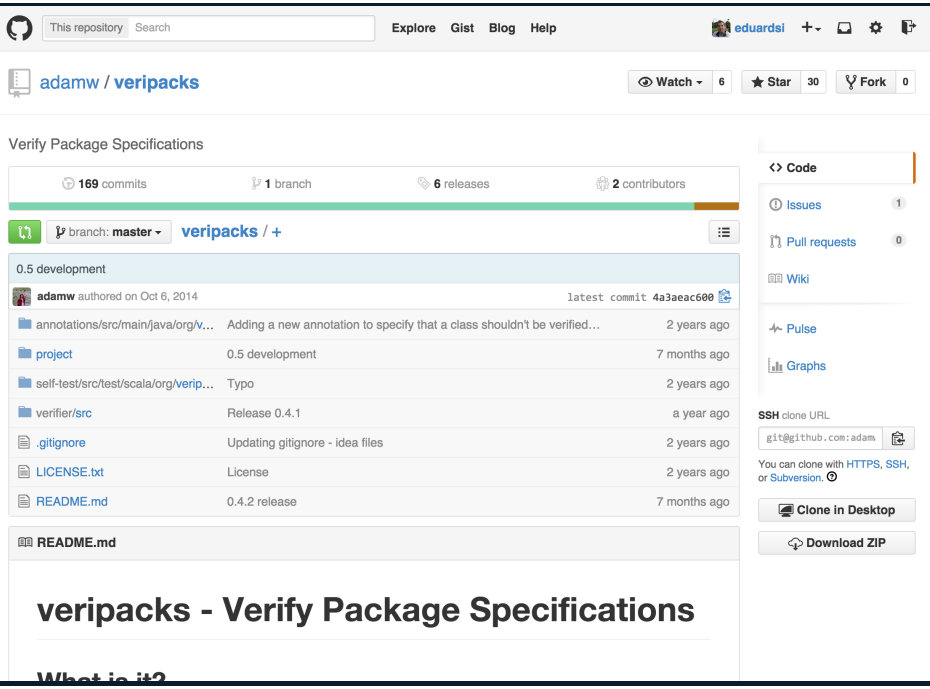
4.1 Example

The following example file contains all types of commands:

```
#
# This is an example of a dependency definition file
#
show allResults

{package} = classycle

[util] = ${package}.util.*
[non-util] = ${package}.* excluding [util]
[class-file] = ${package}.classfile.*

check sets [util] [non-util] [class-file]

check [util] independentOf [non-util]
check [class-file] independentOf [util]
```

CLASSYCLE EXAMPLE

```
#
# This is an example of a dependency definition file
#
show allResults

{package} = classycle

[util] = ${package}.util.*
[non-util] = ${package}.* excluding [util]
[class-file] = ${package}.classfile.*

check sets [util] [non-util] [class-file]

check [util] independentOf [non-util]
check [class-file] independentOf [util]
```


This repository Search
Explore Gist Blog Help
eduardsi
adamw / veripacks
Watch 6
Star 30
Fork 0
Verify Package Specifications
169 commits
1 branch
6 releases
2 contributors
branch: master
veripacks / +
0.5 development
adamw authored on Oct 6, 2014
latest commit 4a3aeac600
annotations/src/main/java/org/v... Adding a new annotation to specify that a class shouldn't be verified... 2 years ago
project 0.5 development 7 months ago
self-test/src/test/scala/org/verip... Typo 2 years ago
verifier/src Release 0.4.1 a year ago
.gitignore Updating gitignore - idea files 2 years ago
LICENSE.txt License 2 years ago
README.md 0.4.2 release 7 months ago
README.md
veripacks - Verify Package Specifications
What is it?
Code
Issues 1
Pull requests 0
Wiki
Pulse
Graphs
SSH clone URL
git@github.com:adamw
You can clone with HTTPS, SSH, or Subversion.
Clone in Desktop
Download ZIP

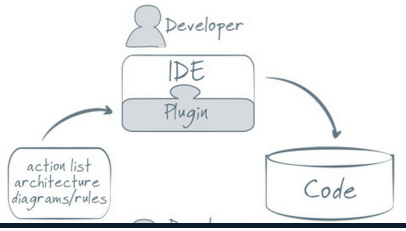
structure101

Products Resources Download Pricing About Blog

A superb weapon in our continuing battle against architecture entropy.

JOLT Productivity Award

Organize for success



Structure101 is an agile *architecture development environment (ADE)* that lets the software development team organize a codebase.

Building its model direct from the code, Structure101 empowers architects to work with the team to maintain architecture rules, diagrams and action lists that organize a codebase into a modular hierarchy with low and controlled coupling.

STAN
Structure Analysis for Java

GENERAL

- Home
- Download
- Purchase
- White Paper
- License Terms

STAN

- Introduction
- Dependency Analysis
- Quality Metrics
- Report Generation
- Eclipse Integration
- Advanced Topics

STAN 2.1.2

Wednesday, 26 March 2014

We are pleased to announce the 2.1.2 maintenance release of ***STAN***, adding support for Java 8.

STAN 2 comes with exciting new features and includes a **community license** option for free, non-commercial use.

New ***STAN 2*** features in a nutshell:

- **Sandbox View** - Explore dependencies between *any* artifacts via Drag'n Drop
- **Map View** - Visualize metric ratings in fancy green-to-red treemaps
- **IDE Tools** - Provides views to show Project and Plug-in dependency graphs

Check our updated white paper for a brief introduction to structure analysis with ***STAN 2***.
Get further information on product variants, license options, pricing, or buy now!

STAN 2.1 adds new export formats for dependency graphs (PDF, SVG, XML, TXT), a simplified *Couplings View*, new class metrics (Ca, Ce) and more. ***STAN 2.1*** requires Java 6 or later.

[Back]

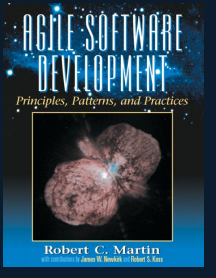
STAN 2.1.2

We are pleased to announce the 2.1.2 maintenance release of *STAN*, adding support for Java 8.

Read more...

MORE

- OO Design Principles & Metrics, Jason Gorman http://goo.gl/RTW9GT
- The Economics of Software Design, J.B. Rainsberger http://goo.gl/ra7080
- SOLID Principles, Eduards Sizovs http://goo.gl/Rpxavd
- Designing Object-Oriented Software, Jouni Smed http://goo.gl/iyE1R2
- Grand Unified Theory Of Software Design, Jim Weirich http://goo.gl/ASqyAs
- Fun With Modules, Kirk Knoernschild http://goo.gl/i8jx8Y
- Patterns of Modular Architecture http://goo.gl/vFqmZO
- Let's turn packages into a module system! http://goo.gl/Mzco8F



EITHER YOU WORK TO CREATE A
COMPANY CULTURE OR YOU DON'T.
EITHER WAY A CULTURE WILL EMERGE.

(C) MARTIN LINKHORST

EITHER YOU WORK TO CREATE A SOFTWARE STRUCTURE OR YOU DON'T. EITHER WAY A STRUCTURE WILL EMERGE.

(C) EDUARDS SIZOVS


THANK YOU