Hammurapi

code review platform

Pavel Vlasov, January 2005

Excurse to history

Hammurapi*, king of Babylon (1792-1750 BC)

Hammurapi (1792-1750 BC) was the sixth ruler of a line of Amorite kings, who had established themselves at the city of Babylon around 1900 BC.

Hammurapi is perhaps most celebrated for his so-called law-code. Although it was not intended to function like a modern law-code, its statement of traditional or contemporary practice in all areas of civil and criminal law was an assertion of Hammurapi's role as the champion of justice. One copy of the text, written in Akkadian cuneiform on a large stela, was carried off as booty by an Elamite army to the city of Susa in the thirteenth century BC, and is now in the Musée du Louvre, Paris. There are 282 laws.

English translation of the Hammurapi Code of Laws can be found here:

http://www.bible-history.com/babylonia/BabyloniaCode_of_Hammurapi.htm

Our Hammurapi is a code review tool – it scans Java source files and finds “smelly places”. We don’t have as many laws (they are called inspectors) as on the Hammurapi stela. Currently there are over 120 built-in inspectors shipped with Hammurapi. Custom inspectors are easy to write.

* Also spelled as Hammura*b*i.

Tools

Hammurapi contains the following tools:

•**Hammurapi** – performs automated reviews of Java source code. Can be executed from command line or as Ant task. Loads all source files in database-backed Jsel repository and generates detailed reports. Intended audience – architects who want to see the whole picture.

•**Quickurapi** (quick Hammurapi) – Sacrifices insight for the sake of speed. Reviews Java files one-by-one using in-memory Jsel repository. Generates simplified reports. Can be executed from command line or as Ant task. Intended audience – developers who care about compliance of their files.

•**Archiver** – packages source files and class files/jars in an archive file (.har) to be processed by Hammurapi or Quickurapi. This tool allows to separate review request and execution in space and time. Possible usage scenarios: a) Troubleshooting b) Archiver can be executed as part of build process (it is very fast) and put an archive into a queue directory or ftp it to another machine to be processed by Hammurapi.

•**Query tool** – Console application. Allows to query source repositories in a way similar to querying databases. Uses OGNL for model navigation. There are plans to switch to OCL. Syntax: select <OGNL expression> from <type> where <OGNL expression>.

•**Plugin framework** – Classes to embed Hammurapi into Java apps.

•**Eclipse plugin** – Hammurapi plugin for Eclipse

Benefits

Gartner says only 32% of the 2.5 million Java developers in the world have genuine knowledge, which means there is a serious lack of high-level development skills.

TCO reduction:

– Developers learn while they work

– Outsourcing and on-boarding is easier because corporate standard are enforced automatically

Information week: "Many companies are not satisfied with outsourcing". One of reasons – low code quality.

– Maintenance is easier because all code follows standards

– Safety net provided by Hammurapi allows to use advanced coding techniques, e.g. complex patterns.

Risk mitigation

- Automated detection of potential problems mitigates risk of runtime failures
- Technology stack inspector mitigates risk of rework caused by usage of improper library or version of library

Jump start

Review sample files:

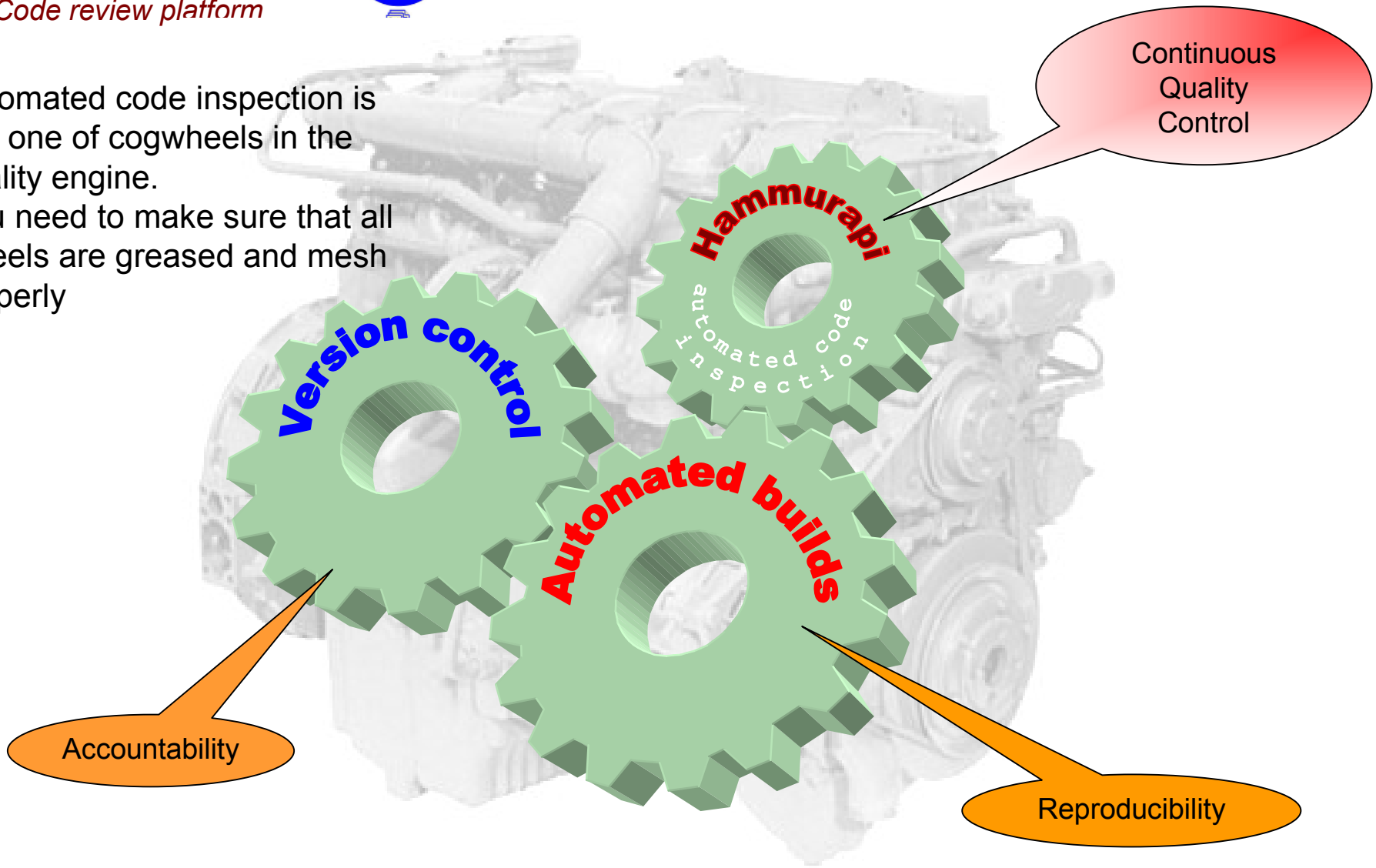
- Download Hammurapi from http://www.hammurapi.org
- Unzip
- Run `ant` in `projects/template` directory
- Open `project/template/review/report.html` in a Web browser

Review your files:

- Create a copy of `projects/template` directory
- Put your source files to `src` directory
- Put jar files to `lib` directory
- Run `ant clean`
- Run `ant`
- Browse report created in `review` directory

Quality engine

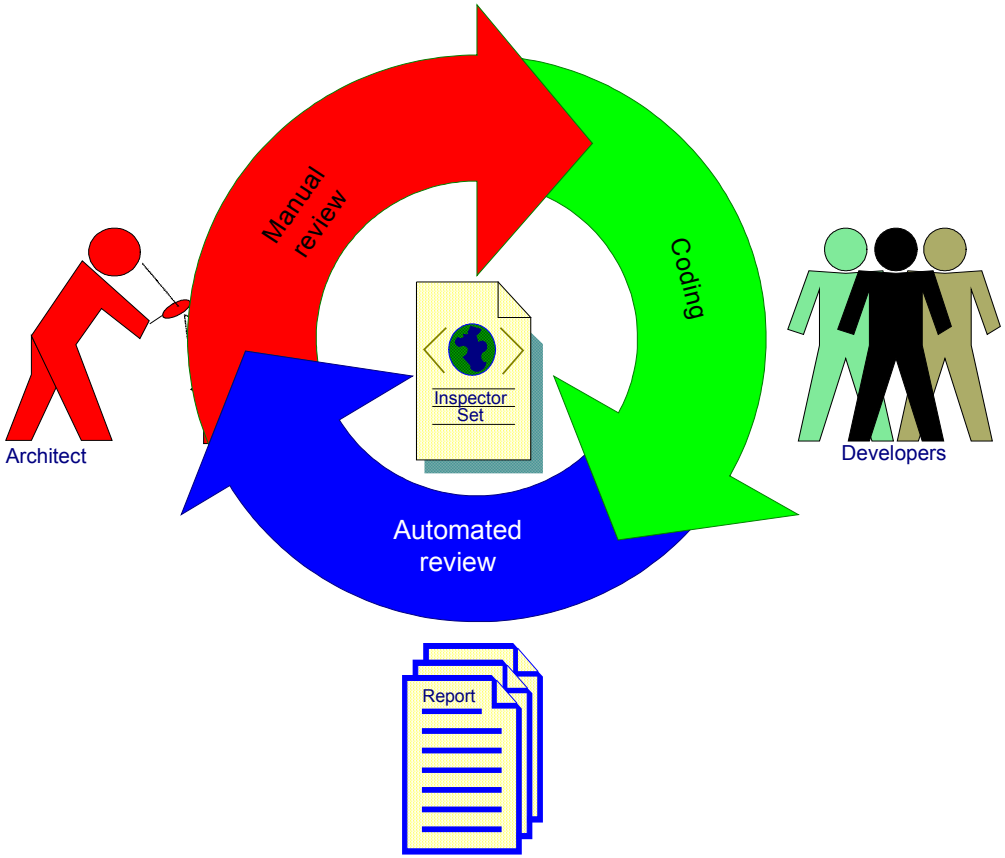
Automated code inspection is just one of cogwheels in the quality engine.
You need to make sure that all wheels are greased and mesh properly



Hammurapi

Code review platform

Enterprise-oriented



- Inspectors and waivers can be loaded from files and URLs, e.g. from a central web server.
- Hierarchical inspector sets and inspector configuration from multiple sources – you can define common rules and then fine tune them for technology, component or review type.
- Waivers – handle special cases.
- Listeners – integration.
- HTML Reports – publish in Intranet.
- Builds can be failed on low code quality.
- Omnipresence:
  - Developer desktop - Eclipse plugin or Quick task
  - Version control - CVS check-in handler (Quick task)
  - Build process – Ant task.

Knowledge management

Results of

- code reviews
- production troubleshooting
- end-of-project “lessons learned” sessions

can be captured as inspectors and disseminated to the whole organization.

Control

Hammurapi allows you to actively defend yourself from bad code by failing builds on:

- **Sigma threshold** – if Sigma is too low
- **DPMO threshold** – if DPMO is too high
- **Severity threshold** – if code contains violations more severe than you can tolerate.
- **Warnings** – if not all inspectors could perform their job (misconfiguration)

Time dimension

Review results are persisted and reviews have "*time dimension*":

- Files changed since last review are marked with "New" icon
- Baselined reviews show deltas between current review and base review
- History annotation shows codebase evolution in a table and charts.
- Comply-on-touch policy
- Only modified files are reviewed – faster incremental reviews while producing complete report.

Hammurapi

Code review platform

Changes highlighting

Files changed since last review

Package is highlighted if it contains changed files

Files

org.hammurapi NEW

- BaseInspector.java
- BaseParameterizableInspector.java NEW
- BaseSelfDescribingInspector.java
- BaseTask.java NEW
- DomInspectorDescriptor.java NEW
- DomInspectorSource.java
- DomWaiver.java
- DomWaiverSource.java
- EmbeddedInspectorSetDocumenter.java
- FilterEntry.java
- FilteringInspector.java
- HammurapiArchiver.java NEW
- HammurapiException.java
- HammurapiFileSet.java
- HammurapiMMTask.java NEW
- HammurapiNonConsumableException.java
- HammurapiRuntimeException.java
- HammurapiTask.java NEW

org.hammurapi NEW

Name	Reviews	Violations	DPMO	Sigma
Inspector.java	1411	28	517	4.781
BaseParameterizableInspector.java NEW	462	7	930	4.611
BaseSelfDescribingInspector.java	1082	29	850	4.638
BaseTask.java NEW	17567	365	1012	4.587
DomInspectorDescriptor.java NEW	7231	136	623	4.728
DomInspectorSource.java	1501	33	1059	4.573
DomWaiver.java	5180	74	768	4.668
DomWaiverSource.java	1475	30	1057	4.574
EmbeddedInspectorSetDocumenter.java	1404	40	1566	4.454
FilterEntry.java	252	5	1269	4.519
FilteringInspector.java	103	0	0	Full compliance
HammurapiArchiver.java NEW	8141	144	1028	4.582
HammurapiException.java	309	7	1974	4.382
HammurapiFileSet.java	349	6	945	4.607
HammurapiMMTask.java NEW	2270	45	1427	4.483
HammurapiNonConsumableException.java	297	8	2087	4.365
HammurapiRuntimeException.java	315	7	1936	4.388

Baselined reviews

Hammurapi

Results

Date	2004/11/28
Baseline date	2004/08/24
Packages	20
Files	306
Codebase	334000 (+80411)
Reviews	531228 (+127421)
Violations	11044 (+2469)
Waived violations	0
DPMO	1015 (-5)
Sigma	4.586 (+0.002)

Baseline date

Deltas

Severity summary

Severity 1, total 202

Inspector	Description	Number
ER-002	Empty catch block	5 (+2)
ER-011	Cyclomatic complexity exceeds specified limit	50 (+24)
ER-035	Switch statement case without 'break'	3 (+3)
ER-048	Use BigDecimal instead of Float or Double for monetary values	87 (+13)
ER-073	Call 'super.clone ()' in all 'clone ()' methods	1
ER-076	Make inner classes "private"	11 (+3)
ER-080	Avoid "for", "do", "while", "if" and "if ... else" statements with empty bodies	9 (-1)
ER-111	LOG4J is mandatory for all logging in ERC. System.out.println will be lost in a system console of our web/app server.	36 (+3)

Hammurapi

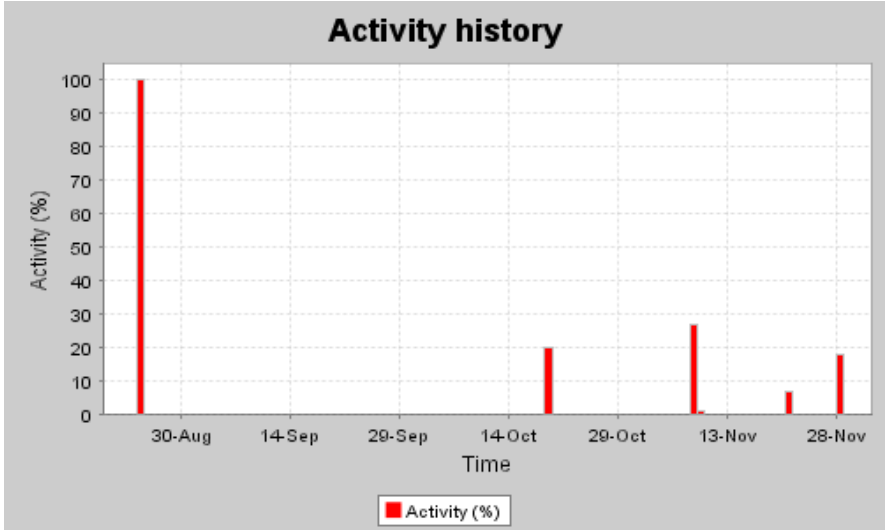
Code review platform

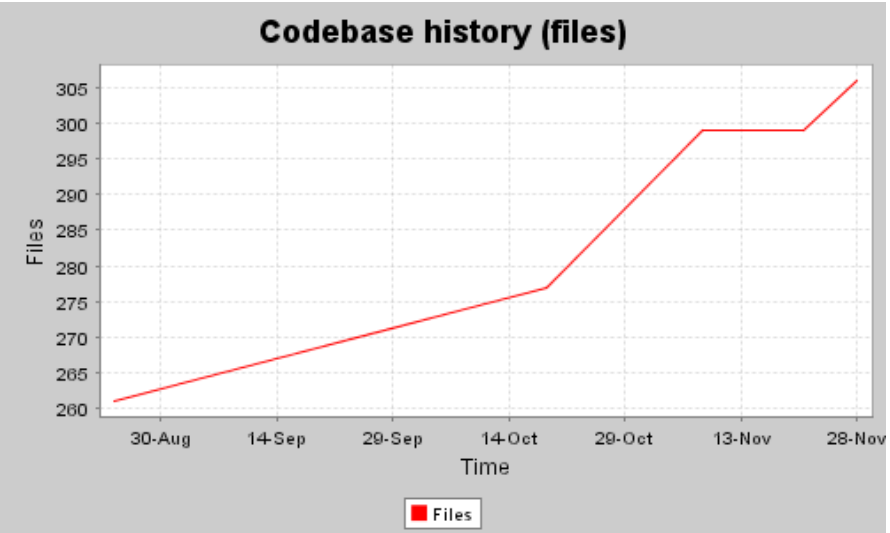
History annotation

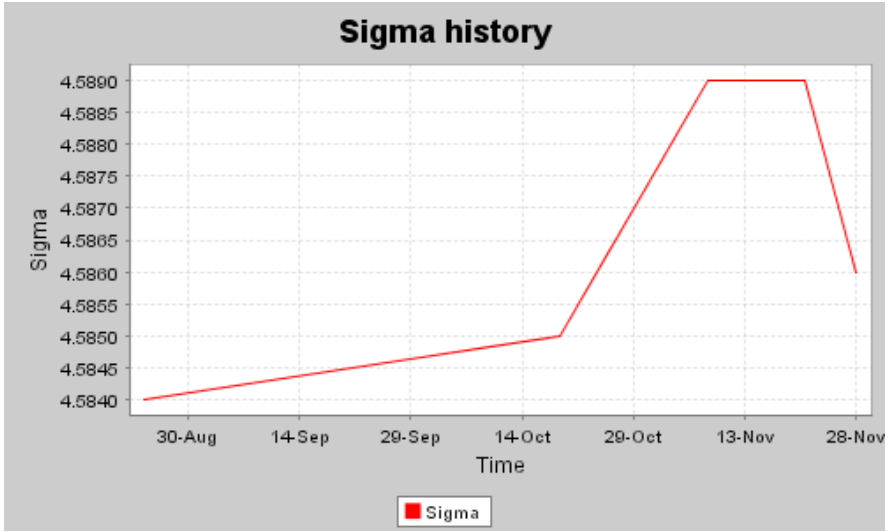
Hammurapi

history of reviews

Date	Description	Reports	Codebase		Reviews	Activity (%)	Violations	Max severity	Sigma	DPMO
			Files	Nodes						
2004-08-24	Release 3.2.0	1	261	253589	403807	100	8575	1	4.584	1020
2004-10-19	Release 3.3.0	1	277	280392	446054	20	9362	1	4.585	1017
2004-11-08	Release 3.4.0	1	299	313013	497744	27	10281	1	4.589	1002
2004-11-09	Release 3.4.1	1	299	313013	497744	1	10281	1	4.589	1002
2004-11-21	Release 3.5.0	1	299	315883	502152	7	10373	1	4.589	1002
2004-11-28	Release 3.6.0	1	306	334000	531228	19	11044	1	4.586	1015







Comply-on-touch

In incremental reviews setup

Inspectors do not have retroactive force

unless “force” attribute is set to “true”

Hammurapi reviews only files modified since last review.

If new inspectors are added to inspector set between reviews then they will report violations only in “touched” files – modified or newly added.

This allows to implement gradual code quality improvement on large code bases – inspectors are added in small groups and developers shall bring to compliance only files they currently work on.

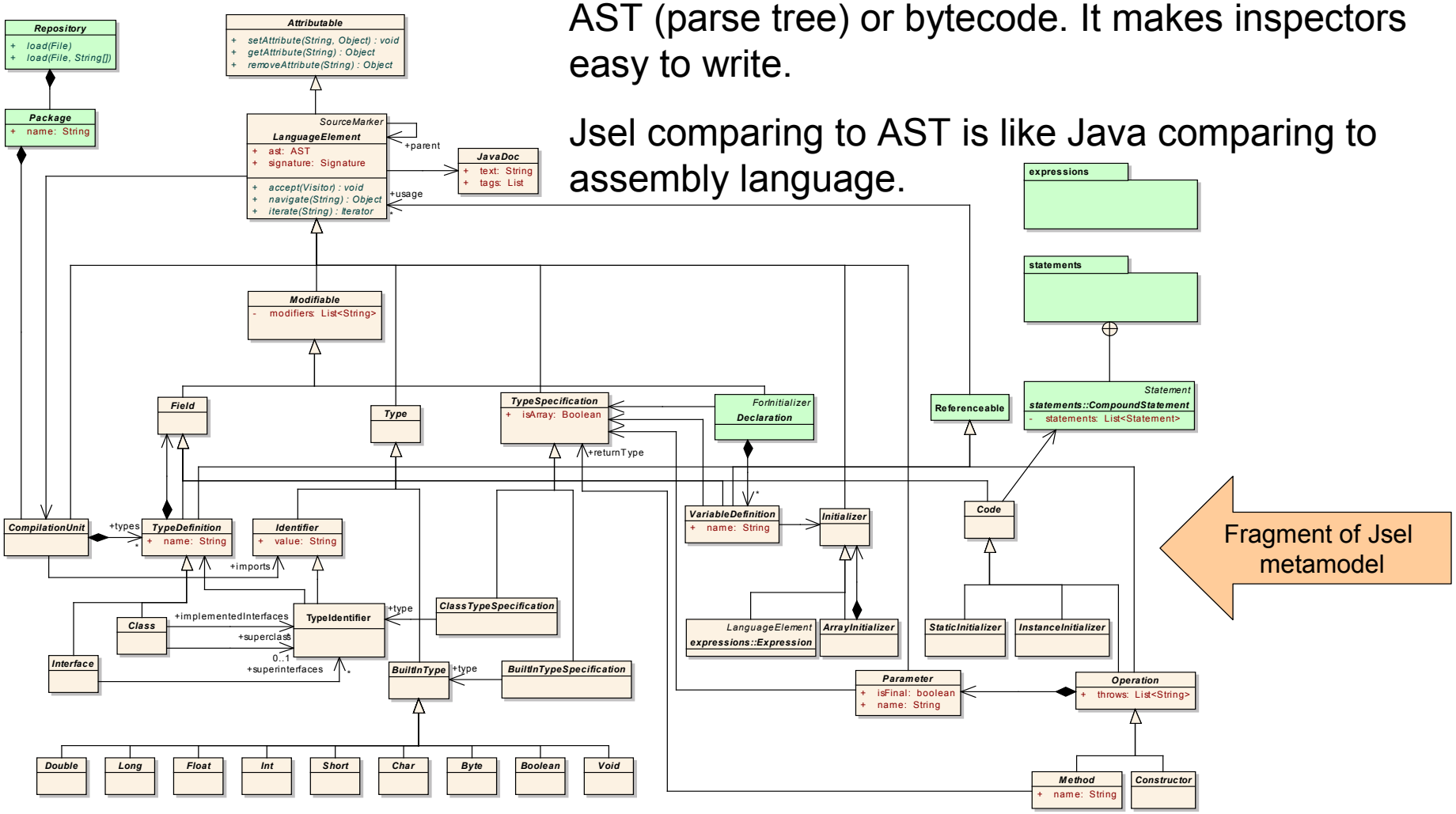
Hammurapi

Code review platform

Jsel metamodel

Hammurapi inspectors work on Java metamodel in contrast with most code review tools which work on AST (parse tree) or bytecode. It makes inspectors easy to write.

Jsel comparing to AST is like Java comparing to assembly language.

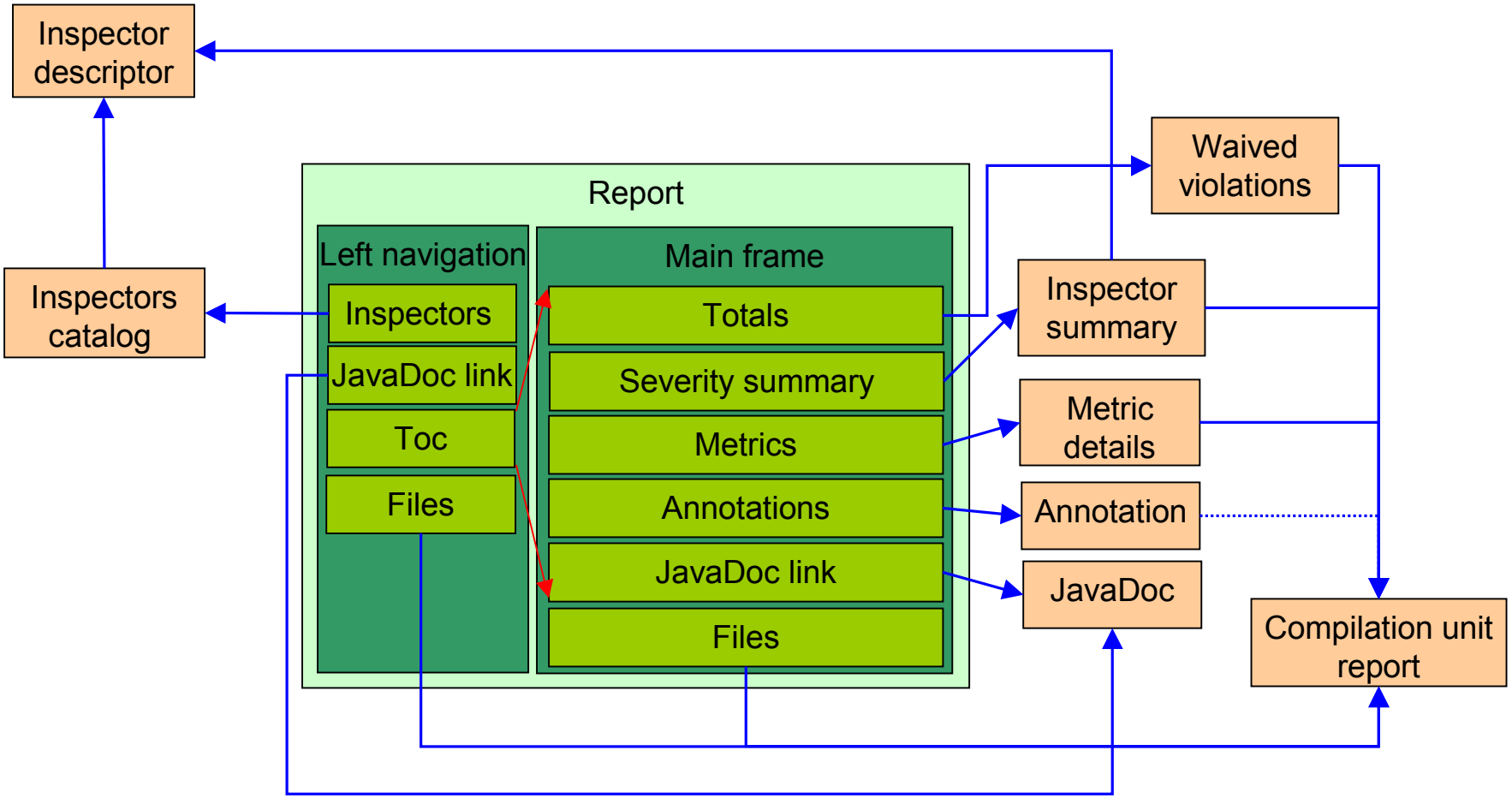


Visitor pattern

- Hammurapi uses modified Visitor pattern to navigate inspectors through source code.
- Navigation is performed through Jsel model, not through AST.
- Inspectors don't need to implement Visitor interface – Dispatching visitor invokes visit() and leave() methods based on type compatibility
- Javadoc comments are parsed and visited by inspectors
- Individual tokens are visited as well

Report structure

Everything is within 2 mouse clicks



Hammurapi

Code review platform

Report

Hammurapi 3.6.0

Navigation frame

Inspectors

JavaDoc

Summary
- Severity summary
- Metrics
- Annotations
  - Technology stack
  - NCSS
  - JSP-Metrics
  - History
  - CallerTrace

Files

org.hammurapi NEW	3340
BaseInspector.java	28
BaseParameterizableInspector.java NEW	7
BaseSelfDescribingInspector.java	29
BaseTask.java NEW	365
DomInspectorDescriptor.java NEW	136
DomInspectorSource.java	33
DomWaiver.java	74
DomWaiverSource.java	30
EmbeddedInspectorSetDocumenter.java	40
FilterEntry.java	5
FilteringInspector.java	
HammurapiArchiver.java NEW	144
HammurapiException.java	7
HammurapiFileSet.java	6
HammurapiMMTask.java NEW	45
HammurapiNonConsumableException.java	8
HammurapiRuntimeException.java	7
HammurapiTask.java NEW	237
Inspector.java	1
InspectorContext.java	2
InspectorContextBase.java	72

Main frame

Hammurapi

Results

Date	2004/11/28
Baseline date	2004/08/24
Packages	20
Files	306
Codebase	334000 (+80411)
Reviews	531228 (+127421)
Violations	11044 (+2469)
Waived violations	0
DPMO	1015 (-5)
Sigma	4.586 (+0.002)

Severity summary

Severity 1, total 202

Inspector	Description	Number
ER-002	Empty catch block	5 (+2)
ER-011	Cyclomatic complexity exceeds specified limit	50 (+24)
ER-035	Switch statement case without 'break'	3 (+3)
ER-048	Use BigDecimal instead of Float or Double for monetary values	87 (+13)
ER-073	Call 'super.clone ()' in all 'clone ()' methods	1
ER-076	Make inner classes "private"	11 (+3)
ER-080	Avoid "for", "do", "while", "if" and "if ... else" statements with empty bodies	9 (-1)
ER-111	LOG4J is mandatory for all logging in ERC. System.out.println will be lost in a system console of our web/app server.	36 (+3)

Severity 2, total 2542

Inspector	Description	Number
ER-004	A class should have no public fields except 'final' or 'static final'"	51
ER-008	Synchronize at the block level rather than the method level	1
ER-009	For statement requires update clause	13 (+4)
ER-010	Assignment inside conditional	3
ER-017	For comprehensibility, formal parameters should be final	1403 (+368)
ER-033	Source file is too long	1 (-1)
ER-037	Logical nesting limit exceeded	3

Navigation frame

Hammurapi 3.6.0

Inspectors

JavaDoc

Summary
- Severity summary
- Metrics
- Annotations
  - Technology stack
  - NCSS
  - JSP-Metrics
  - History
  - CallerTrace

Files

org.hammurapi NEW	3340
BaseInspector.java	28
BaseParameterizableInspector.java NEW	7
BaseSelfDescribingInspector.java	29
BaseTask.java NEW	365
DomInspectorDescriptor.java NEW	136
DomInspectorSource.java	33
DomWaiver.java	74
DomWaiverSource.java	30
EmbeddedInspectorSetDocumenter.java	40
FilterEntry.java	5

Link to inspectors catalog

Link to JavaDoc

Indicates that package contains modified files

Modified since last review

Total number violations in package

Bold font indicates that file contains violations

Number of Violations in file

Hammurapi

Code review platform

Inspectors catalog

Severity

Inspector category

Link to descriptor

Coding Guidelines

1. ER-054 (1) Avoid calling an "abstract" method from a constructor in an "abstract" class
2. ER-033 (2) Source file is too long
3. ER-104 (2) Use a Collection instead of arrays Object[]
4. ER-031 (3) More than one statement per line
5. ER-032 (3) Array declarators should be placed next to the type, not the variable name
6. ER-034 (3) Variables should be declared in individual statements.
7. ER-036 (3) Line is too long
8. ER-096 (3) Empty statements
9. ER-098 (3) No need to provide (public, abstract,) modifiers for interface methods
10. ER-107 (3) Instance variables and method names shouldn't have same name
11. ER-200 (3) Instance variables and the declaring type shouldn't have same name
12. ER-201 (3) Discourage usage of instance variables like a, j by enforcing minimal variable name length.

Database Connection Pool

13. ER-207 (1) SQL Resource Management - Create Statement Without Close Rule: You have to close each created SQL Statement on method level. Use the finally block, but check for null value. If you use a operation in the finally block for closing your SQL resource, please define the operation name in the inspector.xml. Hammurapi will search for this method call and check the parameter list. This rule is only applicable in a connection-pooled environment.
14. ER-208 (1) SQL Resource Management - Create Statement Within a Loop: Never create Statements inside loops. This rule is only applicable in a connection-pooled environment.
15. ER-206 (2) Wrong declaration of SQL Resources Management: Do not declare Statements and ResultSets on Instance Level but use local variables on method level only. You may run in leakage problems if you do not close these resources in a connection-pooled environment.

EJB

16. ER-055 (1) Declare bean classes "public", but not "final"
17. ER-056 (1) Declare 'ejbCreate ()' methods "public", but neither "static" nor "final"
18. ER-057 (1) Declare finder methods "public" and neither "final" nor "static"
19. ER-058 (1) Implement one or more 'ejbCreate ()' methods in bean classes
20. ER-059 (1) Implement matching 'ejbPostCreate ()' methods for every 'ejbCreate()' in EntityBean classes
21. ER-061 (1) Do not define 'finalize ()' method in bean classes
22. ER-063 (1) Declare 'ejbPostCreate ()' "public" and neither "static" nor "final"
23. ER-064 (1) Make the return type "void" for SessionBeans' 'ejbCreate ()' methods
24. ER-065 (1) Make the return type "void" for the 'ejbPostCreate ()' method
25. ER-066 (1) Avoid passing the "this" reference as an argument
26. ER-067 (1) Avoid returning "this"
27. ER-068 (1) Avoid starting, stopping, or managing threads in any way
28. ER-060 (2) Avoid loading native libraries in a Bean class
29. ER-062 (2) Declare all "static" fields in the EJB component "final"

Inspector descriptor

Inspector descriptor is an educational resource on developer's fingertips

ER-204 Allocation of the resource should follow try/finally pattern to ensure proper de-allocation.

Severity	2
Enabled	yes
Waivable	
Rationale	Allocating resource and not properly disposing it is a common problem in applications (JDBC/JNDI/Sockets). Usually this happens due to not using try/finally pattern.
Violation	
Fix	

Violation

```
void problemCode1(DataSource ds) {
  Connection conn = ds.getConnection();
  PreparedStatement stmt = conn.prepareStatement("SELECT * FROM MY_TABLE");
  //...
  conn.close();
}

void problemCode2(DataSource ds) {
  Connection conn = null;
  try {
    Connection conn = ds.getConnection();
    PreparedStatement stmt = conn.prepareStatement("SELECT * FROM MY_TABLE");
    //...
  } finally {
    conn.close();
  }
}

void problemCode3(DataSource ds) {
  Connection conn = ds.getConnection();;
  try {
    //...
  } finally {

    conn.getAutoCommit(); // can throw an exeption and next line never execute
    conn.close();

  }
}
```

Fix

```
Connection conn = ds.getConnection();
try {
  PreparedStatement stmt = conn.prepareStatement("SELECT * FROM MY_TABLE");
        //...
} finally {
  conn.close();
}
```

Summary

Hammurapi

Code review platform

Hammurapi

Results

Date	2004/11/28
Baseline date	2004/08/24
Packages	20
Files	306
Codebase	334000 (+80411)
Reviews	531228 (+127421)
Violations	11044 (+2469)
Waived violations	0
DPMO	1015 (-5)
Sigma	4.586 (+0.002)

Totals with deltas

Link to list of metric sources

Severity summary

Link to descriptor

Link to list of sources

Severity 1, total 202

Inspector	Description	Number
ER-002	Empty catch block	5 (+2)
ER-011	Cyclomatic complexity exceeds specified limit	50 (+24)
ER-035	Switch statement case without 'break'	3 (+3)
ER-048	Use BigDecimal instead of Float or Double for monetary values	87 (+13)
ER-073	Call 'super.clone ()' in all 'clone ()' methods	1
ER-076	Make inner classes "private"	11 (+3)
ER-080	Avoid "for", "do", "while", "if" and "if ... else" statements with empty bodies	9 (-1)
ER-111	LOG4J is mandatory for all logging in ERC. System.out.println will be lost in a system console of our web/app server.	36 (+3)

Severity 2, total 2542

Inspector	Description	Number
ER-004	A class should have no public fields except 'final' or 'static final'"	51
ER-008	Synchronize at the block level rather than the method level	1
ER-009	For statement requires update clause	13 (+4)
ER-010	Assignment inside conditional	3
ER-017	For comprehensibility, formal parameters should be final	1403 (+368)
ER-033	Source file is too long	1 (-1)

Metrics

Name	Number	Min	Avg	Max	Total
Change ratio	306	0.00	0.18	1.00	58.00
Class complexity	307	0.00	13.50	129.00	4147.00
Code length	2078	0.00	8.26	262.00	17168.00
File length	306	20.00	129.43	1044.00	39608.00
NCSS Class Inspector	341	0.00	78.47	956.00	26761.00
NCSS Method Inspector	1908	0.00	8.36	262.00	15962.00
NCSS Method p Class Inspector	1596	1.00	7.24	42.00	11562.00
Operation complexity	2068	0.00	2.19	38.00	4544.00

Annotations

- Technology stack
- NCSS
- JSP-Metrics
- History
- CallerTrace

JavaDoc

Files

Color coding of maximum severity in the file:
1 – Pink,
2 – Yellow,
3 – Green,
Other - White

org.hammurapi NEW

Name	Reviews	Violations	DPMO	Sigma
BaseInspector.java	1411	28	517	4.781
BaseParameterizableInspector.java NEW	462	7	930	4.611
BaseSelfDescribingInspector.java	1082	29	850	4.638
BaseTask.java NEW	17567	365	1012	4.587
DomInspectorDescriptor.java NEW	7231	136	623	4.728
DomInspectorSource.java	1501	33	1059	4.573
DomWaiver.java	5180	74	768	4.668
DomWaiverSource.java	1475	30	1057	4.574
EmbeddedInspectorSetDocumenter.java	1404	40	1566	4.454
FilterEntry.java	252	5	1269	4.519
FilteringInspector.java	103	0	0	Full compliance

Inspector summary

Inspector ER-002 summary

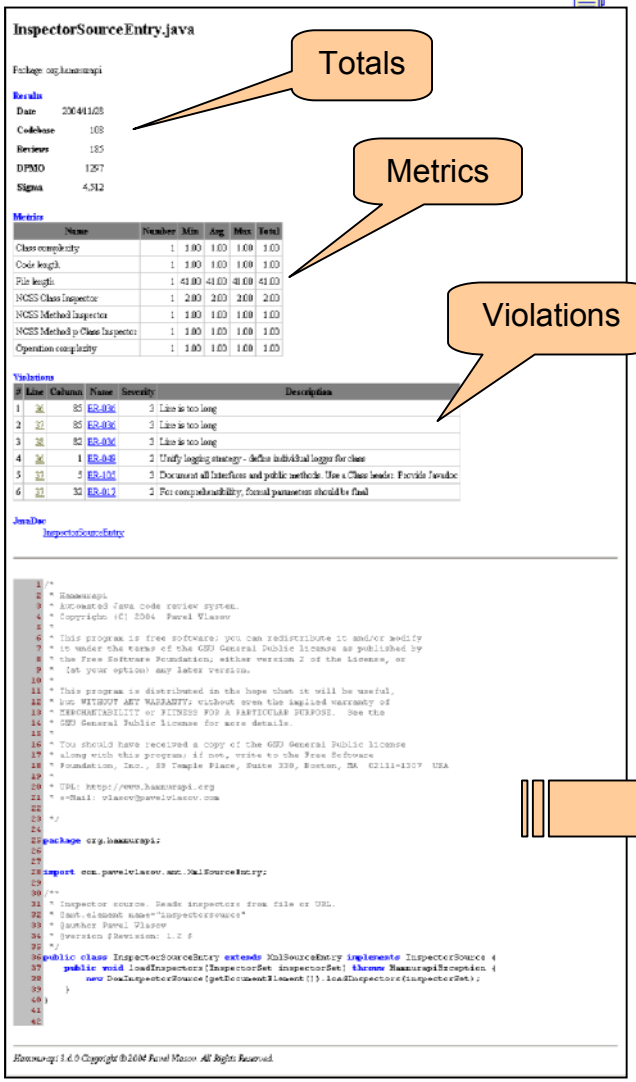
Severity: 1
Version: 2,1,5
Description: Empty catch block

Violations

#	File	Line	Column
1	org\hammurapi\eclipse\plugin\HammurapiBuilder.java	168	19
2	org\hammurapi\eclipse\plugin\HammurapiBuilder.java	180	19
3	org\hammurapi\inspectors\LogExceptionsRule.java	85	19
4	org\hammurapi\inspectors\UnusedVariablesRule.java	81	27
5	org\hammurapi\inspectors\metrics\ArchitecturalLayerInspector.java	571	35

```
571                         } catch (RuntimeException e) {
572                                 // TODO Auto-generated catch block
573                         }
574                 }
575         }
576 }
```

Compilation Unit




```java
/*
 * Hammurapi
 * Automated Java code review system.
 * Copyright (C) 2004  Pavel Vlasov
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * URL: http://www.hammurapi.org
 * e-Mail: vlasov@pavelvlasov.com

 */

package org.hammurapi;


import com.pavelvlasov.ant.XmlSourceEntry;

/**
 * Inspector source. Reads inspectors from file or URL.
 * @ant.element name="inspectorsource"
 * @author Pavel Vlasov
 * @version $Revision: 1.2 $
 */
public class InspectorSourceEntry extends XmlSourceEntry implements InspectorSource {
    public void loadInspectors(InspectorSet inspectorSet) throws HammurapiException {
        new DomInspectorSource(getDocumentElement()).loadInspectors(inspectorSet);
    }
}
```

Metric details

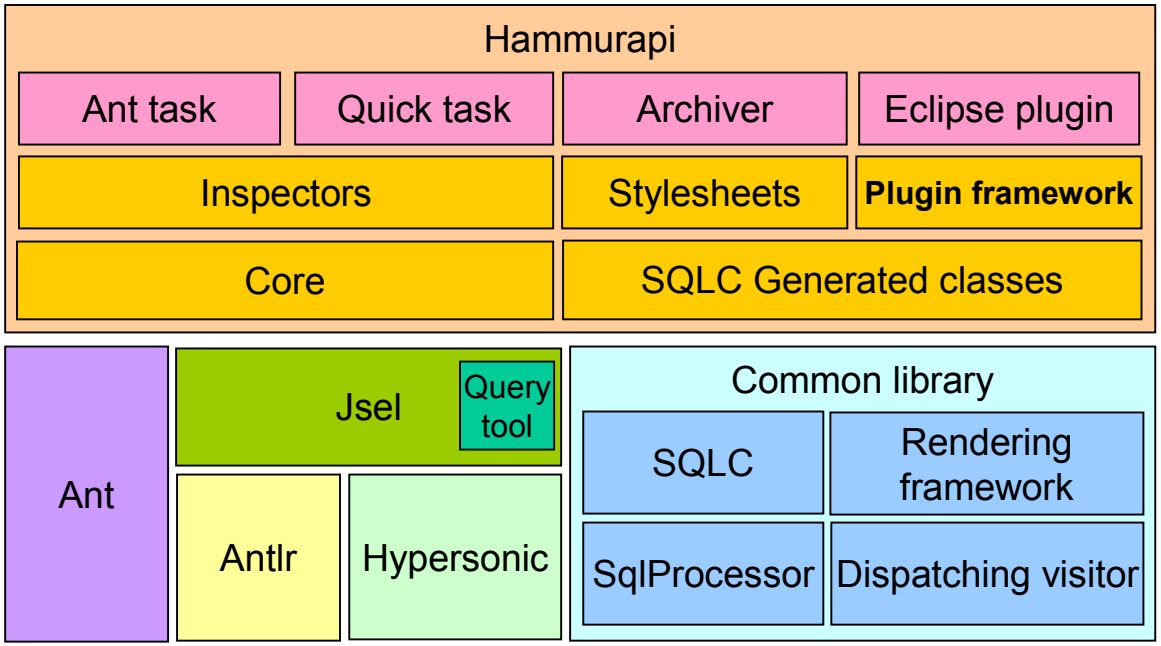
Ordered by value

"File length" metric details

Min: 20.0
Avg: 129.43790849673204
Max: 1044.0
Total: 39608.0
Samples: 306
Measurements

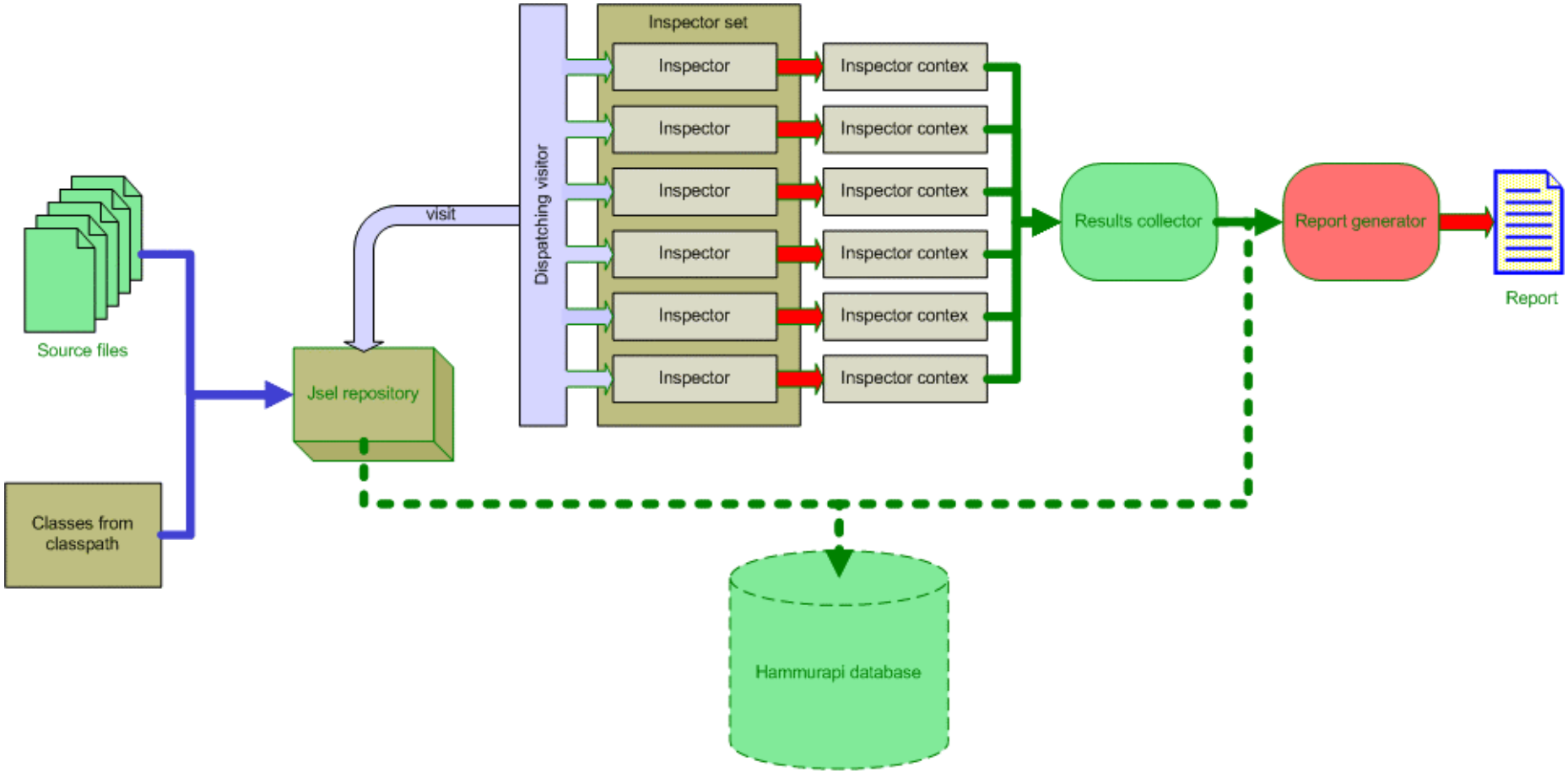
#	Value	Source
1	20.0	org\hammurapi\inspectors\metrics\callertrace\Node.java 7:1
2	26.0	org\hammurapi\inspectors\metrics\ArchitecturalLayerMappingTable.java 7:1
3	29.0	org\hammurapi\inspectors\metrics\callertrace\SearchMethod.java 7:1
4	29.0	org\hammurapi\inspectors\SystemGcMisuseRule.java 7:1
5	33.0	org\hammurapi\results\NamedResults.java 24:1
6	34.0	org\hammurapi\inspectors\SqlStatementAsInstanceVariableRule.java 5:1
7	34.0	org\hammurapi\WaiverEntry.java 25:1
8	34.0	org\hammurapi\results\InlineAnnotation.java 24:1
9	35.0	org\hammurapi\InspectorContextFactory.java 24:1
10	35.0	org\hammurapi\FilteringInspector.java 24:1
11	36.0	org\hammurapi\results\ReviewResults.java 24:1

Architecture



Click on a box to navigate to product home page

Architecture (continued)



Sample inspector

Requires to use `a.equals(b)` instead of `a.compareTo(b)==0`

```
public class UseEqualsInsteadOfCompareToInspector extends BaseInspector {

    public void visit(MethodCall methodCall) throws JselException {
        PrimaryExpression name = methodCall.getName();
        if (name instanceof Dot) {
            name=(PrimaryExpression) name.getOperands().get(1);
        }

        if ("compareTo".equals(name.toString()) && methodCall.getParameters().size()==1) {
            LanguageElement parent = ((LanguageElement) methodCall).getParent();
            if (parent instanceof Equal) {
                Object theOtherOperand=((Expression) parent).getOperands().get(0);
                if (theOtherOperand==methodCall) {
                    theOtherOperand=((Expression) parent).getOperands().get(1);
                }

                if (theOtherOperand instanceof IntegerConstant
                        && ((IntegerConstant) theOtherOperand).getValue()==0) {
                    context.reportViolation(parent);
                }
            }
        }
    }
}
```

Source

Descriptor

```
<inspector-descriptor>
    <name>ER-114</name>
    <category>Coding standards</category>
    <enabled>yes</enabled>
    <severity>3</severity>
    <inspector type="org.hammurapi.inspectors.UseEqualsInsteadOfCompareToInspector"/>
    <description>Use object.equals(anotherObject) instead of
    object.compareTo(anotherObject)==0</description>
    <rationale>equals() is part of the java.lang.Object contract whereas compareTo()
        is part of the java.lang.Comparable contract. Use more generic methods.
        Not every class implements compareTo(), but each
        class has equals() method.
    </rationale>
</inspector-descriptor>
```

Inspector services

Inspectors are provided the following services:

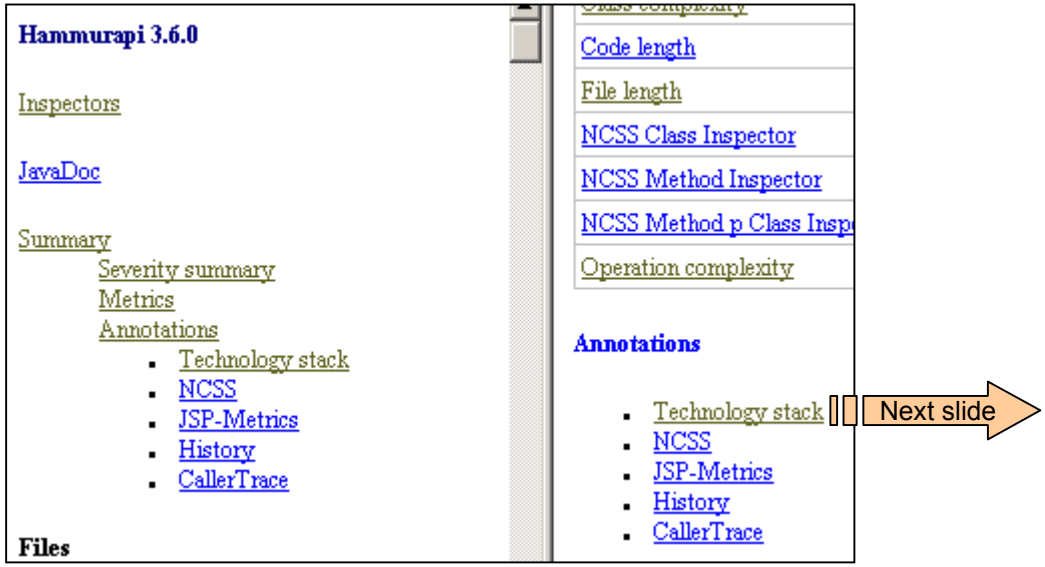
- Configuration injection through setters.
- Inspector Context allows inspectors to
  - Add metrics
  - Add annotations
  - Report violations
  - Issue a waiver for another inspecor
  - Output warning
  - Perform logging at `debug`, `verbose` and `info` levels.
- Session provides:
  - Inspectors interaction
  - Persistency

Annotations

Annotation is a means to customize Hammurapi report by adding arbitrary information.

Two types of annotations :

- Linked annotations
- Inline annotations



As names suggest the first is rendered as a link and the second is inlined into summary page.

Tech stack inspector

Hammurapi

Code review platform

Example of linked annotatoin

Technology stack

Summary Products Licenses

Publishers and Products

Apache foundation

Clients: 36

Name	Description	Catego
Commons	This product was inferred by the technology stack inspector from the package name. You should explicitly add proper information into the technology stack inspector configuration file	
Log4j	This product was inferred by the technology stack inspector from the package name. You should explicitly add proper information into the technology stack inspector configuration file	
Oro	This product was inferred by the technology stack inspector from the package name. You should explicitly add proper information into the technology stack inspector configuration file	
Tools	This product was inferred by the technology stack inspector from the package name. You should explicitly add proper information into the technology stack inspector configuration file	
Xpath	This product was inferred by the technology stack inspector from the package name. You should explicitly add proper information into the technology stack inspector configuration file	

Pavel Vlasov

Clients: 233

Name	Description	Category	License	Clients
Jsel	Parses Java source files and represents them as a graph of Java objects (repository)	Parsers	GPL	174
Hammurapi	Code review tool	Code review	GPL	9
Common library	Collection of utility classes	Misc. library	LGPL	176

Eclipse contributors

Clients: 4

Name	Description	Category	License	Clients
Eclipse platform	Java IDE	IDE	CPL	4

Technology stack

Summary Products Licenses

Licenses

Name	Description	Category	Products	Clients
Apache license			- Commons - Log4j - Oro - Tools - Xpath	36
CPL	Common Public License		- Eclipse platform - JUnit	9
GPL	GNU General Public License		- Jsel - Hammurapi	178
LGPL	GNU Lesser General Public License		- Common library	176
Public domain	Public domain software			
Unknown open source license	This license was inferred stack inspector from the should explicitly add pro technology stack inspect			

Technology stack

Summary Products Licenses

Clients of product *Hammurapi*

Client
org/hammurapi/QuickHammurapiTask.java
org/hammurapi/QuickPackageResults.java
org/hammurapi/QuickResultsCollector.java
org/hammurapi/QuickReviewEngine.java
org/hammurapi/QuickSummary.java
org/hammurapi/inspectors/history/HistoryInspector.java
org/hammurapi/results/persistent/jdbc/AggregatedResults.java
org/hammurapi/results/persistent/jdbc/BasicResults.java
org/hammurapi/results/persistent/jdbc/ResultsFactory.java

Eclipse plugin

Under development

Waivers

Waiver is the way to tell Hammurapi that some finding is not actually a violation. For example sometimes you need to have empty catch block as in the example below:

```
int integer=defaultValue;
if (string!=null) {
try {
          integer=Integer.parseInt(string);
} catch (NumberFormatException e) {
          // do nothing - use default value
}
```

Waivers are defined in XML file. Typical scenario is that the Architect or Senior developer manually reviews Hammurapi findings and decides what to do – fix or give a waiver.

Waivers are bound not to line and column but to parse path. They survive most source modifications

```
<waiver>
    <inspector-name>ER-002</inspector-name>

<signature>org/hammurapi/inspectors/testcases/violations/EmptyCatchBlocKRuleViolationTestCase.java:at[EmptyCatchBlockRuleViolationTestCase]:ao[getFirstByte(java.lang.String)]:eo[1]:es[java.io.IOException]</signature>
     <reason>This exception is ignored for testing purposes.</reason>
     <expiration-date>2004/05/15</expiration-date>
</waiver>
```

Waived violations will appear in the report as shown below:

Waived violations

#	Line	Column	Name	Severity	Description	Waiver reason	Waiver expires
1	57	19	ER-002	1	Empty catch block	To demonstrate waivers in action	2004/05/18

Waivers may have expiration date

•Waivers can be given per location (as described above) or per class/interface, compilation unit or package.

Inspectors interaction

- **Auto-waivers** – inspector can waive findings of another.
- **Filters** – inspector act as an approver on visit() methods for another inspector or multiple inspectors.
- **Accessing other inspector context** – inspector can report violations on behalf of another.
- **Ordering** – It is possible to ensure that visit() method of one inspector is always executed before visit() method of another.
- **Attributes of context and session** – Inspector context and Session are attributable. It allows inspectors share information.
- **Attributes of Jsel elements** – same as above.
- **Database** – Inspectors have access to a database to store information and access information stored by other inspectors or at previous reviews.

Auto waivers

Autowaivers allow one inspector waive finding of another.

Example: if code complies with "**ER-049** *Unify logging strategy - define individual logger for class*" then it violates "**ER-075** *Avoid hiding inherited instance fields*".*

To avoid this situation ER-049 automatically waives ER-075 by calling `context.waive(element, "AvoidHidingInheritedInstanceFields")`. The first parameter is the element for which waiver is given. The second parameter is a logical name of the inspector which finding is being waived.

```
<waives key="AvoidHidingInheritedInstanceFields">
    <name>ER-075</name>
    <reason>Logger is intended to hide superclass logger</reason>
</waives>
```

ER-049 Descriptor contains <waives> element

ER-075 is defined as *waivable*

```
<inspector-descriptor>
    <name>ER-075</name>
    <enabled>yes</enabled>
    <severity>1</severity>
    <inspector type="org.hammurapi.
    <description>Avoid hiding inher
<category>Object Oriented Programmi
    <!--
    <rationale></rationale>
    <resources></resources>
    -->
    <violation-sample><![CDATA[priv
    <fix-sample><![CDATA[private In
    <waivable>yes</waivable>
    <waive-case>Can be autowaived b
</inspector-descriptor>
```

* ER-049 was enhanced and this situation is not the case anymore. It is shown here as a good example of autowaiving

Filters

Filtering allows one inspector filter another, which means stop visit() method of inspector being filtered from being invoked.

This concept is similar to autowaiving, but autowaiving is more precise and elaborate mechanism.

Autowaived violations appear in the report in “Waived violations” section.

Filtering prevents inspector from visiting the node being filtered and thus no violation is ever reported.

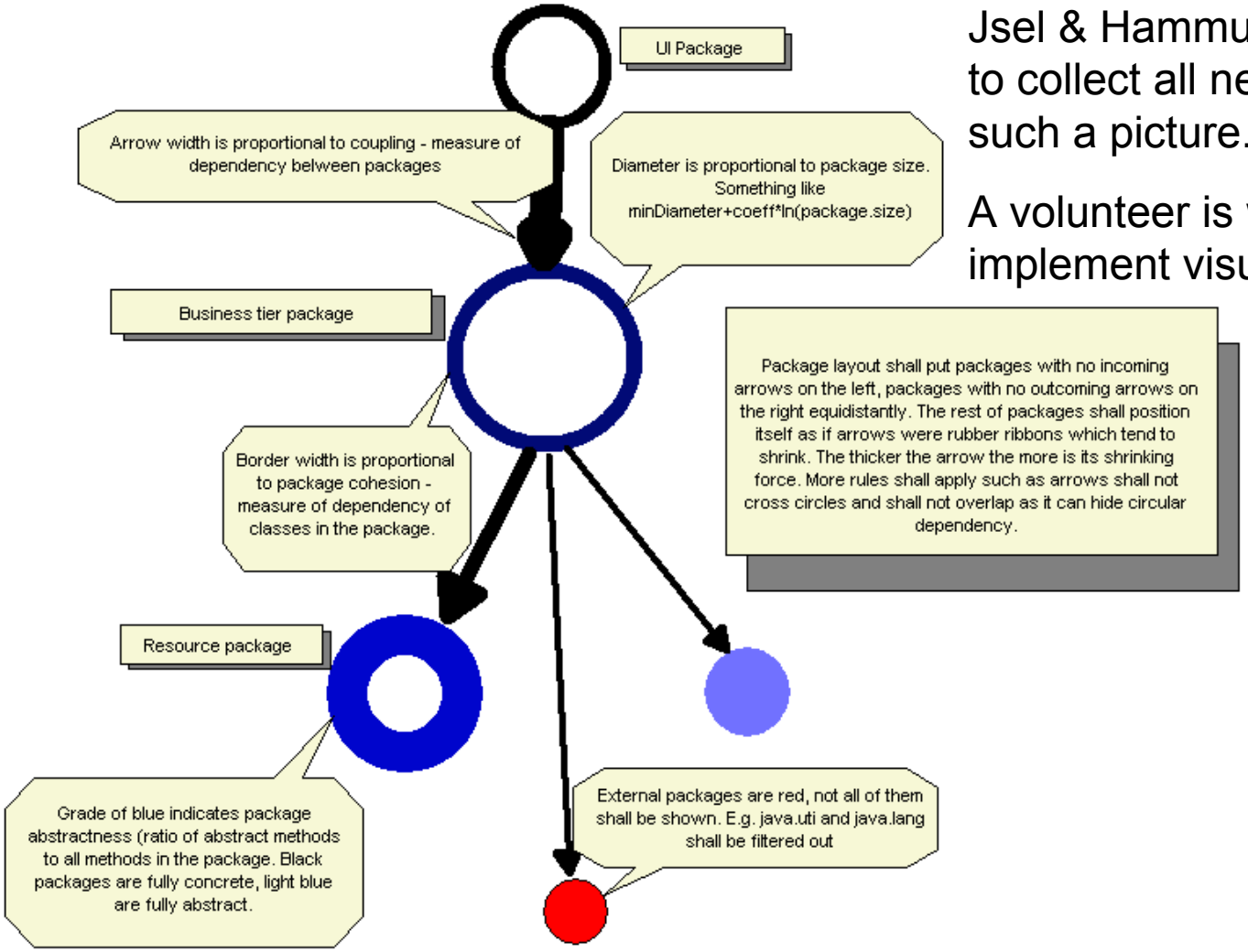
Filter – Inspector is a many-to-many relationship. One filter can filter multiple inspectors and one inspector can be filtered by multiple filters.

Filter may be associated with inspector by name or by category.

Next steps

- “Control center” web application to host and manage inspector sets, projects, waivers, …
- Java 1.5 support
- Review of other content types – Jsel in architecture slide can be easily replaced with another Visitable. Generic support to review any Antlr-generated AST is already in place.
- Multi-module reviews – combine reviews of different content types in one report. E.g. Java, SQL, HTML. Or several Java reviews with different settings. E.g. Java and JSP.
- More inspectors
- Compiled stylesheets to speed-up report generation.
- …

Code structure visualization



Help request

Jsel & Hammurapi provide means to collect all necessary info to build such a picture.

A volunteer is welcome to implement visualization part.