Design Pattern Density Defined

Dirk Riehle

Friedrich-Alexander-University of Erlangen-Nuremberg

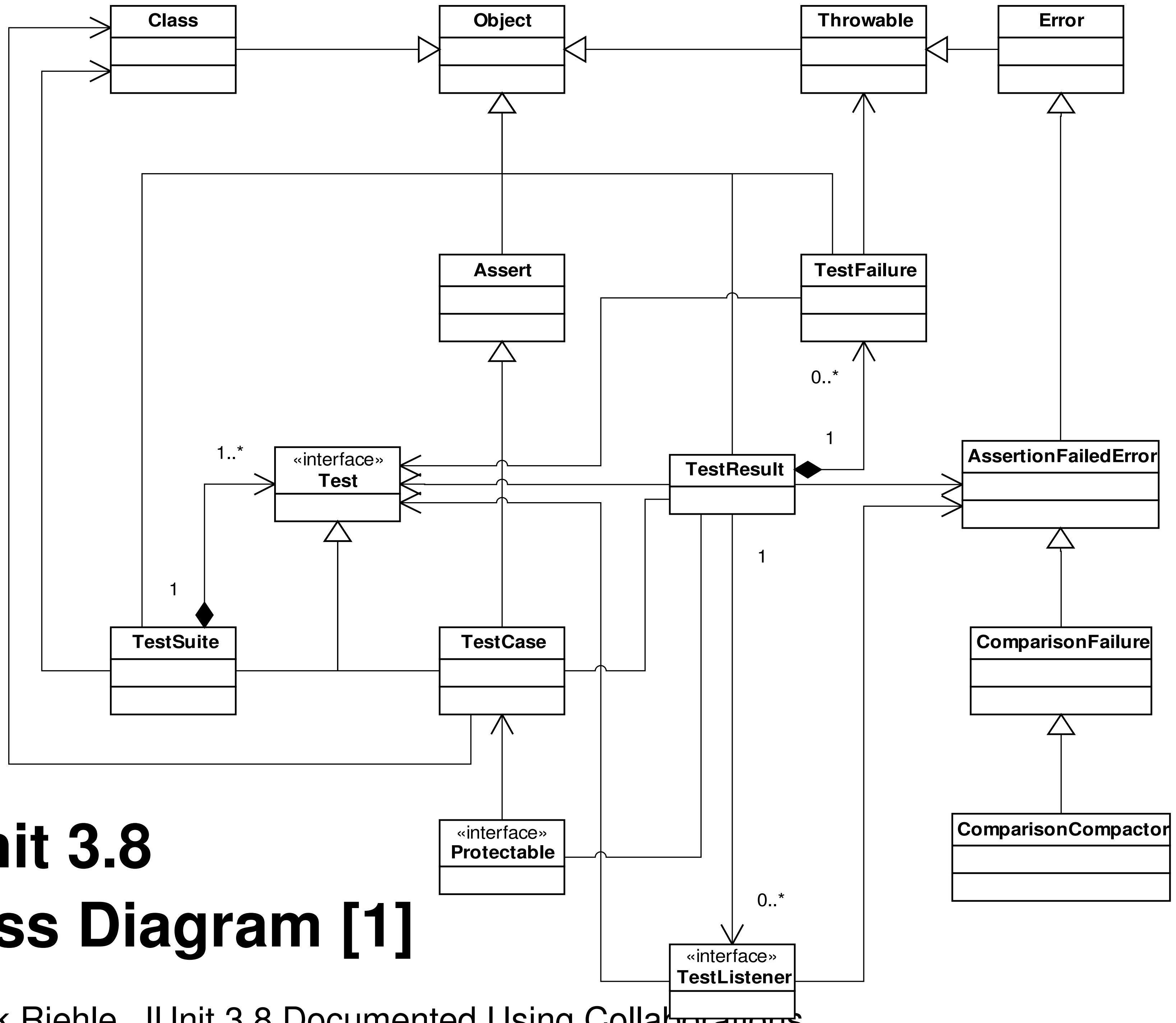
dirk@riehle.org - http://dirkriehle.com - @dirkriehle

OOPSLA / Onward! 2009 - October 27, 2009

Contributions of Paper and Talk

Defines “design pattern density” metric
Defines instrument to assess metric

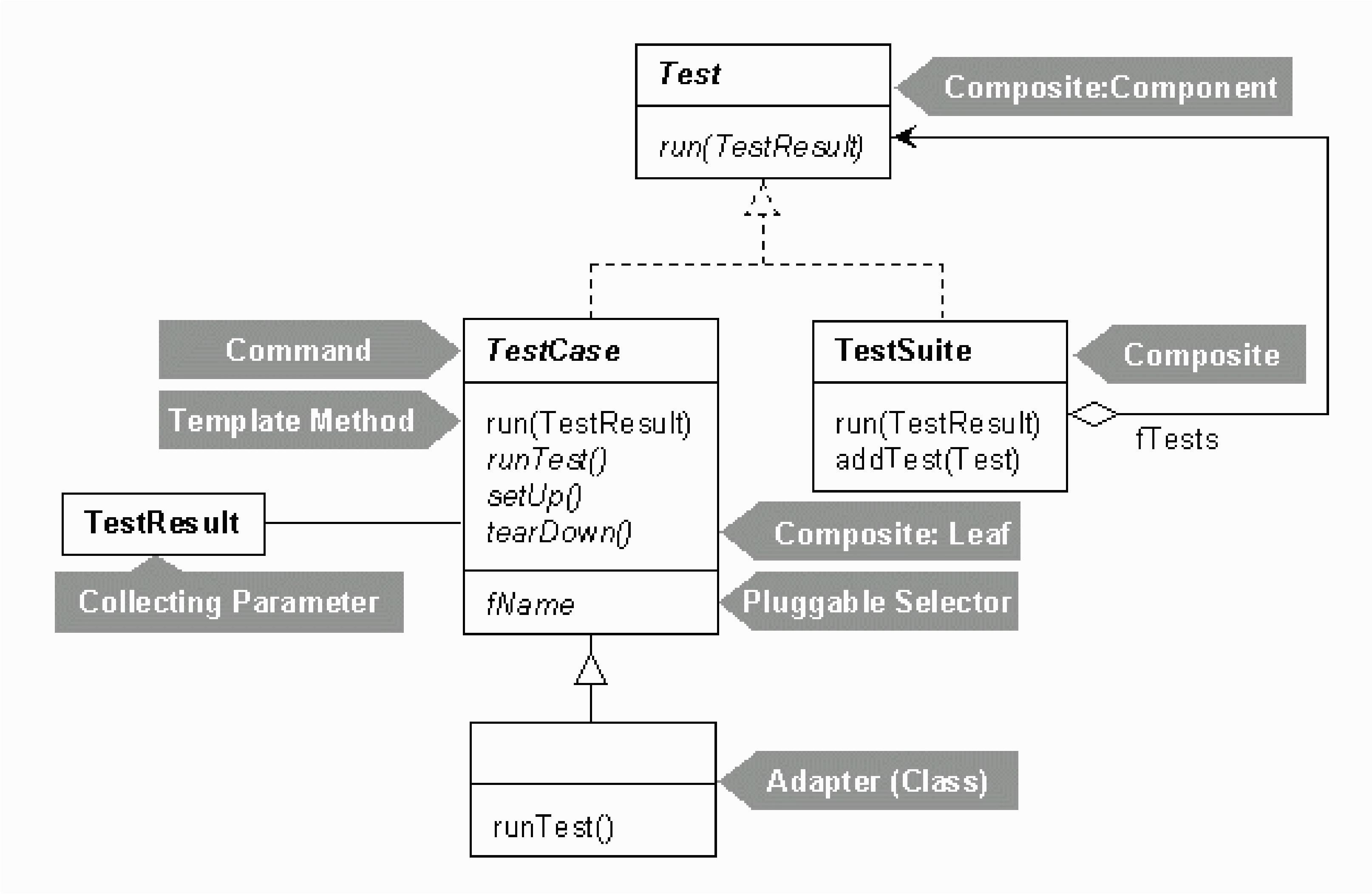
Allows for formalizing hypotheses
Allows for evaluating hypotheses



JUnit 3.8
Class Diagram [1]

[1] Dirk Riehle. JUnit 3.8 Documented Using Collaborations.
In Software Engineering Notes (March 2008). ACM Press, 2008. Article 5.

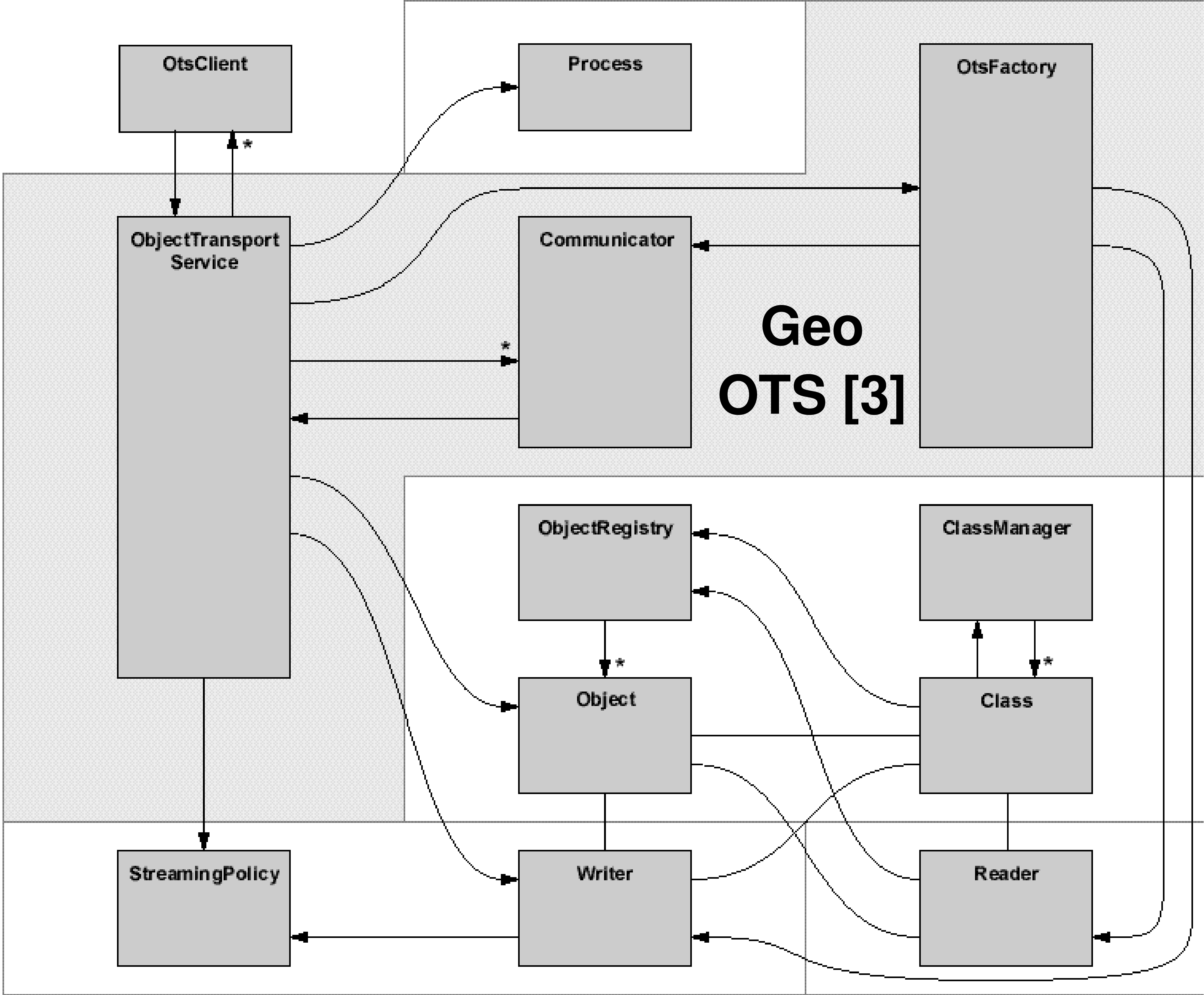
JUnit 3.8 Core Design with Patterns [2]



[2] Kent Beck and Erich Gamma. JUnit: A Cook’s Tour. Available from http://junit.sourceforge.net/doc/cookstour/cooks-tour.htm

Beck and Gamma on JUnit's Design

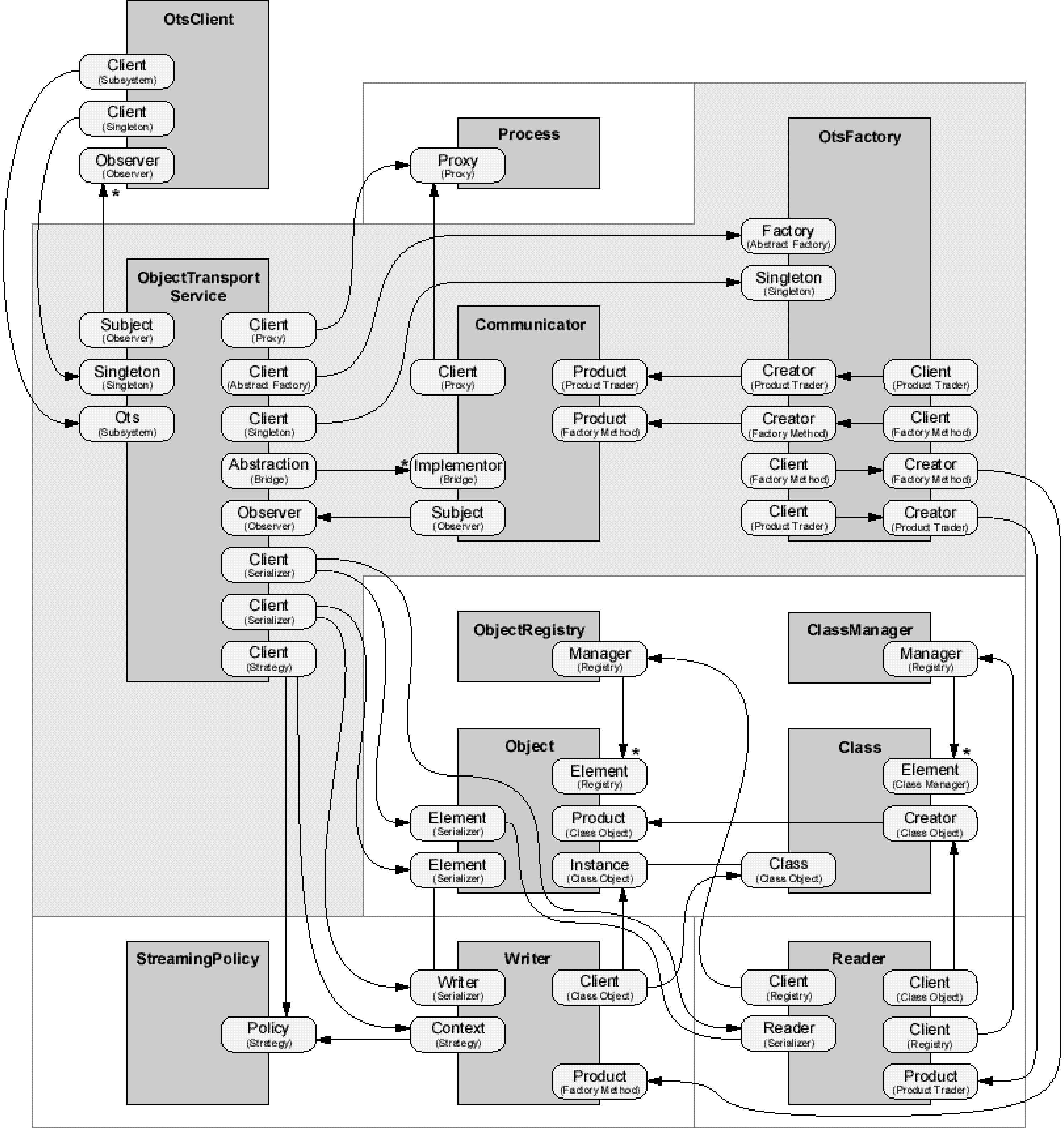
> “Notice how TestCase, the central abstraction in the framework, is involved in four patterns. Pictures of ***mature object designs show this same ‘pattern density’.*** The star of the design has a rich set of relationships with the supporting players.” [2]



Geo OTS [3]

[3] Dirk Riehle, Roger Brudermann, Thomas Gross, Kai-Uwe Matzel. “Pattern Density and Role Modeling of an Object Transport Service.” ACM Computing Surveys 32, 1es (March 2000). Article 10.

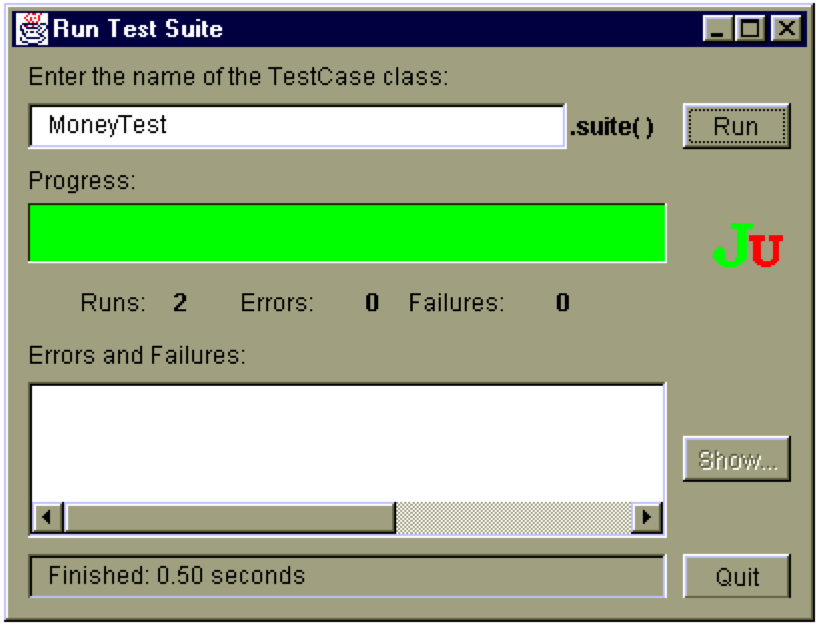
Geo Transport Service Design With Object Collaborations and Design Patterns



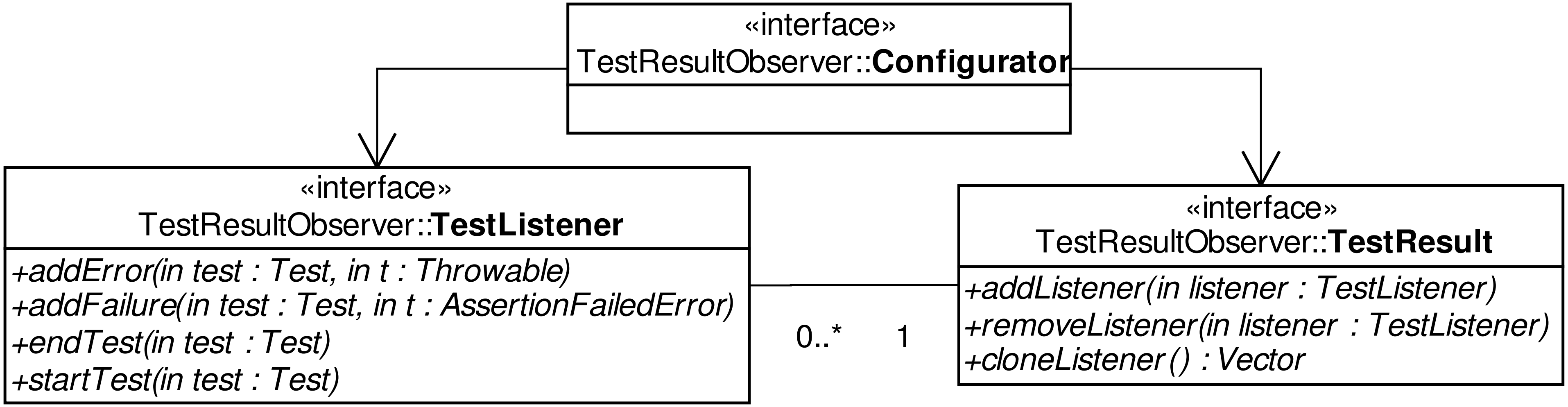
Design Pattern Density **Intuition**

The **pattern density** of an object-oriented design is the *percentage of functionality* in the design defined and explained by design patterns.

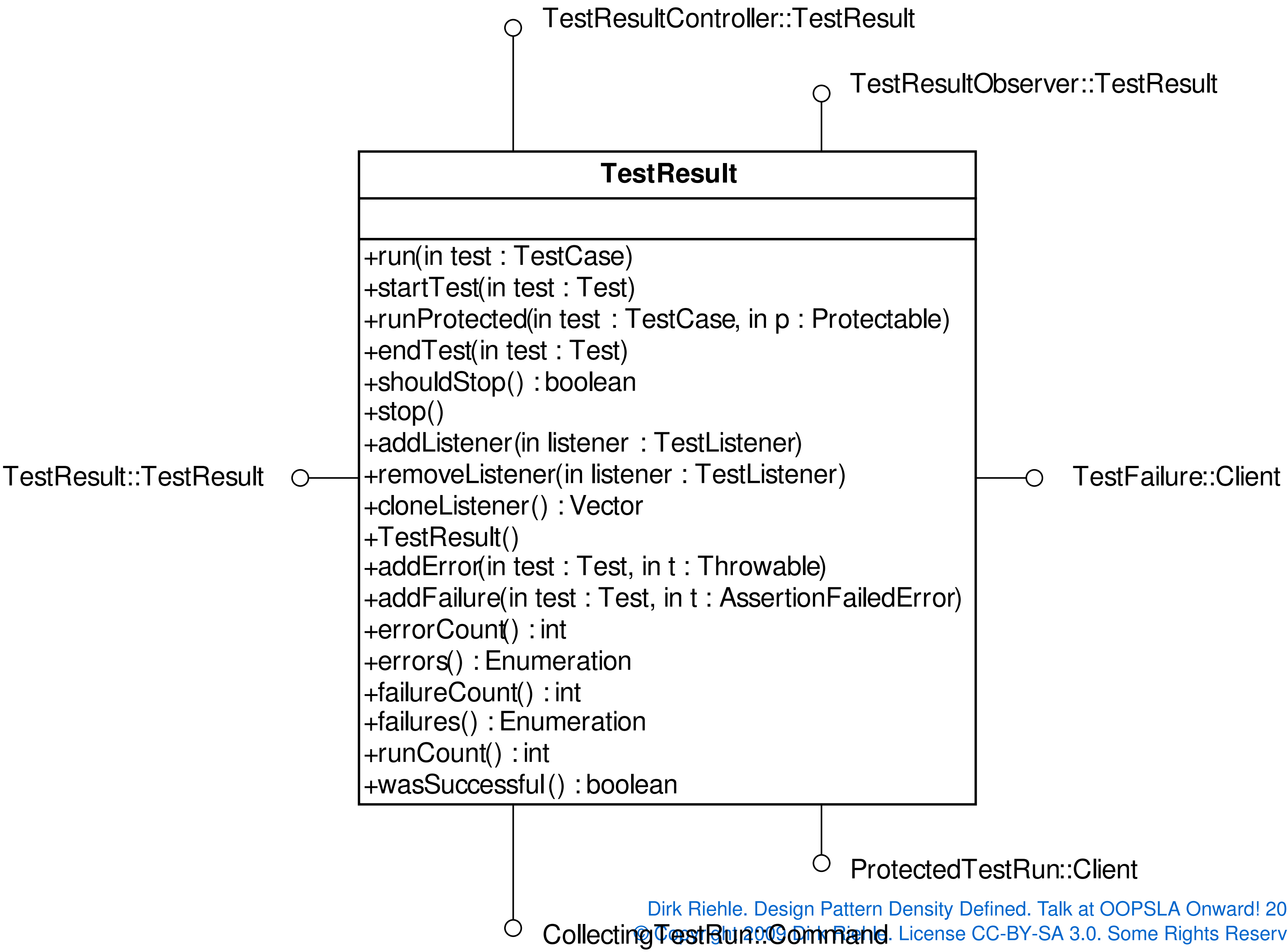
Object Collaboration: TestResultObserver



As JUnit runs over a complete TestSuite, it collects test results (failures and successes); clients can register their interest in the test progress. One such client is the “green/red” progress bar.


«interface»
TestResultObserver::**Configurator**
«interface»
TestResultObserver::**TestListener**
+addError(in test : Test, in t : Throwable)
+addFailure(in test : Test, in t : AssertionFailedError)
+endTest(in test : Test)
+startTest(in test : Test)
0..*
1
«interface»
TestResultObserver::**TestResult**
+addListener(in listener : TestListener)
+removeListener(in listener : TestListener)
+cloneListener() : Vector

Class Focus (Complementary View)



TestResultController::TestResult

TestResultObserver::TestResult

TestResult

+run(in test : TestCase)
+startTest(in test : Test)
+runProtected(in test : TestCase, in p : Protectable)
+endTest(in test : Test)
+shouldStop() : boolean
+stop()
+addListener(in listener : TestListener)
+removeListener(in listener : TestListener)
+cloneListener() : Vector
+TestResult()
+addError(in test : Test, in t : Throwable)
+addFailure(in test : Test, in t : AssertionFailedError)
+errorCount() : int
+errors() : Enumeration
+failureCount() : int
+failures() : Enumeration
+runCount() : int
+wasSuccessful() : boolean

TestResult::TestResult

TestFailure::Client

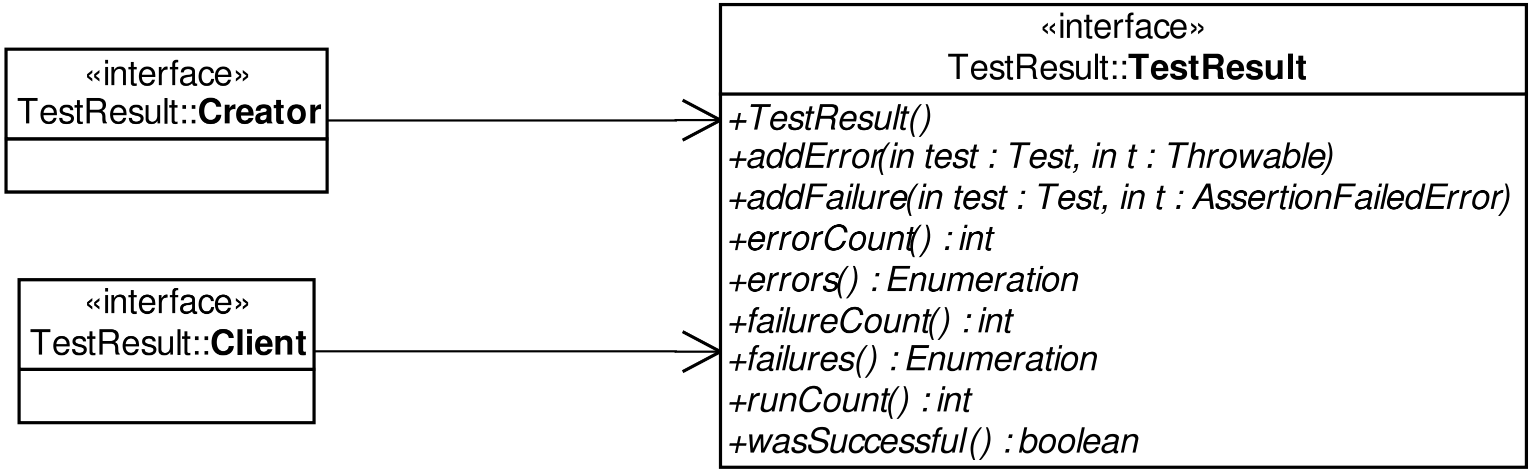
ProtectedTestRun::Client

CollectingTestRun::Command

Dirk Riehle. Design Pattern Density Defined. Talk at OOPSLA Onward! 2009.

JUnit 3.8 Documented Using Collaborations

1.



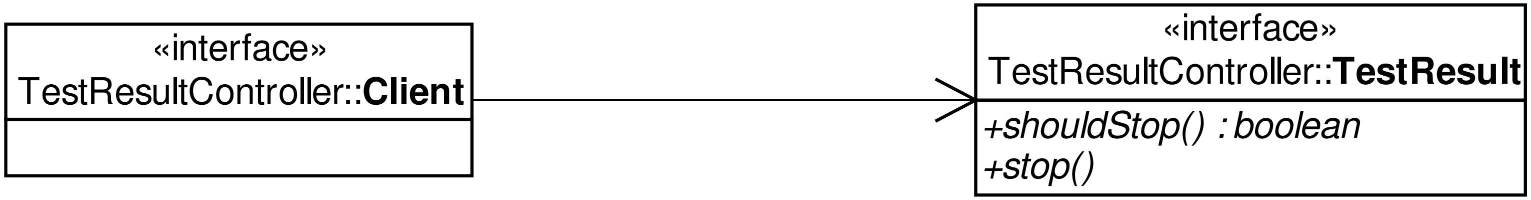
«interface»
TestResult::**Creator**

«interface»
TestResult::**Client**

«interface»
TestResult::**TestResult**

+TestResult()
+addError(in test : Test, in t : Throwable)
+addFailure(in test : Test, in t : AssertionFailedError)
+errorCount() : int
+errors() : Enumeration
+failureCount() : int
+failures() : Enumeration
+runCount() : int
+wasSuccessful() : boolean

2.

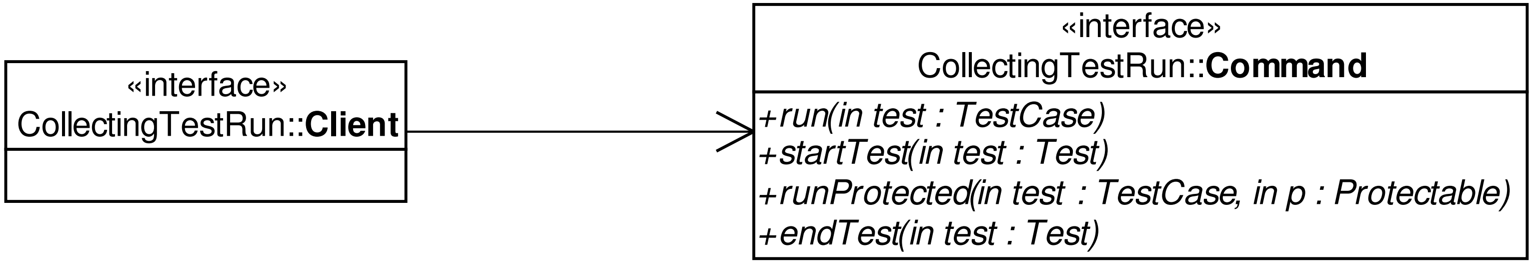


«interface»
TestResultController::**Client**

«interface»
TestResultController::**TestResult**

+shouldStop() : boolean
+stop()

3.



«interface»
CollectingTestRun::**Client**

«interface»
CollectingTestRun::**Command**

+run(in test : TestCase)
+startTest(in test : Test)
+runProtected(in test : TestCase, in p : Protectable)
+endTest(in test : Test)

... and many more, see [1]

Design Pattern Density **Defined** [4]

The **pattern density** of an object-oriented design is ***the percentage of its collaborations*** that are design pattern instances.

[4] Dirk Riehle. “Design Pattern Density Defined.” In Proceedings of the 2009 Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA Onward! ‘09). ACM Press, 2009.

Dealing with the Inheritance Interface

- Intra-object communication
  - In role modeling, is the same as inter-object communication
  - Within collaborations objects frequently talk to themselves!

```
public collaboration TestRunMethod {

   free role Client {
      // no methods
   }

   role TemplateMethod {
      public void runBare() throws Throwable;
   }

   role PrimitiveMethods {
      protected void runTest() throws Throwable;
      protected void setUp() throws Exception;
      protected void tearDown() throws Exception;
   }
}

// role to class assignments
public class TestCase provides
   TemplateMethod, PrimitiveMethods ...
```

Name of Collaboration	Number of Collaborations	Number of Roles in Collaboration	Number of Methods in Collaboration	Is it a pattern? If so which?
TestCase	1	2	4	-
TestSuite	1	2	4	-
TestSuiteTestCreation	1	2	4	-
TestRun	1	2	1	Command
TestCaseTestRun	1	2	2	-
TestSuiteTestRun	1	2	1	-
TestHierarchy	1	3	13	Composite
TestResult	1	3	9	Collecting Parameter
TestResultController	1	2	2	-
TestResultObserver	1	3	7	Observer
CollectingTestRun	1	2	4	Command
ProtectedTestRun	1	3	2	Adapter
TestRunMethod	1	2	4	Template Method
Assertions	1	2	38	-
TestFailure	1	2	6	-
AssertionFailedError	1	2	2	-
ComparisonFailure	1	2	3	-
ComparisonCompactor	1	2	2	Strategy
CompactMethod	1	2	6	Composed Method

JUnit's Design Pattern Density

JUnit 3.8 Case Study	
Number of classes/interfaces	11
Number of collaborations	19
Number of pattern instances	9
Number of roles in total	42
Ratio roles per class/interface	3.8
Design pattern density	**47%**

The paper [4] presents a few more case studies.

Some More Case Studies

Case study	**[1]**	**[2]**	**[3]**
Number of interfaces and interface classes	17	13	20
Number of collaborations	34	20	28
Number of pattern instances	20	12	20
Number of roles assigned to classes	75	44	66
Ratio roles per class/inter-face	4.4	3.4	3.3
Design pattern density (interface architecture)	**59%**	**60%**	**71%**
[1] - The Geo framework			
[2] - The KMU Desktop framework			
[3] - The JHotDraw core framework			

A Step Back: The Big Picture

Defined metric “design pattern density”
Defined instrument to assess metric

What does the metric measure?
What can we predict from the measure?

Some Hypotheses Using Design Pattern Density

1. As a framework matures, the design pattern density of a framework increases.
2. As a framework matures, its design pattern density approaches a fixed point value.

   ...

7. The closer a framework version’s design pattern density is to its fixed point, the easier on average the framework is to learn and use.

We can now evaluate these hypotheses

Limitations of Metric and Instrument

- Assessing designs using collaborations and patterns is hard work
- After the fact assessment is even harder given muddled designs
- Some of these assessments are likely to be subjective
- Object collaborations don't fully cover all aspects of a design
- A practical application of the metric may be subject to gaming

Future Work

- Extend metric and instrument from collaborations to aspects
- Broadly create more case study material
- Evaluate hypotheses of interest

Document License: CC-BY-SA 3.0

Questions? Feedback!

dirk@riehle.org - http://dirkriehle.com - @dirkriehle