Crap4J

Nigel talking crap as usual ....

Code Complexity Metric

- McCabe's Cyclomatic Complexity measures number of paths through the source code
- Measured using static analysis
- Studies have shown CC >10 has higher risk of defects
- To get full coverage, test count must be >= CC

Code Coverage Metric

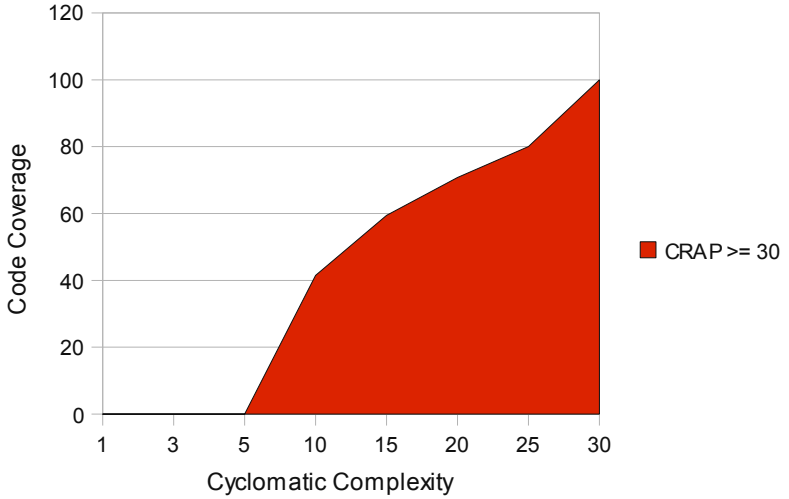
- Measures %age of code that has been *run*
  - Doesn't ensure that code has been *tested*, just *run*.
  - Useful as an anti-metric. 80% coverage means that 20% definitely hasn’t been tested.
  - Can be used to identify areas of risk (untested code)
- Different ways of measuring produce different results:
  - Line coverage
  - Branch coverage
  - Path coverage

Crap4J Metric

- Detects code that is risky and/or difficult to maintain
- Combines code complexity and coverage
  - CRAP(m) = comp(m)^2 * (1 – cov(m)/100)^3 + comp(m)

  where comp(m) is the cyclomatic complexity of method m,

  and cov(m) is the test code coverage



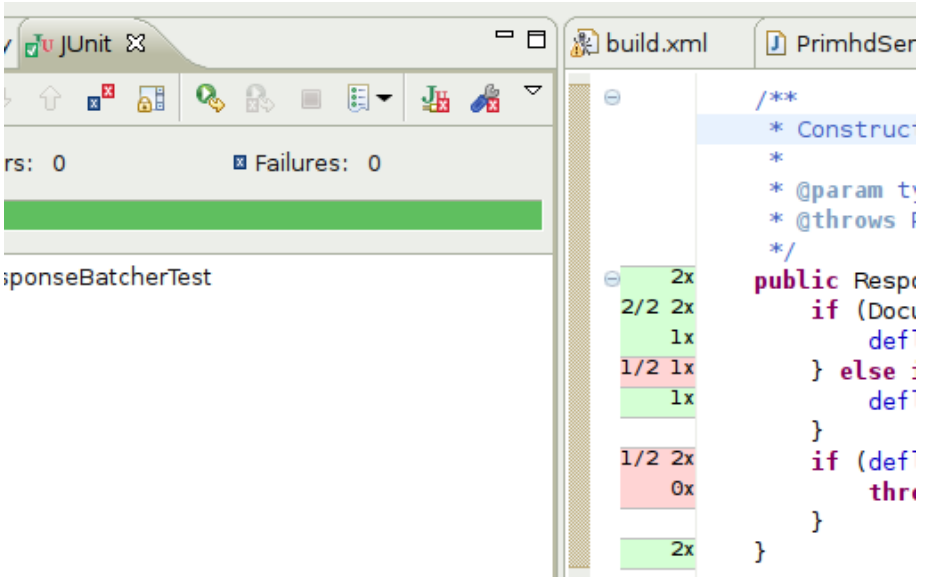
- CRAP threshold set to 30 by default, should be lowered for new code.
- The metric may evolve over time.

Eclipse plugin

- Update site - http://www.crap4j.org/downloads/update
- Click on project node in Package Explorer, then click on crap4j button

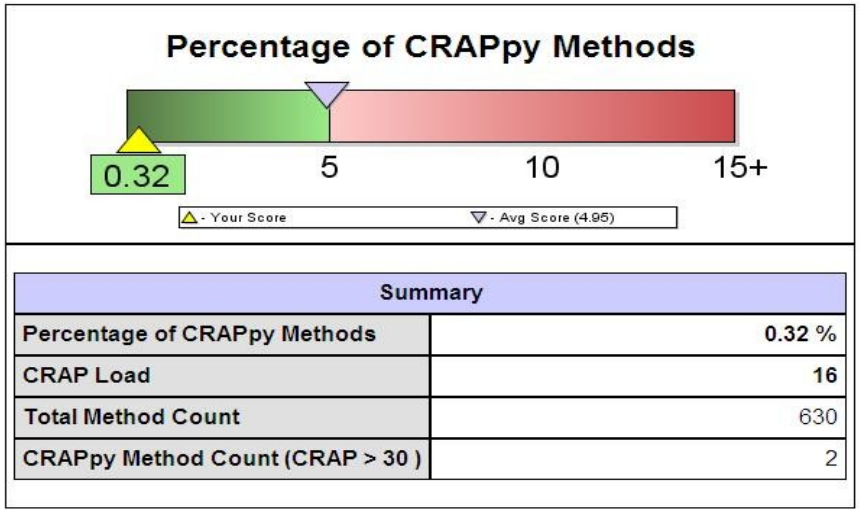
- Wait a while.....
- ..... and CRAP report will appear

- Plugin also includes the excellent Agitar TestRunner



CRAP report

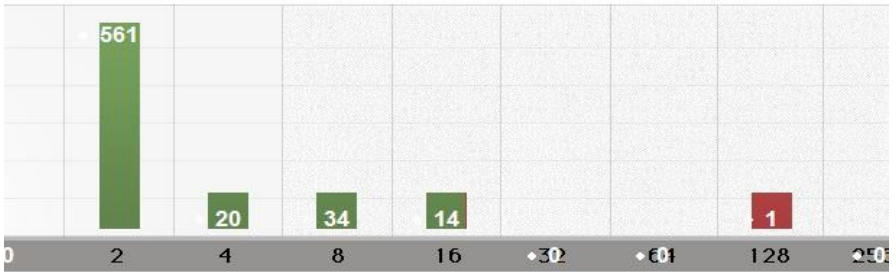
- Compares against industry average



Summary	
Percentage of CRAPpy Methods	0.32 %
CRAP Load	16
Total Method Count	630
CRAPpy Method Count (CRAP > 30)	2

Share and Compare

Share your results, (anonymously or publicly) & compare your project to others.



- Allows you to drill down and show CRAP, coverage and complexity per method

Method	Complexity	Coverage	CRAP	CRAP Load
public int getCyclomaticComplexity(java.lang.String, com.agitar.org.objectweb.asm.tree.MethodNode) org.crap4j.complexity . CyclomaticComplexity	3	0.00 %	12.00	0
private java.util.Map<java.lang.Integer, java.util.List<org.crap4j.external.MyCoveragePoint>> readCoveragePoints(java.io.DataInput) org.crap4j.external . SuperRunnerCoverageReader	4	26.67 %	10.31	0

Ant task

- Download from www.crap4j.org
- Provides options to restrict the classes to be analyzed.
- Currently needs a workaround to run on Windows, see http://tutansblog.blogspot.com.
- Generates the CRAP report.

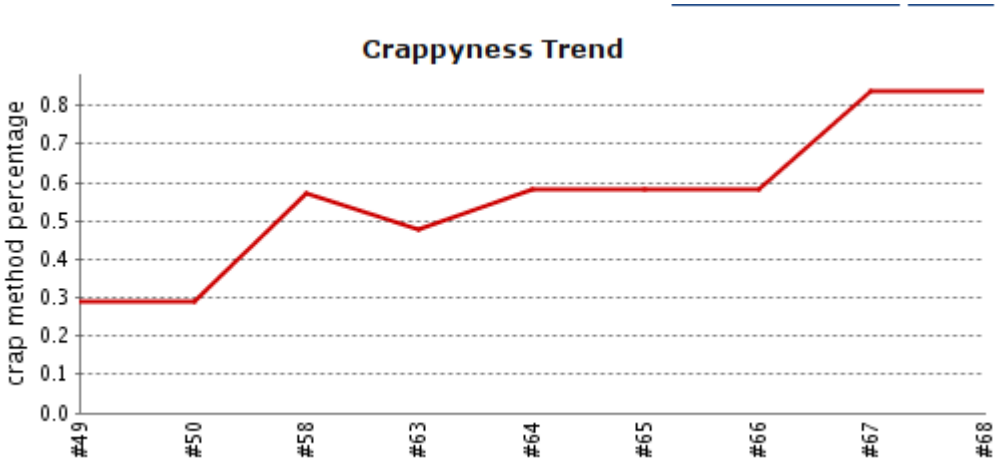
```
<crap4j projectdir="${basedir}" outputDir="${reports.crap4j.dir}" dontTest="false" debug="false">
        <classes>
                <pathElement location="${dest.dir}" />
        </classes>

        <srces>
                <pathElement location="${src.dir}" />
        </srces>

        <testClasses>
                ........
```

Hudson plugin

- Plugin for Hudson CI server
- Displays CRAP trend graphs
- Drill down to CRAPpy methods per build
- Takes a while to run – run it in a nightly or slow feedback loop



How do I reduce my CRAP?

- Refactor your code to reduce complexity
- Add more tests to get better code coverage
- The CRAP load on the report gives an indication of the minimum amount of work needed.

Anything to beware of?

- Does not currently work with Eclipse 3.4
- Project has been dormant, but is being revived now. (It's a good time to get involved eg. submitting patches, porting to other IDEs).
- Only calculates coverage from JUnit tests
- Only measures CRAP within a method, not higher order anti-patterns such as excess coupling.
- Eclipse plugin does not allow you to restrict the classes to be analyzed (but Ant task does)
- Adds AGITAR_ entries to Eclipse classpath (but these can be ignored).
- Don’t focus solely on the metric:
  - peer review code and tests to ensure quality
  - consider other risks (business, security, etc..)

Crap4J - Alberto's 3 point plan

1. Know your crap
   - ✓ Use Crap4J to measure your current code base
2. Cut the crap
   - ✓ Increase your test coverage
   - ✓ Reduce your code complexity
3. Don't take crap from nobody
   - ✓ Measure code quality from 3rd party suppliers
   - ✓ Put Quality Level Assurance agreements in place