Measuring code coverage

Uncovering the Atlassian Clover engine
About me

Marek Parfianowicz

• Support Engineer and Software Developer at Spartez (since 2012)
• Senior Software Engineer at Lufthansa Systems (2004-2012)

• Technical University of Gdańsk, computer science
• Java, C/C++

• developer of the Atlassian Clover product
Agenda

A bit of theory ...

- What a code coverage is not? Why to use it?
- Collecting coverage data
- Code coverage metrics

Dive into the code...

- Parsing Java
- Parsing Groovy

Harness the runtime ...

- Clover's coverage recorders
- Multi-threaded application
- Multi-threaded tests
- Multiple JVMs
What a code coverage is NOT?

• it's **not** a silver bullet – it's just a metric

• it's **not** a „positive” metric
  → it does **not** measure how your code is good
  → it does **not** tell if your tests are correct
  → it tells how much *crap* do you have

• it's **not** a „must have”
  → follow the "80-20" rule
Why to use code coverage?

• to identify risky code
  – code coverage && code metrics && test failure history

• to assist in development
  – coverage highlighting in text editor
  – code reviews

• to speed-up test execution
  – selecting subset of tests
  – fail-fast approach
<table>
<thead>
<tr>
<th>Feature</th>
<th>JVMTI</th>
<th>Bytecode instr.</th>
<th>Source instr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method coverage</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Statement coverage</td>
<td>line only</td>
<td>indirectly</td>
<td>yes</td>
</tr>
<tr>
<td>Branch coverage</td>
<td>indirectly</td>
<td>indirectly</td>
<td>yes</td>
</tr>
<tr>
<td>Works without source</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Requires separate build</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Works without specialized runtime</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Gathers source metrics</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Compilation time</td>
<td>no impact</td>
<td>variable</td>
<td>variable</td>
</tr>
<tr>
<td>Runtime performance</td>
<td>high impact</td>
<td>variable</td>
<td>variable</td>
</tr>
<tr>
<td>Container friendly</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
Method coverage

measures whether a method was entered at all during execution

class MethodCoverage {

    static String foo(boolean b, int i) {
        if (b) {
            return "true";
        }
        if (i > 0) {
            return "positive";
        } else {
            return "negative or zero";
        }
    }

    public static void main() {
        foo(true, 0);
    }
}
Statement coverage

measures whether given statement was executed at least one time

class StatementCoverage {
    static String foo(boolean b, int i) {
        String s;
        if (b) {
            s += "true";
        }
        if (i > 0) {
            s += "positive";
        } else {
            s += "negative or zero";
        }
        return s;
    }
    public static void main() {
        foo(true, 5);
        foo(false, 10);
    }
}
Branch coverage

measures which possible branches in flow control structures are followed

class BranchCoverage {
    static String foo(boolean b, int i) {
        String s;
        if (b) { // true only
            s += "true";
        }
        if (i > 0) { // true & false
            s += "positive";
        } else { // i <= 0
            s += "negative or zero";
        }
    }
    public static void main() {
        foo(true, 0);
        foo(true, 10);
    }
}
Condition / decision coverage

check every possible value of input condition and decision outcome

class ConditionDecisionCoverage {
    static boolean foo(boolean a, boolean b, boolean c) {
        if ((a || b) && c) {
            return true;
        } else {
            return false;
        }
    }
    public static void main() {
        foo(true, true, true);
        foo(false, false, false);
    }
}

check every possible value of input condition and decision outcome
Modified condition / decision cov.

check every possible value of input condition and decision outcome

each condition has been shown to affect that decision independently

class ModifiedConditionDecisionCoverage {

    static boolean foo(boolean a, boolean b, boolean c) {
        if ((a || b) && c) {
            return true;
        } else {
            return false;
        }
    }

    public static void main() {
        foo(false, false, true);   // a,b are influencing
        foo(true, false, true);    // a,c are influencing
        foo(false, true, true);    // b,c are influencing
        foo(true, true, false);    // c is influencing
    }
}
Agenda

A bit of theory ...
- What a code coverage is not? Why to use it?
- Collecting coverage data
- Code coverage metrics

Dive into the code...
- Parsing Java
- Parsing Groovy

Harness the runtime ...
- Clover's coverage recorders
- Multi-threaded application
- Multi-threaded tests
- Multiple JVMs
class JavaRecognizer extends Parser; // parser class name
/* grammar rules */
variableDefinitions
    : variableDeclarator (COMMA! variableDeclarator)*
    ;

class JavaLexer extends Lexer; // lexer class name
/* tokens */
COMMA    : ',' {nc();};
IDENT
options {testLiterals=true;}
    : {nc();} ('a'..'z'|'A'..'Z'|'_'|'$') ('a'..'z'|'A'..'Z'|'_'|'0'..'9'|'$')*
    ;
ANTLR: `java.g` => `JavaLexer` + `JavaRecognizer`

```java
in = new InputStreamReader(new FileInputStream(sourceFile));
lexer = new JavaLexer(in);
filter = new CloverTokenStreamFilter(lexer);
parser = new JavaRecognizer(filter);
parser.compilationUnit();
filter.instrument(); // instrument the code
filter.write(out);    // write to output
```
Parsing Java
Definition of statement in ANTLR

statement [CloverToken owningLabel] returns [CloverToken last]
{
    CloverToken first = null;
    last = null;
}

: {first = (CloverToken)LT(1);}
{
    // if | for | while | throw etc ...
    | "return" (expression)? SEMI!
    | SEMI // empty statement
}

if (last == null) {
    last = (CloverToken)LT(0);
}

instrumentStatementBefore(first, last);
CloverToken extends antlr.CommonHiddenStreamToken {
    // List<? extends Emitter> preEmitters;
    // addPreEmitter, triggerPreEmitters
}

class CloverTokenStreamFilter extends antlr.TokenStreamHiddenTokenFilter {
    public void write(Writer outWriter) throws IOException {
        PrintWriter out = new PrintWriter(outWriter);
        CloverToken curr = this.first;
        while (curr != null) {
            curr.triggerPreEmitters(out);
            if (curr.getText() != null) out.print(curr.getText());
            curr = curr.getNext();
        }
    }
}
CloverToken instrumentStatementBefore(CloverToken start, CloverToken end) {
    start.addPreEmitter(new StatementInstrEmitter(...));
    return start;
}

class StatementInstrEmitter {
    String instr = ""
    StatementInstrEmitter(...) {
        info = db.addStatement(...);
        instr = "Recorder.inc(" + info.getDataIndex() + ");";
    }
    void emit(Writer out) throws IOException {
        out.write(instr);
    }
}
public IMoney add(IMoney m) {
    return m.addMoney(this);
}

public IMoney add(IMoney m) {
    try{
        __CLR3_1_600hckkb3w8.R.inc(3);
        __CLR3_1_600hckkb3w8.R.inc(4);
        return m.addMoney(this);
    }
    finally {
        __CLR3_1_600hckkb3w8.R.flushNeeded();
    }
}
public void testAdd() {
    ...
    try {
        ...
    } catch (java.lang.Throwable $CLV_t2$) {
        $CLV_t$ = $CLV_t2$;
        __CLR3_1_67b7bhckkb42x.R.rethrow($CLV_t2$);
    } finally {
        __CLR3_1_67b7bhckkb42x.R.globalSliceEnd(getClass().getName(), "MoneyTest.testAdd", 263, $CLV_t$);
    }
}
Parsing Groovy
Groovyc build phases

INITIALIZATION
source files are opened and environment configured

PARSING
the grammar is used to produce tree of tokens representing the source code

CONVERSION
an abstract syntax tree (AST) is created from token trees

SEMANTIC_ANALYSIS
performs consistency and validity checks that the grammar can’t check for, and resolves classes

CANONICALIZATION
complete building the AST

INSTRUCTION_SELECTION
instruction set is chosen, for example java5 or pre-java5

CLASS_GENERATION
creates the binary output in memory

OUTPUT
write the binary output to the file system

FINALIZATION
perform any last cleanup
Parsing Groovy
Groovy's AST (1)

my.jar/META-INF/services/
org.codehaus.groovy.transform.ASTTransformation:
com.acme.MyGroovy

@GroovyASTTransformation (phase = CompilePhase.INSTRUCTION_SELECTION)
public class MyGroovy implements ASTTransformation {
  public void visit(ASTNode[] astNodes, SourceUnit sourceUnit) {
    for (ClassNode clazz : sourceUnit.getAST().getClasses()) {
      new MyTransformer(sourceUnit, clazz).visitClass(clazz);
    }
  }
}

}
public class MyTransformer extends ClassCodeExpressionTransformer {
    public Expression transform(Expression expr) {
        Expression transformed = super.transform(expr);
        if (transformed instanceof ElvisOperatorExpression)
            transformed = transformElvis((ElvisOperatorExpression)transformed);
        transformed.setSourcePosition(expr);
        return transformed;
    }

    private Expression transformElvis(ElvisOperatorExpression elvis) {
        transformed = new ElvisOperatorExpression(
            new StaticMethodCallExpression(this.currentClass,
                "elvisEvalWrapper",
                new ArgumentListExpression(
                    elvis.getTrueExpression(),
                    new ConstantExpression(getDataIndex(elvis))",
                    elvis.getFalseExpression()));
        return transformed;
    }
}
Agenda

A bit of theory ...
- What a code coverage is not? Why to use it?
- Collecting coverage data
- Code coverage metrics

Dive into the code...
- Parsing Java
- Parsing Groovy

Harness the runtime ...
- Clover's coverage recorders
- Multi-threaded application
- Multi-threaded tests
- Multiple JVMs
Agenda

A bit of theory ...
- What a code coverage is not? Why to use it?
- Collecting coverage data
- Code coverage metrics

Dive into the code...
- Parsing Java
- Parsing Groovy

Harness the runtime ...
- Clover's coverage recorders
- Multi-threaded application
- Multi-threaded tests
- Multiple JVMs
Clover's coverage recorders

MODULE ONE

A.java
B.java
C.java

instrumentation session#1
timestamp 1000

clover.db

instrumentation session#2
timestamp 2000

A.class
get
B.class
get
C.class
get
getRecorder("clover.db", 1000, ...)

C.class (recompiled)
get
D.class
get
getRecorder("clover.db", 2000, ...)

MODULE TWO

E.java

instrumentation session#1
timestamp 3000

clover-two.db

getRecorder("clover-two.db", 3000, ...)

E.class
Clover's coverage recorders

- **:FixedCoverageRecorder**
  - clover.db
    - `long[] hitCounts`
    - `getRecorder("clover.db", 1000, ...)`
- **:FixedCoverageRecorder**
  - clover.db
    - `long[] hitCounts`
    - `getRecorder("clover.db", 1000, ...)`
- **:FixedCoverageRecorder**
  - clover.db
    - `long[] hitCounts`
    - `getRecorder("clover.db", 2000, ...)`
- **:FixedCoverageRecorder**
  - clover-two.db
    - `long[] hitCounts`
    - `getRecorder("clover-two.db", 2000, ...)`
- **:FixedCoverageRecorder**
  - clover-two.db
    - `long[] hitCounts`
    - `getRecorder("clover-two.db", 3000, ...)"
Multi-threaded applications

```java
public class CoverageRecorder {
    private final int[] elements;
    private final PerTestRecorder testCoverage;
    // ...

    public void inc(int index) {
        testCoverage.set(index);
        elements[index]++;
    }
}
```
Multi-threaded tests

- keep pool of active per-test recorders
  - goal: achieve maximum performance
  - volatile implementation in JDK1.4 sucks

- per-test recording strategies
  - single-threaded (no synchronization)
  - volatile (JDK1.5+)
  - synchronized blocks
public static class Volatile implements ThreadVisibilityStrategy {

    private volatile ActivePerTestRecorderAny recorders; /* a pool of recorders */

    public Volatile(CoverageRecorder coverageRecorder) {
        recorders = new ActivePerTestRecorderNone(coverageRecorder);
    }

    public synchronized void testStarted(int testRunId) { /* memory barrier flush. */
        recorders = recorders.testStarted(testRunId);
    }

    public synchronized LivePerTestRecording testFinished(int testRunId, ErrorInfo ei) {
        RecordingResult sliceAndRecorders = recorders.testFinished(testRunId, ei);
        recorders = sliceAndRecorders.recorders;
        return sliceAndRecorders.recording;
    }

    public void inc(int index) { /* no 'synchronized' */
        recorders.inc(index);
    }

}
Multiple JVMs

- JUnit
- Clover Client
- HDD

- <configuration>
  - <distributedCoverage host="localhost"/>
  - <configuration>

- Deploy
- Connect to Clover Server
- Send 'testStarted'
- Send 'testFinished' =>

- Build Server
- Clover Server
- Report Server

- Read database and coverage recordings and create report
- Create clover.db
- Read clover.db and write coverage recordings
Questions
Thank you

Clover: http://www.atlassian.com/software/clover
Contact: mparfianowicz@atlassian.com
BACKUP SLIDES
Atlassian dogfooding

- Clover with Clover
  - ca 60% coverage in total
  - 80-90% in core modules
- Bamboo with Clover
  - in a separate plan, supplementary
- JIRA with Clover
  - very limited usage as Clover does not measure coverage for JavaScript
Clover reporting features

- Top risks
- Quick wins
- Coverage map
- Test contribution
- Class complexity
- Code metrics
Test optimization

Build times (Cumulative)

https://confluence.atlassian.com/display/CLOVER/About+Test+Optimization
Android platform

Dalvik VM

- instrumentation is quite trivial in Clover
- limit of 65k methods in one image
- no bytecode manipulation
  - java, scala – ok
  - groovy – not possible

Clover-for-Android alpha

https://confluence.atlassian.com/display/CLOVER/Clover-for-Android
Tool integrations