

Code coverage criteria and their effect on test suite qualities

Mikhail Kalkov

mikhail.kalkov@gmail.com

Dzmitry Pamakha

dzmitry.pamakha@gmail.com

Master Programme in Software Engineering and Technology
Computer Science and Engineering Department
Chalmers University of Technology
Gothenburg, Sweden

December 17, 2013

Contents

- 1 Introduction
 - Motivational example
 - Research questions
- 2 Coverage criteria
- 3 Test suite qualities
- 4 Coverage visualization
- 5 Bullseye plug-in for Eclipse
- 6 Conclusion
- 7 Discussion

Motivational example

Motivational example

Some program

```
package se.chalmers.students.kalkov;

public class QuickPower {

    private int square(int n) {
        return n * n;
    }

    public int raise(int base, int power) {

        // Validate input
        if (base < 0 || power < 0) {
            throw new IllegalArgumentException(
                "Only natural numbers are supported");
        }

        // Handle special cases
        if (base == 0) {
            return 0;
        } else if (base == 1 || power == 0) {
            return 1;
        }

        // Common case: base > 1 && exponent > 0
        if (power % 2 == 1) {
            return base * raise(base, power - 1);
        } else {
            return square(raise(base, power / 2));
        }
    }
}
```

Motivational example

Some program

```
package se.chalmers.students.kalkov;

public class QuickPower {

    private int square(int n) {
        return n * n;
    }

    public int raise(int base, int power) {

        // Validate input
        if (base < 0 || power < 0) {
            throw new IllegalArgumentException(
                "Only natural numbers are supported");
        }

        // Handle special cases
        if (base == 0) {
            return 0;
        } else if (base == 1 || power == 0) {
            return 1;
        }

        // Common case: base > 1 && exponent > 0
        if (power % 2 == 1) {
            return base * raise(base, power - 1);
        } else {
            return square(raise(base, power / 2));
        }
    }
}
```

Unit tests

```
package se.chalmers.students.kalkov;

import static org.junit.Assert.assertEquals;

public final class QuickPowerTest {

    private QuickPower qp;

    @Before
    public void setUp() {
        qp = new QuickPower();
    }

    @Test(expected = IllegalArgumentException.class)
    public void testNegativeBase() {
        qp.raise(-1, 10);
    }

    @Test(expected = IllegalArgumentException.class)
    public void testNegativePower() {
        qp.raise(10, -1);
    }

    @Test
    public void testBaseIsOne() {
        assertEquals(1, qp.raise(1, 10));
    }

    @Test
    public void testCommonCases() {
        assertEquals(16, qp.raise(2, 4));
        assertEquals(3125, qp.raise(5, 5));
    }
}
```

Motivational example

How do you know that you have not forgotten to test some behaviour?

Motivational example

It turns out many people use *code coverage analysis* to answer this question.

Motivational example

It turns out many people use *code coverage analysis* to answer this question.

```
package se.chalmers.students.kalkov;

public class QuickPower {

    private int square(int n) {
        return n * n;
    }

    public int raise(int base, int power) {

        // Validate input
        if (base < 0 || power < 0) {
            throw new IllegalArgumentException(
                "Only natural numbers are supported");
        }

        // Handle special cases
        if (base == 0) {
            return 0;
        } else if (base == 1 || power == 0) {
            return 1;
        }

        // Common case: base > 1 && exponent > 0
        if (power % 2 == 1) {
            return base * raise(base, power - 1);
        } else {
            return square(raise(base, power / 2));
        }
    }
}
```

```
package se.chalmers.students.kalkov;

import static org.junit.Assert.assertEquals;

public final class QuickPowerTest {

    private QuickPower qp;

    @Before
    public void setUp() {
        qp = new QuickPower();
    }

    @Test(expected = IllegalArgumentException.class)
    public void testNegativeBase() {
        qp.raise(-1, 10);
    }

    @Test(expected = IllegalArgumentException.class)
    public void testNegativePower() {
        qp.raise(10, -1);
    }

    @Test
    public void testBaseIsOne() {
        assertEquals(1, qp.raise(1, 10));
    }

    @Test
    public void testCommonCases() {
        assertEquals(16, qp.raise(2, 4));
        assertEquals(3125, qp.raise(5, 5));
    }
}
```


Research questions

- What are common coverage criteria and their limits of applicability?

Research questions

- What are common coverage criteria and their limits of applicability?
- Which test suite qualities can be measured or affected by utilizing code coverage data?

Research questions

- What are common coverage criteria and their limits of applicability?
- Which test suite qualities can be measured or affected by utilizing code coverage data?
- What are common ways to present code coverage data and do they faithfully reveal important test suite qualities?

Research questions

- What are common coverage criteria and their limits of applicability?
- Which test suite qualities can be measured or affected by utilizing code coverage data?
- What are common ways to present code coverage data and do they faithfully reveal important test suite qualities?
- How to implement an Eclipse plug-in for BullseyeCoverage so that code coverage data can be fully utilized?

General considerations

- Place of coverage analysis in software testing

General considerations

- Place of coverage analysis in software testing
- Boundaries of software under test

General considerations

- Place of coverage analysis in software testing
- Boundaries of software under test
- Source code and object code coverage

General considerations

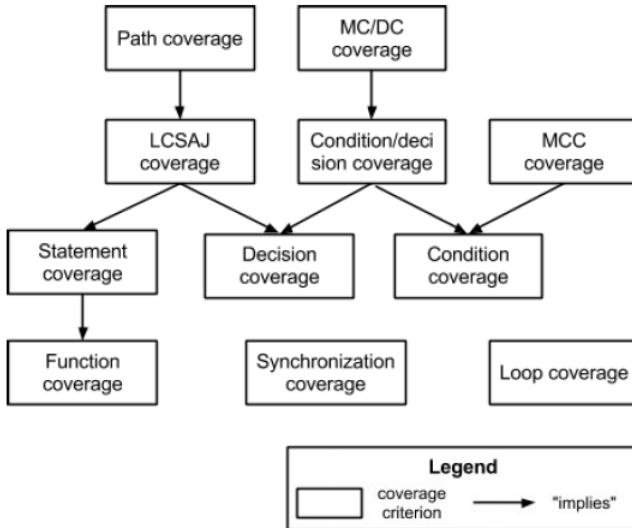
- Place of coverage analysis in software testing
- Boundaries of software under test
- Source code and object code coverage
- Control flow and data flow criteria

Coverage criteria defined in DO-178B/ED-12B

Requirements - Criteria	Stat.	Decis.	Cond.	C/D	MC/DC	MCC
Every point of entry and exit in the program has been invoked at least once		+	+	+	+	+
Every statement in the program has been invoked at least once						
Every decision in the program has taken all possible outcomes at least once		+		+	+	+
Every condition in a decision in the program has taken all possible outcomes at least once			+	+	+	+
Every condition in a decision has been shown to independently affect that decisions outcome					+	+
Every combination of condition outcomes within a decision has been invoked at least once						+

Reference: *Hayhurst-Veerhusen-Chilenski-Rierison 2001. A practical tutorial on MC/DC*

Criteria overview



Verified qualities

Test **case** qualities

- necessary
- correct
- falsifiable
- traceable to its source of origin
- independent from other test cases
- atomic
- efficient
- maintainable
- consistent in the returned value

Verified qualities

Test **case** qualities

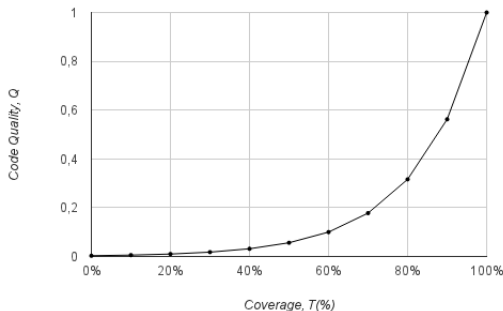
- necessary
- correct
- falsifiable
- traceable to its source of origin
- independent from other test cases
- atomic
- efficient
- maintainable
- consistent in the returned value

Test **suite** qualities

- complete
- effective
- minimal
- consistent
- modifiable
- regression-sensitive

Defect detection

- Errors of omission
- Defect clusterization
- Relationship to post-release defects
- Regression sensitivity
- Test suite minimization
- Generation of test cases



Proposed qualities

Test **case** qualities

- necessary
- correct
- falsifiable
- traceable to its source of origin
- independent from other test cases
- atomic
- efficient
- maintainable
- consistent in the returned value

Test **suite** qualities

- complete
- effective
- minimal
- consistent
- modifiable
- regression-sensitive

Coverage data presentation

Case study of several code coverage tools with the following goals:

- Which coverage criteria the tools support and how they present code coverage data
- Which test suite qualities can be measured or affected using the data from the tools

Coverage data presentation

Case study of several code coverage tools with the following goals:

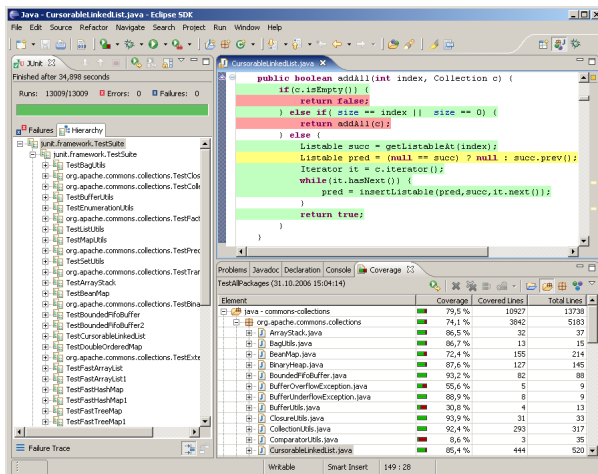
- Which coverage criteria the tools support and how they present code coverage data
- Which test suite qualities can be measured or affected using the data from the tools

The following user stories are utilized:

- Test suite completeness: reveal insufficiently tested components
- Test suite minimization: run only relevant test cases to decrease testing time
- Test suite prioritization: run tests in specific order to detect regressions quickly
- Test suite maintainability: easily connect test cases and parts of software under test
- Test case generation: generate missing test cases based on existing tests and code coverage

Eclemma

Supports line, statement, method coverage for Java.



The screenshot shows the Eclipse IDE with the Eclemma plugin. The main editor displays the 'addAll' method of 'CursorableLinkedList.java'. The code is color-coded: green for lines with 100% coverage, red for lines with 0% coverage, and yellow for lines with partial coverage. The left sidebar shows a project hierarchy with a 'Failures' view. The bottom pane shows a 'Coverage' table with columns for Element, Coverage, Covered Lines, and Total Lines.

Element	Coverage	Covered Lines	Total Lines
java - commons-collections	79,5 %	10927	13738
org.apache.commons.collections	74,1 %	3842	5183
ArrayStack.java	86,5 %	32	37
BagUtils.java	86,7 %	13	15
BeanMap.java	72,4 %	155	214
BinaryHeap.java	87,6 %	127	145
BoundedFifoBuffer.java	93,2 %	82	88
BufferOverflowException.java	55,6 %	5	9
BufferUnderflowException.java	88,9 %	8	9
BufferUtils.java	30,8 %	4	13
ClosureUtils.java	93,9 %	31	33
CollectionUtils.java	92,4 %	293	317
ComparatorUtils.java	8,6 %	3	35
CursorableLinkedList.java	85,4 %	444	520

Source code is annotated with green, red or yellow lines.

Coverage statistics view provides line coverage on Java sources organized as a tree.

Eclemma addresses only the test suite completeness.

Clover

Clover supports statement, decision and function criteria for Java. Like Eclemma, Clover employs source code annotations and an aggregate coverage statistics view to visualize measurements. Additionally, Clover maps coverage provided by each test case to the source code.

Tests that hit line # 648

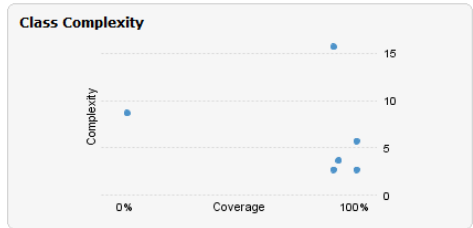
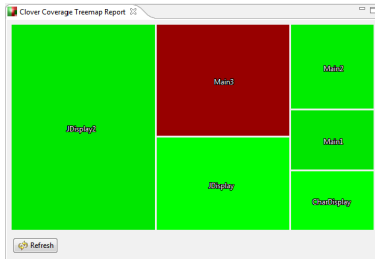
Highlight	Test Contribution	Test	Result
652 4	↑	com.google.inject.internal.MapMakerTestSuite.ComputingTest.testRecomputeAfterReclamation	PASS
653 4	↑	com.google.inject.internal.MapMakerTestSuite.ReferenceCombinationTestSuite.MapTest.testClear	PASS
654 0	↑	com.google.inject.internal.MapMakerTestSuite.ReferenceMapTest.testInternedKeyCleanupWithWeakValue	PASS
655 4	↑	com.google.inject.internal.MapMakerTestSuite.ReferenceMapTest.testKeyCleanupWithWeakValue	PASS
656 4	↑	com.google.inject.internal.MapMakerTestSuite.ReferenceMapTest.testValueCleanupWithSoftKey	PASS
657 4	↑	com.google.inject.internal.MapMakerTestSuite.ReferenceMapTest.testInternedKeyCleanupWithSoftValue	PASS
658	↑	com.google.inject.internal.MapMakerTestSuite.ReferenceMapTest.testValueCleanupWithWeakKey	PASS
659	↑	com.google.inject.internal.MapMakerTestSuite.ReferenceMapTest.testKeyCleanupWithSoftValue	PASS
660	↑	com.google.inject.internal.MapMakerTestSuite.ComputingTest.testRecomputeAfterReclamation	PASS
661 2335	↑		
662	↑		
663	↑		
664	↑		
665	↑		

// a segment array (the compile has a tough time with arrays of
// inner classes of generic types apparently). Safe because we're

Clover

More advaced features of Clover:

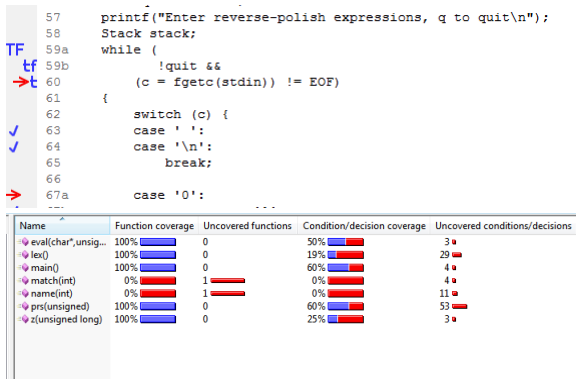
- running tests based on their failure history or code changes
- reordering of test cases based on their comparable coverage
- treemap and complexity views



Thus, Clover satisfies the test suite completeness, minimization, prioritization and maintainability stories.

BullseyeCoverage

Measures coverage for C/C++ using function and condition/decision coverage metrics by instrumenting source code after a preprocessor but before a compiler.



Column with coverage information is attached to read-only plain text code viewer.

Coverage statistics view is similar to the one used by EclEmma and Clover.

Bullseye addresses the test suite completeness.

CTC++

CTC++ is another tool for C/C++ that supports condition/decision coverage as well as more advanced multicondition coverage (MCC) and modified condition/decision coverage (MC/DC) criteria.

```

1  /* File calc.c ----- */
2  #include "calc.h"
3  /* Tell if the argument is a prime (ret 1) or not (ret 0) */

6      4  int is_prime(unsigned val)
7      {
8          unsigned divisor;

2      4  8      if (val == 1 || val == 2 || val == 3)
0      8      1: T || _ || _
2      8      2: F || T || _
0      8      3: F || F || T
8      8      4: F || F || F
-      8      MC/DC (cond 1): 1 - 4
-      8      MC/DC (cond 2): 2 + 4
-      8      MC/DC (cond 3): 3 - 4

2      9      return 1;
2      2  10     if (val % 2 == 0)
2      11     return 0;
0      2  - 12     for (divisor = 3; divisor < val / 2; divisor += 2)
0      13     {
0      0  - 14         if (val % divisor == 0)
0      - 15             return 0;
16     }

```

Coverage information is shown in HTML files in the web browser via a column attached to the source code.

CTC++ satisfies only the test suite completeness.

Coverage visualization conclusion

- All examined code coverage tools support test suite completeness analysis by annotating source code with coverage information and providing a coverage summary view
- Other use cases are not widely supported due to lack of integration with test frameworks

Suggestions based on case study:

- Develop code coverage tool together with a unit test framework or the whole language infrastructure
- Use logarithmic coverage ranks scale instead of linear percentage one
- IDE integration helps a lot
- Possibility to set boundaries of software under test

Requirements on Bullseye plug-in

Major functional requirements:

- Annotate source code with coverage information
- Show aggregate coverage at class/file/directory level

Other functional requirements:

- Sorting and filtering based on user selection
- Integration with Eclipse framework: persistence, code navigation, hide/unhide information on user request, update coverage information on external updates

Non-functional requirements:

- Support coverage report with up to 10000 source files and source files with up to 10000 lines
- Non-blocking UI
- Lazy loading of plug-in

Domain analysis and design

Input data

- Report file in XML format
- Eclipse Modeling Framework (EMF)

Target environment

- Java 6
- Eclipse Platform 3.6, Eclipse C/C++ Development Tooling(CDT)

Design

- Comprehensive usage of Eclipse extension points and API
- Two independent tracks: the first for source code annotations and the second for coverage statistics view

Results

The screenshot displays the Eclipse IDE interface with the following components:

- Project Explorer:** Shows the project structure for 'SampleCDTProject', including folders for 'coverage', 'Debug', and 'src', and files like 'calc1.cpp', 'calc2.c', 'calc1', 'calc2', 'Makefile', 'NsUriFixer.class', and 'README'.
- Source Editor:** Displays the code in 'calc2.c'. A popup for line 14 shows 'if (p == 0 && ...' with a 'Press F2 to focus' message.
- Coverage Browser:** A table at the bottom showing coverage statistics for 'calc2.c' and its functions.

Name	Function coverage	Uncovered (total func...	C/D coverage
src	75%	3 (12)	47%
calc1.cpp	75%	3 (12)	47%
calc1	80%	1 (5)	47%
calc2	71%	2 (7)	48%
eval(char*, unsigne...	100%	0 (1)	50%
lex(void)	100%	0 (1)	19%
main()	100%	0 (1)	60%
match(int)	0%	1 (1)	0%
name(int)	0%	1 (1)	0%

Column with coverage information is attached to the usual CDT editor. Popup shows coverage for multiple annotations on the same line without changing source code.

Coverage statistics view shows function and C/D coverage on C/C++ sources organized as a tree.

Experience

After 6 weeks of development, the first prototype was put in production.

- Short iterations enable fast adoption of new requirements and more frequent deliveries
- Good software architecture creates loosely coupled and easily extensible applications despite its big size
- Industrial frameworks(in our case EMF) are extremely powerful and robust
- Access to Eclipse source code flattens learning curve and simplifies development

Conclusion

- All research questions have been answered
- The BullseyeCoverage plug-in was put into production
- Issues pertaining to practical aspects of code coverage were identified
- Directions for further work were suggested

Thank you for attention!
It is discussion time now.