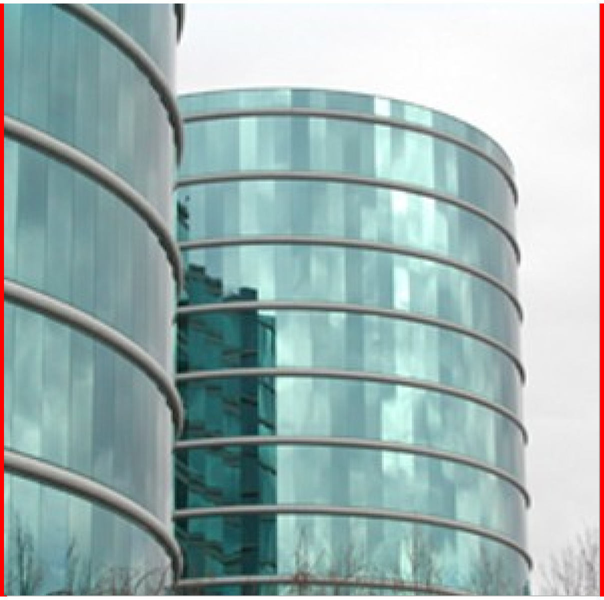


Полезные метрики покрытия

Практический опыт и немного теории

Александр (Шура) Ильин
JavaSE quality architect

What's this NOT about?

- Not about a particular code coverage tool
  - Different tools for different needs
- Not about hoe to write tests to reach higher code coverage
  - There are different techniques
- No data for the projects I am working on
  - Sorry.

What's this about?

- Personal experience
  - My
  - Yours (depends on speaker abilities to speak fast)
- Using coverage as a metric
- What metrics to select

Disclaimer

- The views expressed on this presentation belong to the presenter and do not necessarily reflect the views of Oracle
- Докладчик работает в “продуктовой лавке” :)
- Ideas, approaches and conclusions described are coming from experience of the presenter

What's a metric

- A type of measurement user to gauge some quantifiable components.

What's a metric

- A type of measurement user to gauge some quantifiable components.
- Metrics also used to track progress toward a goal.

What's a metric

- A type of measurement user to gauge some quantifiable components.
- Metrics also used to track progress toward a goal.
- A goal must be SMART
  - **Simple** (who, what, where, when, which, why)
  - **Measurable** (has a metric)
  - **Attainable** (the metric has a target value)
  - Realistic (willing and able to work towards)
  - **Timely** (results arrive on time to be used)

How do you test the application?

How do you test the application?

By tests

How do you test the application?

By tests

Metrics

- Number of tests
- Passrate
- Number of bugs
- Etc, etc, etc

How do you test the tests?

How do you test the tests?

Hm ….

How do you test the tests?

By … application?

What's the goal of testing?

- Prove the system working properly

Practically impossible

- Prove the system not working properly

Never ending process

- Verify conformance to requirements

Too generic

What's the goal of testing?

- Prove the system working properly

Practically impossible

- Prove the system not working properly

Never ending process

- Verify conformance to requirements

Too generic

- Money (not to loose money on poor quality)

Yep. I need my bills to be paid.

- Report quality

On time and accurately. Still, too abstract.

What's a quality?

Separate presentation.

No holy war today.

Two goals. *

- Defect escape reduction
- Test coverage (different kinds off)

- All the rest is bullshit *
  - Misleading
  - Not to the point
  - No target

(*) Honest opinion of the presenter

Two goals. *

- Defect escape reduction
- Test coverage (different kinds off)

- All the rest is rubbish *
  - Misleading
  - Not to the point
  - No target

(*) Honest opinion of the presenter

What's a bug escape metric?

Ratio of defects sneaked out unnoticed

In theory: $$\frac{\text{\# defects not found before release}}{\text{\# defects in the product}}$$

Practical: $$\frac{\text{\# defects found after release}}{\text{\# defects found after + \# defects found before}}$$

ORACLE

What's a bug escape metric?

Ratio of defects sneaked out unnoticed

$$\frac{\text{\# defects found after release}}{\text{\# defects found after} + \text{\# defects found before}}$$

Perfect! Direct metric of quality of testing

What's a bug escape metric?

Ratio of defects sneaked out unnoticed

$$\frac{\text{\# defects found after release}}{\text{\# defects found after + \# defects found before}}$$

Perfect! Direct metric of quality of testing

Problem: coming too late!

Is defect escape a good metric?

- **Simple** (+.)
- **Measurable (+)**
- **Attainable (?)**
- Realistic
- **Timely (-!)**

What's test coverage?

A ratio of some numerable items “covered” in testing.

# tested / # existing

Implementation coverage

- Code coverage
  - Line, block
  - Entry/exit
  - Branch/predicate
  - Path/sequence
- Race coverage
- Data coverage

Specification coverage

- Requirement
- Assertion
- API
- UI
- Data coverage

Assumes test logic is right

Code coverage

Shows to what extent source code is covered

- Class
- Method (function)
- Block
- Line
- Branch (condition)/predicate
- Path (sequence)
- Entry/exit

Code coverage analysis – what for?

A) Improve test base

1. Measure coverage
2. Find uncovered code
3. Develop more tests
4. Goto 1.

B) Find dead code

1. Find something uncovered
2. Try to develop tests
3. Fail
4. Go to developers
5. Argue
6. Argue
7. Argue
8. Create a bug
9. Dead code removed

C) Use as a metric

Is code coverage a good metric?

so far ...

- **Simple** (+)
- **Measurable** (+)
- **Attainable** (?)
- Realistic
- **Timely** (+!)

Block code coverage – what's the target?

100%? Unrealistic.

80%? Why not 78?

50%? What does it guarantee?

100% block/line coverage

- Does it mean that all bugs found?

Nope.

100% block/line coverage

- Does it mean that all bugs found?

Nope.

- Does it mean that all scenarios tested?

Nope.

100% block/line coverage

- Does it mean that all bugs found?

Nope.

- Does it mean that all scenarios tested?

Nope.

- What does it mean, then?

Nothing, really.

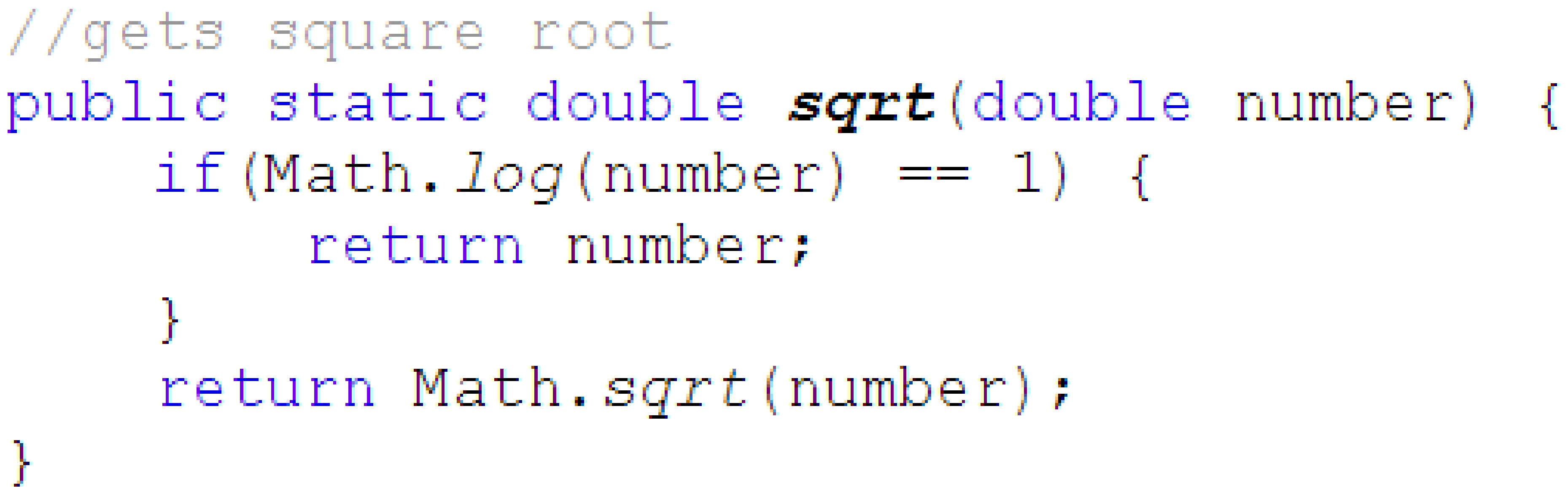
100% block/line coverage

```
//checks whether the value is zero
public static boolean isZero(float number) {
    boolean result = false;
    if(number != 0) {
        result = false;
    }
    return result;
}
```

Branch coverage helps?

```
//checks whether the value is positive
public static boolean isPositive(float number) {
    if(number >= .1) {
        return true;
    }
    return false;
}
```

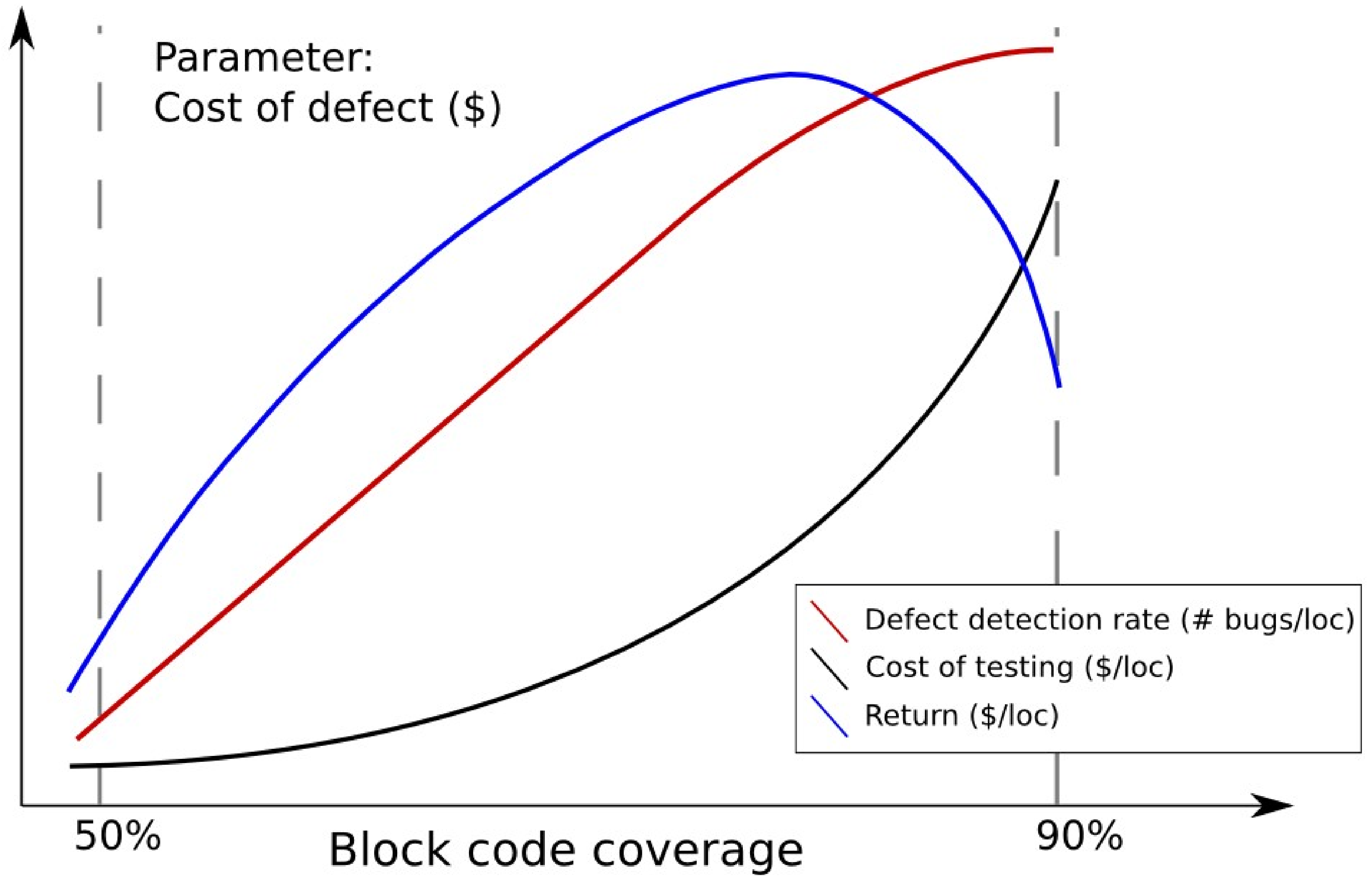
Data coverage?



```
//gets square root
public static double sqrt(double number) {
    if(Math.log(number) == 1) {
        return number;
    }
    return Math.sqrt(number);
}
```

Block code coverage – what's the target?

... target comes from ROI



Is block/line coverage a good metric?

- **Simple** (+)
- **Measurable (+)**
- **Attainable (-)**
- Realistic
- **Timely** (+!)

Code coverage is very useful.

- Tie CC data to
  - Code complexity
    - More complicated code is – more thorough testing it needs
  - Code stability
    - Recently changed code needs to be tested better
  - Bug density
    - Is there a correlation
  - Static analysis tools
    - To filter the false positives
- Use CC as
  - Check-in criteria
    - Commit only if tested
  - Test selection
    - Run only tests for changed code
  - Hot spot detection
    - Usually coverage also shows hit counts

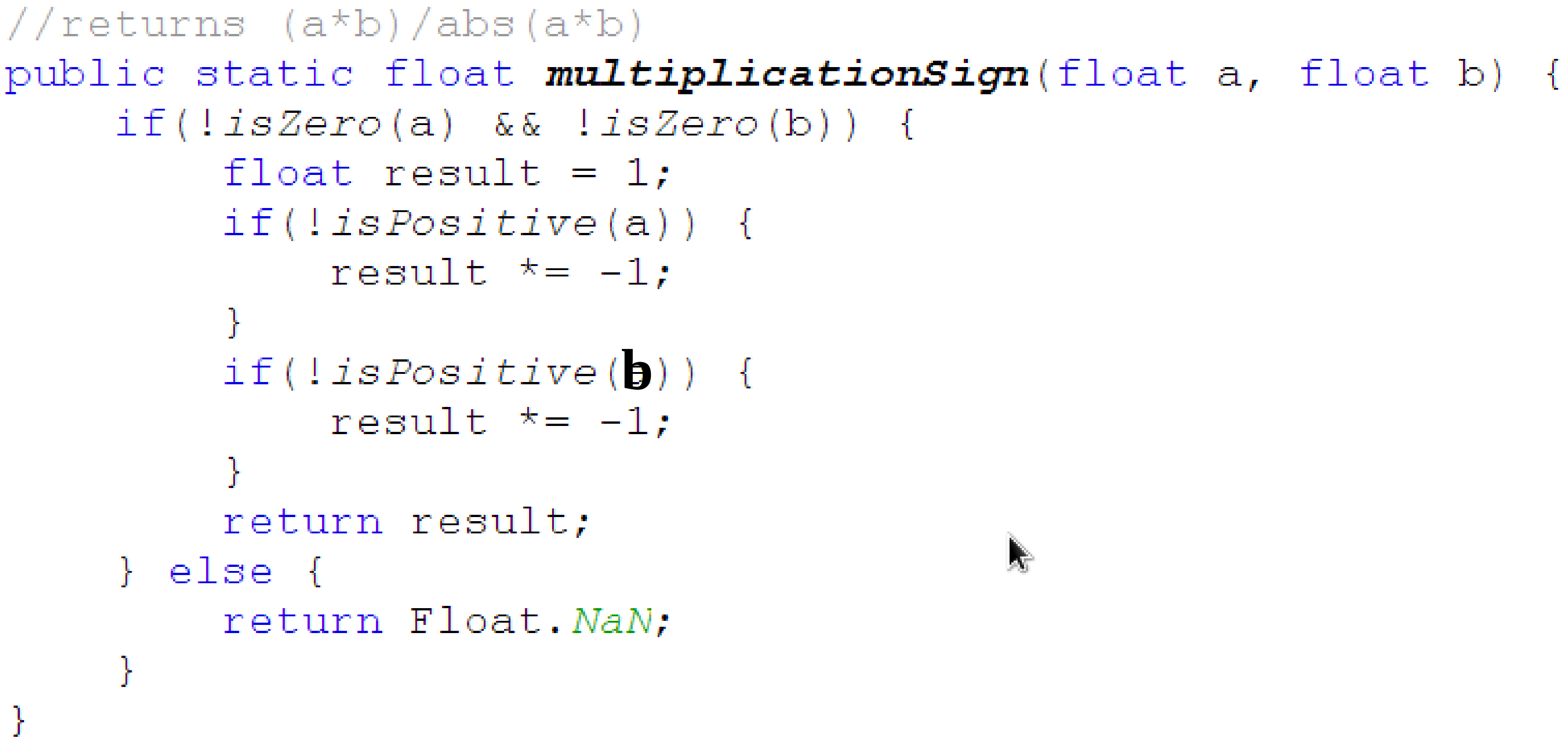
What about 100% sequence coverage?

Does it guarantee full testing?

100% sequence coverage

```
//returns (a*b)/abs(a*b)
public static float multiplicationSign(float a, float b) {
    if(!isZero(a) && !isZero(b)) {
        float result = 1;
        if(!isPositive(a)) {
            result *= -1;
        }
        if(!isPositive(a)) {
            result *= -1;
        }
        return result;
    } else {
        return Float.NaN;
    }
}
```

100% sequence coverage



```
//returns (a*b)/abs(a*b)
public static float multiplicationSign(float a, float b) {
    if(!isZero(a) && !isZero(b)) {
        float result = 1;
        if(!isPositive(a)) {
            result *= -1;
        }
        if(!isPositive(b)) {
            result *= -1;
        }
        return result;
    } else {
        return Float.NaN;
    }
}
```

What about sequence coverage?

- 100% sequence coverage does not prove a code to be fully tested
  - Libraries may have defects

100% sequence coverage

```
//checks whether the value is positive
public static boolean isPositive(float number) {
    if(number >= .1) {
        return true;
    }
    return false;
}
```

What about sequence coverage?

- 100% sequence coverage does not prove a code to be fully tested
  - Libraries may have defects
  - Branches could be incorrect
- 100% sequence + 100% predicate + 100% data +100% XYZ?
  - No idea
  - Have not seen a mathematical proof yet
  - Does not matter

What else wrong with sequence coverage?

- Tools availability
  - None for Java
- Expensive
  - Execution
  - Analysis
- Complicated
  - Not all paths are valid
  - Loops
- 100% likely not attainable
  - number of paths is exponential to the number of branches

Is sequence coverage a good metric?

- **Simple** (+.)
- **Measurable** (+)
- **Attainable** (-.)
- Realistic
- **Timely** (+)

Coverage data is not free

- Do just as much as you can consume *
  - Requires infrastructure work
  - Requires some development
  - Requires some analysis

(*) The rule of thumb

Coverage data is not free

- ~~Do just as much as you can consume~~
  - Requires infrastructure work
  - Requires some development
  - Requires some analysis
- Do just a little bit more than you can consume *
  - Otherwise how do you know how much you can consume?

(*) The rule of thumb

So, is there a good coverage metric, then?

Attainability seems to be the most trouble so far

- Manageable amount of work
  - Obtain
  - Analyze
  - Improve
- Clear target
  - Explainable
  - Verifiable

Specification coverage

- Requirements
- Assertions
- Public API
- UI
- Custom coverage metrics

Assertions

public final void wait(long timeout, int nanos) throws InterruptedException

Causes current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

Assertions

- Mark all the assertions with ID
  - Mark up the specification itself
  - Create a separate storage
- Develop tests
  - An assertion deserves a few
  - A test could cover a few
- Mark tests by assertion IDs
- When next specification version is released
  - Compare assertions with a previous version
  - Verify all tests for changes assertions
  - Scan for new assertions
  - Develop new tests

Requirements

Same as assertions

- Number the requirements
- Mark the tests
- Validate all tests when requirements change

Public API*

Is a set of program elements suggested for usage by public documentation.

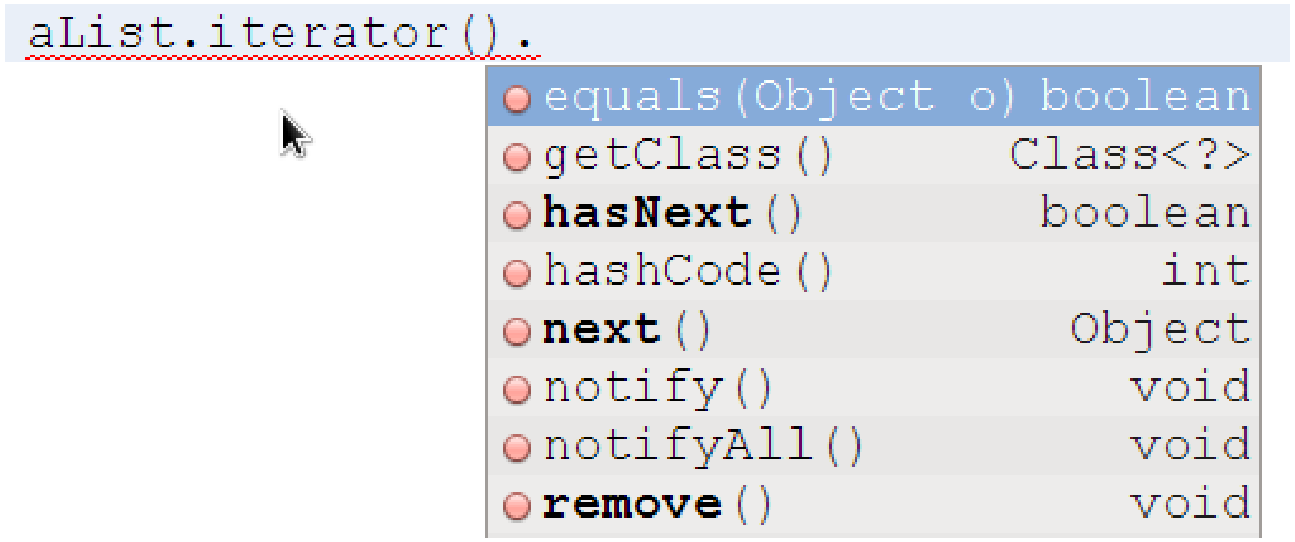
For example: all functions and variables which are described in documentation.

For a Java library: all public and protected methods and fields mentioned in the library javadoc.

For Java SDK: … of all public classes in **java** and **javax** packages.

(*) Only applicable for a library or a SDK

Public API



```
aList.iterator().
```

```
equals(Object o) boolean
getClass()       Class<?>
hasNext()         boolean
hashCode()            int
next()             Object
notify()             void
notifyAll()          void
remove()             void
```

True Public API (c)

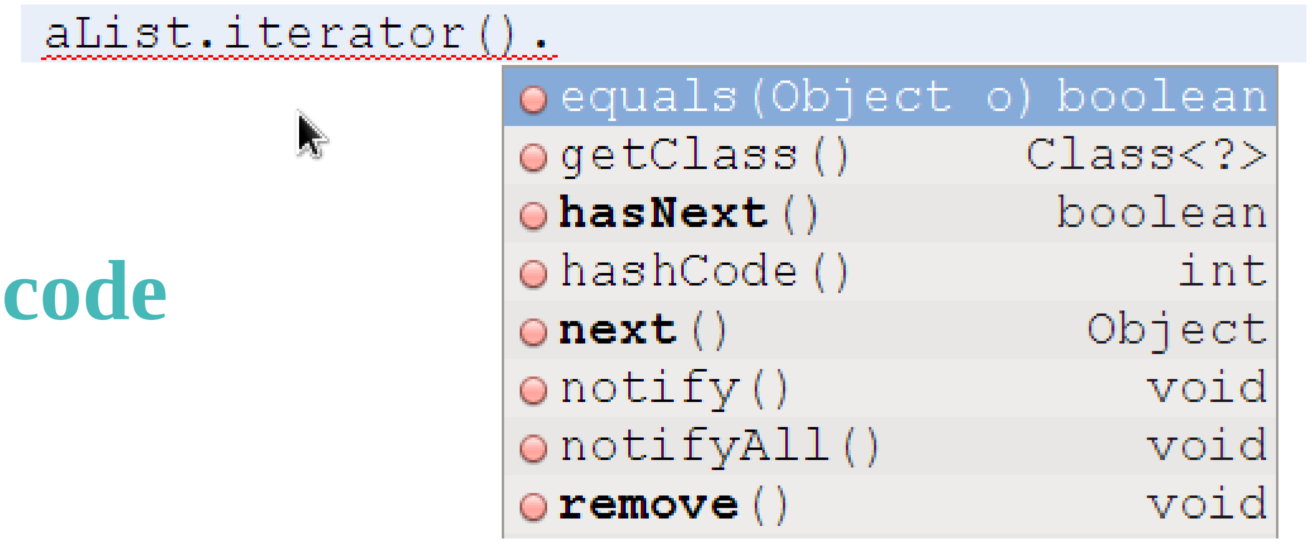
Is a set of program elements which could be accessed directly by a library user

Public API

+

all extensions of public API in non-public classes

True public API example



```
aList.iterator().
```

```
equals(Object o)   boolean
getClass()         Class<?>
hasNext()          boolean
hashCode()         int
next()             Object
notify()           void
notifyAll()        void
remove()           void
```

My code

```
public class ArrayList<E> implements List<E> {

    //...

    @Override
    public Iterator<E> iterator() {
        return new Itr();
    }

    private class Itr implements Iterator<E> {
```

ArrayList.java

(True) Public API

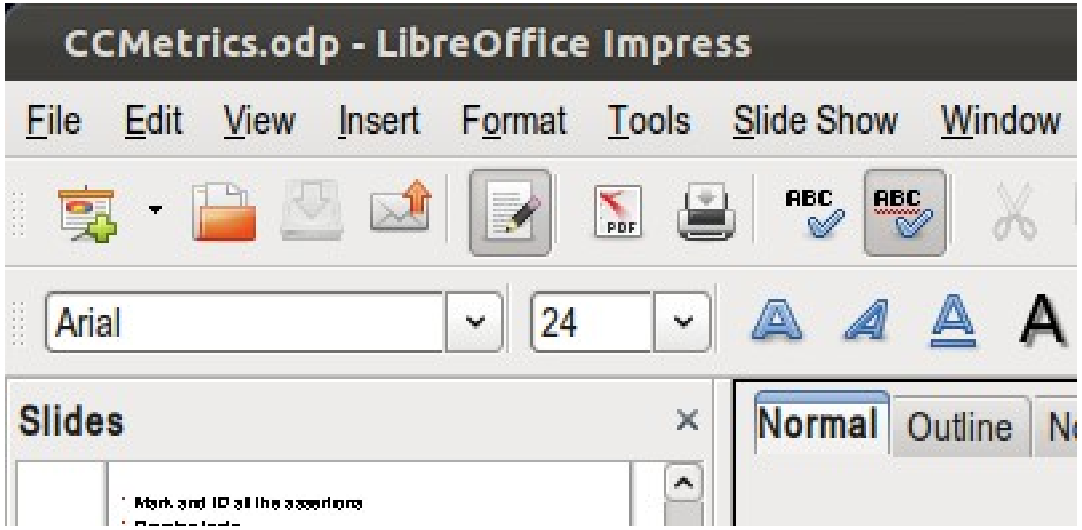
- Is obtainable from code coverage
- May require some work on top of existing CC tools

True Public API

- Does require some work on top of an existing CC tools
- Emma, JCov

UI coverage

- % dialogs inspected
- % menus pushed
- % buttons pushed
- % text fields was entered a value in
- etc.



ORACLE

UI coverage

- In some cases obtainable from code coverage (works well for factory methods)
  - Action coverage: **javax.swing.Action** implementations
- If not
  - “Instrument” the application
    - Adding test hooks
    - Using event queue
  - Obtain a “template” somehow else

Are these metrics attainable?

May be not in the beginning

Are these metrics attainable?

For every product to be claimed to be tested well, the target value is ...

Are these metrics attainable?

For every product to be claimed to be tested well, the target value is ...

100%

Why 100%?

- Every requirement must be tested by at least one test
- Every statement in spec must be exercised at least once
- Every method in public API must be called at least once
- Every button in a UI needs to be pushed at least once

For exceptions there are exclude lists.

So, are these metrics good?

- **Simple** (+)
- **Measurable (+)**
- **Attainable (+)**
- Realistic
- **Timely (+)**

But wait … what is the point?

- What does 100% public API coverage means?
- UI coverage?
- Specification coverage?
- Requirements coverage?

But wait … what is the point?

- What does 100% public API coverage means?

Nothing.

- UI coverage?
- Specification coverage?
- Requirements coverage?

But wait … what is the point?

- What does 100% public API coverage means?

Nothing.

- UI coverage?

Not that we are done with testing.

- Specification coverage?

- Requirements coverage?

But wait … what is the point?

- What does 100% public API coverage means?

Nothing.

- UI coverage?

Not that we are done with testing.

- Specification coverage?

Not anywhere near with been done.

- Requirements coverage?

But wait … what is the point?

- What does 100% public API coverage means?

Nothing.

- UI coverage?

Not that we are done with testing.

- Specification coverage?

Not anywhere near with been done.

- Requirements coverage?

Well … at least we touched each and every of'm

But wait … what is the point?

- What does 100% public API coverage means?

Nothing.

- UI coverage?

Not that we are done with testing.

- Specification coverage?

Not anywhere near with been done.

- Requirements coverage?

Well … at least we touched each and every of'm

These metrics are **required but not sufficient**.

More such metrics

- Property coverage
  - Property is a field with a setter and getter
- UML element coverage
  - In UML diagram
- Language element coverage
  - Expression Language

- Make your own.

Summary

No perfect coverage metric

- Implementation based
  - target value comes from ROI
  - too heavy
  - yet do not prove anything
- Specification based
  - Are not thorough enough

Summary

On the bright side, any of them could be useful

- Test base improvement
- Finding
  - Dead code
  - Untestable assertions
  - Unreachable UI
  - Etc.

Comments?

Alexandre (Shura) Iline

Oracle

Quality architect

Java SE SQE group

alexandre.iline@oracle.com