

Software Testing

Lecture 5 – Static Analysis

A diagnostic quiz

- How much of this lecture's topic do you already know?
- We'll repeat at the end of the session

- Answer the quiz
- Work alone

5 minutes

The diagnostic quiz

1. What *is* “static analysis”?
2. What’s the relationship, in terms of practical value, between static analysis and dynamic testing?
3. What’s the most commonly used form of static analysis?
4. What can code quality analysis tools (e.g. Lint, FindBugs, FXCop etc) do for you as a developer?
5. Contract verification (as supported by SPARK and Eiffel) supports very powerful and accurate analysis. What are the main drawbacks to using it?
6. What’s the biggest barrier, in practice, that real-world projects face when they think about implementing more advanced static analysis?

So, you've written down your answers...

- ...so I'll give you mine
- (but in the form of a set of lectures)

Definition (*from lecture 1*)

Dynamic Testing

Testing code by executing it.

Definition (*from lecture 1*)

Static Analysis

Testing code without executing it.

What are some advantages of dynamic testing?

- Don't need special tools
 - JUnit etc are optional
- Don't need many special skills to get started
 - And lots of people know some of them
- Don't necessarily need the source code
 - If you can run it, you can test it
 - (although maybe not in the way you most want to)
- Testers don't need to be programmers

What are some (disadvantages of / common problems encountered when doing) dynamic testing?

- Achieving complete coverage can be really hard (even impossible)
- Some failures require really bizarre combinations of inputs to trigger
 - Sometimes even nastier dependencies e.g. timing, network load
- When you find a failure on a larger-scale (e.g. system) test, often not clear what the low-level cause is

Upshot of problems – some faults are hard to find by dynamic testing

So, these lectures will look at some static analysis options:

1. Type checking
2. Code quality analysis
3. Contract verification

SECTION 1 – TYPE CHECKING

What happens if we run this Javascript?

```
function onlyWorksOnNumbers(x)
{
    return x * 10;
}
```

```
onlyWorksOnNumbers("Hello, world!");
```

=> NaN

What if we add type annotations, and check it with a type checker?

```
function onlyWorksOnNumbers(x: number)
{
    return x * 10;
}
```

```
onlyWorksOnNumbers("Hello, world!");
```

```
$> flow
```

```
hello.js:5:5,19: string
```

```
This type is incompatible with
```

```
hello.js:3:10,15: number
```

What are some other examples of type rules?

- Java
 - Array index must be an integer
 - Can't put a Java Double into a Float or Int (a "narrowing conversion")
 - Can't put an Animal object into a Cat variable (a "downcast")

Java's typing can't catch everything

```
private static void printStringLength(String s)  
{  
    System.out.println("String " + s + " is " + s.length() + " characters  
    long");  
}
```

```
public static void main(String[] args) {  
    printStringLength("Hello");  
}
```

String Hello is 5 characters long

Java's typing can't catch everything (2)

```
private static void printStringLength(String s)
{
    System.out.println("String " + s + " is " + s.length() + " characters long");
}

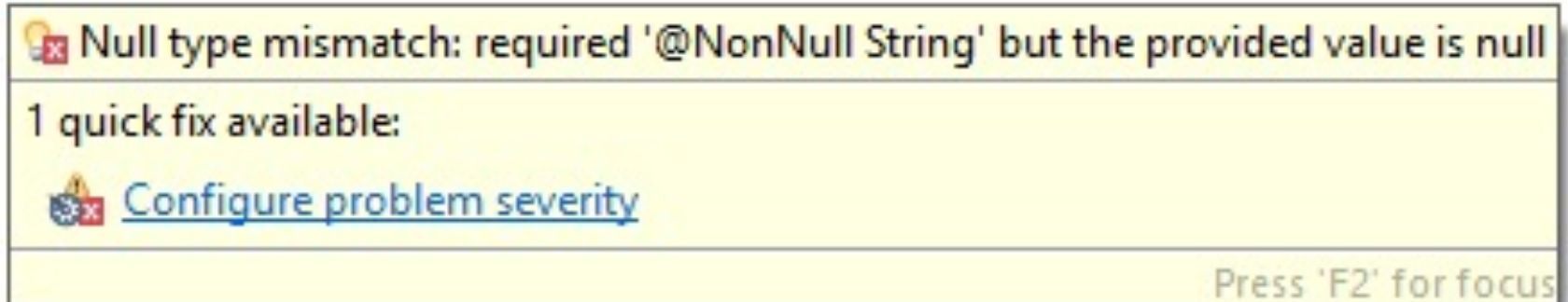
public static void main(String[] args) {
    printStringLength(null);
}
```

```
Exception in thread "main"
java.lang.NullPointerException
at BadNull.printStringLength(BadNull.java:6)
at BadNull.main(BadNull.java:11)
```


We can strengthen our type system

```
private static void printStringLength(@NonNull String s)
{
    System.out.println("String " + s + " is " + s.length() + "
    characters long");
}
```

```
public static void main(String[] args) {
    printStringLength(null);
}
```



Java's typing can't catch everything – example 2

```
private static String[] strings =  
    new String[] {"Zero", "One", "Two"};  
  
private static void printString(int index)  
{  
    System.out.println(strings[index]);  
}  
  
public static void main(String[] args) {  
    printString(1);  
}
```

One

Java's typing can't catch everything – example 2 (2)

```
private static String[] strings =  
    new String[] {"Zero", "One", "Two"};  
  
private static void printString(int index)  
{  
    System.out.println(strings[index]);  
}  
  
public static void main(String[] args) {  
    printString(3);  
}
```

Exception in thread "main"

java.lang.ArrayIndexOutOfBoundsException: 3
at BadIndex.printString(BadIndex.java:7)
at BadIndex.main(BadIndex.java:12)

Again, we can strengthen our type system

```
private static String[] strings =  
    new String[] {"Zero", "One", "Two"};  
  
private static void printString(@intRange(0,2) int index)  
{  
    System.out.println(strings[index]);  
}  
  
public static void main(String[] args) {  
    printString(3);  
}
```

Unresolved compilation problem:

The method printString(@intRange(0,2) int index)
in the type BadIndex is not applicable for the arguments (int=3)

at BadNull.main([BadNull.java:10](#))

(and might be able to streamline that further)

```
private static final String[] strings =  
    new String[] {"One", "Two", "Three"};  
  
private static void printString(@intRangeOfArray(strings) int index)  
{  
    System.out.println(strings[index]);  
}  
  
public static void main(String[] args) {  
    printString(3);  
}
```

Unresolved compilation problem:

The method printString(@intRange(0,2) int index)
in the type BadIndex is not applicable for the arguments (int=3)

at BadNull.main([BadNull.java:10](#))

Definition

Static Type Checking

Checking source code prior to run-time to determine whether the type rules in force (from the language or from other tools) have been respected.

So do all modern languages use strong compile-time (static) type checking?

Some do

- Java
- C++
- C#
- Rust
- Go
- SPARK
- Haskell

Some don't

- Javascript
- Python
- Lua
- Objective-C

When good programmers have static type checking available, do they always use the strongest form of it?

- E.g. do good Java programmers always use extended type annotations (e.g. as supported by FindBugs)?
- No, they often don't

How far can we take type checking?

- Predicate types?
 - $\text{even?}: [\text{int} \rightarrow \text{bool}] = \{i:\text{int} \mid \exists(j:\text{int}): i = 2 \times j\}$
- Requires you to do proofs in order to do type checking.
 - Is *even?* non-empty?
 - If I write *even?(x)* I need to prove that
$$\exists(j:\text{int}): x = 2 \times j$$
 - Also think about subtyping; is the set of even numbers that are larger than 100 a subset of *even?* Proof?
- Might be too expensive for all but the most safety critical systems.

Class exercise

- Answer these questions:
 - Why don't all modern languages use strong static typing?
 - Why do good programmers not always use the strongest typing available for their language?
 - When can strong static typing help us most?
- Take about 3-5 minutes on this

Why don't all modern languages use strong static typing?

- Many programmers prefer the *feel* of languages without static typing
 - E.g. see <http://martinfowler.com/bliki/DynamicTyping.html>
- Risk of false positives for faults
 - i.e. type system causes error report for otherwise fault-free program
- Difficulties of understanding complex type errors
 - *"But why is it the wrong type? I don't see the problem."*
- Evidence of real-world benefits of static typing is actually inconclusive
 - E.g. see discussion at <http://lambda-the-ultimate.org/node/4494> of (static typing) vs (dynamic typing + good unit testing)

Why do good programmers not always use the strongest typing available for their language?

- (all the reasons from the previous slide)
- Generally, the strongest typing using all available tools is very strong indeed
 - E.g. Java has FindBugs type annotations, JML, Xtend
 - Also see the discussion earlier about predicate typing and subtyping
- Learning curve for these complex type systems

When can strong static typing help us most?

- When we (the whole team) really understand the language/type system being used
- Systems that will be used a lot but won't change much
 - e.g. core library components (see EiffelBase)
- Systems that need very high integrity (very low defect rates)
 - E.g. safety-critical avionics

- Any questions?

SECTION 2 – CODE QUALITY ANALYSIS

Definition

Code Quality Analysis (CQA)

Automated checking of source code for patterns that are often (but not always) indicators of errors.

What's the most common form of CQA?

- Compiler warnings

```
private boolean a;  
  
public void doSomething(boolean b)  
{  
    if(a = b)  
    {  
        //do something  
    }  
    ...  
}
```

Warning: Possible accidental assignment in place of a comparison. A condition expression should not be reduced to an assignment


```
if (launchApproved);  
    launchRocket()
```

Warning: Empty control-flow statement

```
java.util.Date myDate = new  
java.util.Date();  
    int currentDay = myDate.getDay();
```

Warning: The method `getDay()` from the type `Date` is deprecated

```
void foo() throws IOException {  
    FileReader reader = new FileReader("file");
```

 Resource leak: 'reader' is never closed

2 quick fixes available:

@ [Add @SuppressWarnings 'resource' to 'reader'](#)

@ [Add @SuppressWarnings 'resource' to 'foo\(\)'](#)

Press 'F2' for focus

```
    int ch;  
    while ((ch = reader.read()) != -1) {  
        System.out.println(ch);  
        reader.read();  
    }  
}
```

Sometimes, warnings are false alarms

```
private boolean a;
```

```
public void doSomething(boolean b)
{
    if(a = b)
    {
        //do something
    }
    ...
}
```

Sometimes, warnings are false alarms

```
private boolean a;  
  
public void doSomething(boolean b)  
{  
    a = b;  
  
    if(a)  
    {  
        //do something  
    }  
    ...  
}
```

Compilers don't implement every possible warning

- Why not?
- Because compilers are complex enough already
- Because compiler runtimes need to be fast, and *predictable*

Luckily, there are lots of third-party CQA tools

- Grandparent of all this is Lint
<http://www.gimpel.com/html/index.htm>
- Main open-source one for Java is FindBugs
- <http://findbugs.sourceforge.net/>
- Also FXCop for .Net, PyLint for Python...

What can FindBugs detect?

See

<http://findbugs.sourceforge.net/bugDescriptions.html>

**CNT: Rough value of known constant found
(CNT_ROUGH_CONSTANT_VALUE)**

e.g. you used a literal of 3.1415 for Pi

**Eq: equals method fails for subtypes
(EQ_GETCLASS_AND_CLASS_CONSTANT)**

E.g. if you write `Foo.class == o.getClass()` – only valid when lhs is of type Foo, not any subclass of Foo

**NP: toString method may return null
(NP_TOSTRING_COULD_RETURN_NULL)**

Officially that's allowed, but it's bad practice because most programmers won't expect it

Nm: Confusing method names (NM_CONFUSING)

"The referenced methods have names that differ only by capitalization."

**IL: A collection is added to itself
(IL_CONTAINER_ADDED_TO_ITSELF)**

Does that ever make sense?

**m: Class defines equal(Object); should it be
equals(Object)? (NM_BAD_EQUAL)**

"This class defines a method equal(Object). This method does not override the equals(Object) method in java.lang.Object, which is probably what was intended."

CQA tools can generate a lot of output

- An existing codebase may have huge numbers of things that they can warn about
- Not all of those will be real faults
- ... though it looks like newer tools are getting better at this
 - E.g. FindBugs developers aims for at least 50% of warnings being something that the user will want to fix

CQA tools can often be usefully tuned

- E.g. see PC-Lint tutorials here
 - http://www.riverblade.co.uk/downloads/slides/taming_the_lint_monster_slides.pdf
 - <http://www.riverblade.co.uk/downloads/articles/Taming%20the%20Lint%20Monster%20pt%201.pdf>
 - <http://www.riverblade.co.uk/downloads/articles/Taming%20the%20Lint%20Monster%20pt%202.pdf>

Sometimes even after tuning the output is still huge

- It may be that you can't fix the problem for this codebase
- May need to give up, and only use CQA on future codebases that you build to be clean from the start

Are CQA tools worthwhile?

- They can be, if you use them well
- Google's experiences with Findbugs
 - <http://dx.doi.org/10.1145/1831708.1831738>
 - *"hundreds of engineers, thousands of warnings"*
- *"77% of the reviews identified the warnings as real defects"*
 - ... but none of them were associated with serious misbehaviours in their current production code
 - Though did find serious faults in not-yet-deployed code

Is FindBugs a CQA tool... or a type-checking tool... or what?

- It's both, and probably other things too
- I'm giving you hard category distinctions to help make sense of things, but the reality is messier

Class exercise

- Answer these questions:
 - Why do most programmers *not* use third-party CQA tools?
 - Is there a time when a CQA tool might have saved you a lot of effort and stress?
 - Could you usefully apply a CQA tools to something you're working on at the moment (or will be soon)?
- Take about 3-5 minutes on this

Why do most programmers not use third-party CQA tools?

- One more thing to install, configure etc
- Can generate a lot of "noise" that you need to investigate
 - It may take a while to get your code basically clean
- Experiences like Google's — mildly positive, but not a huge benefit

- Any questions?

SECTION 3 – CONTRACT VERIFICATION

We can say a lot using type systems...

- ... but any type system will fail to catch some faults

```
private static Stack<String> strings = new Stack<>();
```

```
private static void populateStack()  
{  
    //do nothing  
}
```

```
private static String popString()  
{  
    return strings.pop();  
}
```

```
public static void main(String[] args) {  
    populateStack();  
    popString();  
}
```

```
Exception in thread "main" java.util.EmptyStackException  
    at java.util.Stack.peek(Unknown Source)  
    at java.util.Stack.pop(Unknown Source)  
    at ContractBase.popString(ContractBase.java:10)  
    at ContractBase.main(ContractBase.java:14)
```

```

private static Stack<String> strings = new Stack<>();

/*@ ensures !strings.isEmpty()
   @*/
private static void populateStack()
{
    //do nothing - I've forgotten to implement this
}

/*@ requires !strings.isEmpty()
   @*/
private static String popString()
{
    return strings.pop();
}

public static void main(String[] args) {
    populateStack();
    popString();
}

```

Verification failure: *populateStack()* doesn't ensure *!strings.isEmpty()*

That's using special annotations and tools

- Yes, but retrofitted onto standard Java (it's JML)
- SPARK fulfils a similar role for ADA
 - Check with SPARK tools, compile with a standard compiler
- A little bit on JML, then a bit more on SPARK, Ada, and restrictions to make it all tractable

Java Modeling Language (JML)

- JML is a behavioral interface specification language
- JML is a specification language that allows specification of the behavior of an API
 - not just its syntax, but its semantics
- JML specifications are written as annotations
 - As far as Java compiler is concerned they are comments but a JML compiler can interpret them

JML

- **Goal:** Make writing specifications easily understandable and usable by Java programmers,
 - so it stays close to the Java syntax and semantics
- JML supports design by contract style specifications with
 - Pre-conditions
 - Post-conditions
 - (Class) invariants

JMLAnnotations

- JML annotations are added as comments to the Java source code
 - either between `/*@ . . . @*/`
 - or after `//@`
 - These are ***annotations*** and they are ignored by the Java compiler
- JML properties are specified as Java boolean expressions
 - JML provides operators to support design by contract style specifications such as `\old` and `\result`
 - JML also provides quantification operators (`\forall`, `\exists`)
- JML also has additional keywords such as
 - `requires`, `ensures`, `signals`, `assignable`, `pure`, `invariant`, `non null`, ...

JML contracts

- Preconditions (REQUIRES) are written as a `requires` clauses
- Postconditions(EFFECTS) are written as `ensures` clauses
- MODIFIES are written as `modifiable` clauses
- Rep invariants are written as `invariants` clauses

Simple Example

- Consider the spec. of a swap routine in Java

```
public static void swap(int [] a) {  
    /* @ requires a.length == 2  
       @ modifiable a  
       @ ensures ( (a[0]==\old(a[1]) &&  
                   @      (a[1]==\old(a[0])) )  
    */  
}
```

Simple Example: Exception

- Consider the spec. of a swap routine in Java
- But assume that it throws an exception when given an array of length $\neq 2$.

```
public static void swap(int [] a) {  
    /* @ modifiable a  
    @ ensures ( (a[0]==\old(a[1]) &&  
    @          (a[1]==\old(a[0])) )  
    @ signals NullPointerException (a == NULL)  
    @ signals LengthException (a.length != 2)  
    */
```

JML Quantifiers

- JML supports several forms of quantifiers
 - Universal and existential (`\forall` and `\exists`)
 - General quantifiers (`\sum`, `\product`, `\min`, `\max`)
 - Numeric quantifier (`\num_of`)

```
(\forall Student s; class272.contains(s);  
    s.getProject() != null)
```

```
(\forall Student s; class272.contains(s)  
    ==> s.getProject() != null)
```

- Without quantifiers, we would need to write loops to specify these types of constraints

JML Quantifiers (cont)

- Quantifier expressions
 - Start with a declaration that is local to the quantifier expression

```
(\forall Student s; ...
```
 - Followed by an optional range predicate

```
... class272.contains(s); ...
```
 - Followed by the body of the quantifier

```
... s.getProject() != null)
```

Using JML (or contracts)

- To provide better error messages during testing.
- To help generate test cases.
 - Requires/ensures clauses can act as specifications for tests.
- (Theoretically): to reduce the amount of testing you need to do.
 - Do you believe this?

There are often a lot of restrictions

- SPARK 2005 prohibits:
 - Dynamic memory allocation
 - Pointers
 - Or any other source of aliasing
 - Recursion
 - Expressions with side effects
 - E.g. there's no equivalent of this C code:

```
while (i++ < 5) {  
    printf("Count is %d\n", i);  
}
```

Spark Ada 2005

```
package Odometer
--# own Trip, Total : Integer;
is
    procedure Zero_Trip;
    --# global out Trip;
    --# derives Trip from ;
    --# post Trip = 0;

    function Read_Trip return Integer;
    --# global in Trip;

    function Read_Total return Integer;
    --# global in Total;

    procedure Inc;
    --# global in out Trip, Total;
    --# derives Trip from Trip & Total from Total;
    --# post Trip = Trip~ + 1 and Total = Total~ + 1;

end Odometer;
```

CodePeer

- <https://www.adacore.com/codepeer>

Isn't this is too complicated, too restrictive, and too deforming to your code, to ever use in real projects?

- Yeah, it usually is
- And it uses lots of skills that most programmers don't have

But sometimes you need ultra-high confidence of correctness

- Avionics (flight control)
 - C130J mission computer
- Very high security
 - E.g. Tokeneer
 - <http://www.adacore.com/sparkpro/tokeneer/>
 - Common Criteria EAL5
- Air traffic control
- Nuclear plant control?

- Any questions?

SECTION 4 – COMMONALITIES

What do these three approaches have in common?

- Answer the above question
 - (you may come up with multiple answers)
- Take about 5 minutes on this

A commonality

- All three types of SA, here, provide a language...
 - Type specifications/annotations
 - CQA program annotations, and CQA tool config
 - Contracts
- ...which can express powerful assertions which can be convincing checked (automatically, or mostly so)...
- ...given
 - Suitable special-purpose tools
 - A suitably skilled analyst

- Any questions?

SECTION 5 – THAT QUIZ AGAIN

The diagnostic quiz - repeated

- Why have I asked you to do this test again?
- Reminder – quizzes like this give a big increase in long-term recall

- Answer the quiz again
- Work alone

5 minutes

The diagnostic quiz – my answers

1 What *is* “static analysis”?

From lecture 1 – “Testing code without executing it”

2 What’s the relationship, in terms of practical value, between static analysis and dynamic testing?

They’re complimentary – they tend to find different kinds of faults.

3 What’s the most commonly used form of static analysis?

Type checking, typically by a compiler or IDE.

The diagnostic quiz – my answers (2)

4 What can code quality analysis tools (e.g. Lint, FindBugs, FXCop etc) do for you as a developer?

They can find patterns in your code that are often, but not always, signs that there is a fault.

5 Contract verification (as supported by SPARK and Eiffel) supports very powerful and accurate analysis. What are the main drawbacks to using it?

It places severe constraints on the language constructs and programming idioms you can use. You have to write your program to support it.

6 What's the biggest barrier, in practice, that real-world projects face when they think about implementing more advanced static analysis?

Skills and expertise. Using advanced SA tools involves specific skills with them, and time spent working on that isn't getting the product out the door.

- Any questions?

Now

By email