Principles of Software Construction: Objects, Design, and Concurrency

\{Static & Dynamic\} x \{Typing & Analysis\}

Christian Kästner    Vincent Hellendoorn
Quiz

https://rb.gy/ql0x0m
How Do You Find Bugs?

```java
private static int getValue(Integer i) {
    return i.intValue();
}
```
How Do You Find Bugs?

- Run it?

```java
public class Fails {
    public static void main(String[] args) {
        getValue(i: null);
    }

    private static int getValue(Integer i) {
        return i.intValue();
    }
}
```

Exception in thread "main" java.lang.NullPointerException
  at misc.Fails.getValue(Fails.java:9)
  at misc.Fails.main(Fails.java:5)
How Else Do You Find Bugs?

```java
public class Fails {
    public static void main(String[] args) {
        getValue(0); // null
    }

    private static int getValue(Integer i) {
        return i.intValue();
    }
}
```
Static Analysis

IntelliJ can look at this code and say:

```java
public static void main(String[] args) {
    getValue(i: null);
}

private static int getValue(Integer i) {
    return i.intValue();
}
```

Passing 'null' argument to parameter annotated as @NotNull
Static Analysis

How?

```java
public static void main(String[] args) {
    getValue(1: null);
}

private static int getValue(Integer i) {
    return i.intValue();
}
```

Passing 'null' argument to parameter annotated as @NotNull
Static Analysis

How?

- We know at compile time where getValue gets routed to
- getValue calls a method on i
- i can be null

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

private static int getValue(Integer i) {
    return i.intValue();
}
```

Passing 'null' argument to parameter annotated as @NotNull
Static Analysis

How about JS?

```javascript
function getValue(x) {
    return x.valueOf();
}
```
Static Analysis

Run it: ✓

```
function getValue(x) {
    return x.valueOf();
}

console.log(getValue("32"));
console.log(getValue(null));
```

PROBLEMS 3 OUTPUT TERMINAL DEBUG CONSOLE

```
return x.valueOf();
^  
TypeError: Cannot read property 'valueOf' of null
```
Static Analysis

Why no warning?

```javascript
function getValue(x) {
    return x.valueOf();
}

console.log(getValue("32"));
console.log(getValue(null));
```
Another Java vs JS Example

class Foo {
    constructor(x) {
        this.x = x;
    }
}

function bar(foo) {
    return foo.x;
}

var foo = new Foo(3);
console.log(bar(foo));
console.log(bar(3));

public static void main(String[] args) {
    Foo foo = new Foo(x: 3);
    bar(foo);
    bar(foo: 3);
}

private static void bar(Foo foo) {
    System.out.println(foo.x);
}
Static vs. Dynamic Typing

- The more knowledge we inject in the code, the more bugs we can catch at compile time
  - Types, nullity annotations, invariants

- At compile-time:
  - Dynamically typed languages assume nothing
    - Types exist only for values
  - Static typing is not completely precise either
    - Objects have declared types and run-time types
    - Different “strength” type systems
Static vs. Dynamic Typing

- The more knowledge we inject in the code, the more bugs we can catch at compile time
  - Types, nullity annotations, invariants
- Is it worth it?
  - Dynamic typing can severely limit inference
  - But… static types are a lot of work
Static vs. Dynamic Typing

- The more knowledge we inject in the code, the more bugs we can catch at compile time
  - Types, nullity annotations, invariants
- Is it worth it?
  - Dynamic typing can severely limit inference
  - But… static types are a lot of work
Static vs. Dynamic Typing

Okay, but:

Top languages over the years

https://octoverse.github.com/#geographical-distribution-of-active-users
False Dichotomy?

Yes, but:

Top languages over the years
Partial Types

- Low effort, some utility
  - Static types exist and are checked at compile-time
  - Dynamic types are used at run-time
    - So annotations get ignored!
  - Type checker can be shallow or deep; TS is shallow
Types in TypeScript

```typescript
function getValue(x: number) {
    return x.valueOf();
}

console.log(getValue(null));
```

Argument of type 'null' is not assignable to parameter of type 'number'. ts(2345)

View Problem   No quick fixes available
Types in TypeScript

```javascript
function getValue(x: number | null) {
    return x.valueOf();
}

console.log(getValue(null));
```

Object is possibly 'null'. ts(2531)

(parameter) x: number | null

View Problem No quick fixes available
Step Back

- Why do we care about types so much?
Step Back

- Why do we care about types so much?
  - We care about *common mistakes*
  - Type errors happen to be very common
  - What else is common?
Step Back

Why do we care about types so much?
- We care about *common mistakes*
- Type errors happen to be very common
- What else is common?
  - Nullity errors
  - Missing imports
  ```java
  public void forward(String sender) {
      if (sender == "me") {
          sendSelf();
      } else if (sender == "other") {
  ```
Static Analysis

- Detect real or plausible bugs based on code patterns
  - Plausible: look for risk-prone areas
    - Deeply nested loops
    - Overly general types (e.g., ‘any’ in TS)
    - Dead code_UNUSED variables
    - Any other places we often make mistakes?
Static Analysis

- How?
  - Program analysis + Vocabulary of patterns

https://deepsource.io/blog/introduction-static-code-analysis/
Static Analysis

- **Step 1: Tokenization**
  - Tokens are like the words of software
  - *Lexical* categories, incl. punctuation, identifiers, operators, strings

```
color = input("Enter your favourite color: ")
```

![Diagram of tokenization process](https://deepsource.io/blog/introduction-static-code-analysis/)
Static Analysis

- Step 2: Parsing
  - To the compiler/interpreter, software is a tree
  - Root node is file/module
  - Leaves mainly identifiers, literals
  - Internal nodes capture structure

\[
x = 1 \rightarrow \text{assign}\]
\[
x \quad 1
\]

Consider checking out: [https://ast.carlosroso.com/](https://ast.carlosroso.com/)
Static Analysis

- **Step 2: Parsing**
  - What does this get us?
  - Rich structure
    - Syntactic types (variables, method calls)
    - Dead code, deep nesting
  - A lot of type resolution
    - What vars are stored, loaded
    - Not complete!
    - Need to *build* to understand imports
Static Analysis

- Step 2b: Advanced Analysis
  - The compiler doesn’t stop at parsing
  - Familiar?

```java
public boolean div(int a, int z) {
    int x = 5;
    if (a <= 1) {
        x = a - 1;
    }
    return z / x;
}
```
Step 2b: Advanced Analysis

- The compiler doesn’t stop at parsing
- There is a lot more down this rabbit hole
  - Control/data-flow, abstract interpretation, (dynamic) symbolic execution,
- Consider a Programming Languages or Compilers course
Static Analysis

- Step 3: register analyzers
  - At the core: walk the tree

```python
class ListDefinitionChecker(BaseChecker):
    msg = "usage of 'list()' detected, use '[]' instead"

def visit_Call(self, node):
    name = getattr(node.func, "id", None)
    if name and name == list.__name__ and not node.args:
        self.violations.append((self.filename, node.lineno, self.msg))
```
Static Analysis

- Step 3: register analyzers
  - At the core: walk the tree
  - Sometimes more complex

```python
class UnusedImportChecker(BaseChecker):
    def __init__(self):
        self.import_map = defaultdict(set)
        self.name_map = defaultdict(set)

    def _add_imports(self, self, node):
        for import_name in node.names:
            # Store only top-level module name ("os.path" -> "os").
            # We can't easily detect when "os.path" is used.
            name = import_name.name.partition(".")[0]
            self.import_map[self.filename].add((name, node.lineno))

    def visit_Import(self, self, node):
        self._add_imports(node)

    def visit_ImportFrom(self, self, node):
        self._add_imports(node)

    def visit_Name(self, self, node):
        # We only add those nodes for which a value is being read from.
        if isinstance(node.ctx, ast.Load):
            self.name_map[self.filename].add(node.id)
```

https://deepsource.io/blog/introduction-static-code-analysis/
Static Analysis

- Compared to Linters:
  - Linters mainly enforce style -- comments, quotes, idioms
    - This also requires static analysis! Just nothing particularly fancy
  - Some overlap; good conventions help avoid bugs
Static Analysis

- Compared to Parsers:
  - Parsers check for syntactic correctness
    - Can catch bugs as well, e.g. missing ","
  - Parsing is often a key step in static analysis
    - Hard to do right with just text/regexes.
  - Parsing is a platform for further analyses
    - control-flow, data-flow
So… Static Analysis for Everything?

● Can we find every bug?
  ○ No! Rice’s Theorem
    "Any nontrivial property about the language recognized by a Turing machine is undecidable.‘‘ -- Henry Gordon Rice, 1953
  ○ Every static analysis is necessarily incomplete or unsound or undecidable (or multiple of these)
So… Static Analysis for Everything?

- Can we find every bug?
- Can we guarantee correctness?
So... Static Analysis for Everything?

- Can we find every bug?
- Can we guarantee correctness?
  - Yes, but... much less useful

```java
public class Fails {
    public static void main(String[] args) {
        getValue(i : null);
    }

    private static int getValue(Integer i) {
        return i.intValue();
    }
}
```
Soundness & Precision

● Since we can’t perfectly analyze behavior statically
  ○ We may miss things by being cautious (unsound; false negative)
  ○ We might identify non-problems (imprecision, false positive)

Program state covered in actual execution

Program state covered by abstract execution with analysis

unsound (false negative)

imprecise (false positive)
The Social Side

- How to deploy tools that are neither sound nor complete?
Static Analysis at Google

- Centered around FindBugs (succeeded by SpotBugs)
  - Essentially, a huge collection of risky patterns on Java bytecode
  - Annotated with five levels of concern
Static Analysis at Google

● Three experiments in the early 2000s:

1. **A dashboard:** run FindBugs overnight, report results in a centralized location

   *Failed because:* dashboard is outside the developer’s workflow
Static Analysis at Google

● Three experiments in the early 2000s:

1. A dashboard: run FindBugs overnight, report results in a centralized location
   
   Failed because: dashboard is outside the developer’s workflow

2. Recurring FixIt events: company-wide one-week effort to fix warnings
   
   Failed because: actually fixed some bugs, but FindBugs is too imprecise
   (44% of issues were “bugs”, but only 16% mattered)
Static Analysis at Google

- Three experiments in the early 2000s:
  1. **A dashboard**: run FindBugs overnight, report results in a centralized location
     
     *Failed because*: dashboard is outside the developer’s workflow
  2. **Recurring FixIt events**: company-wide one-week effort to fix warnings
     
     *Failed because*: actually fixed some bugs, but FindBugs is too imprecise
     (44% of issues were “bugs”, but only 16% mattered)
  3. **Add to Code Review**: run on every change, allow toggling warnings
     
     *Failed because*: too imprecise; suppressing FPs made it inconsistent
Static Analysis at Google

Okay so then what?

● What went wrong / what do we need?
Static Analysis at Google

Okay so then what?

- What went wrong / what do we need?
  1. Precision is key -- developers lose faith in inaccurate tools
  2. Provide timely warnings -- in-IDE or rapidly on builds
     a. Checkers are way more useful during coding
  3. Make a platform -- allow adding useful checks
Static Analysis at Google

Specifically:

● At compile-time:
  ○ Perfectly Precise
    ■ No false-positives; never halt a build incorrectly
  ○ Simple
  ○ Actionable
    ■ Ideally to the point of auto-fix suggestions
Static Analysis at Google

Specifically:

- **At review time: TriCoder**
  - 90%+ precise
    - If it drops below, checker gets disabled! Onus on checker authors to fix
  - Actionable, but may require some work
  - Improve correctness or code quality
  - Some compile-time checks moved to review-time!

- **Ran 50K times per day -- in 2018**
package com.google.devtools.staticanalysis;

public class Test {
    public boolean foo() {
        return getString().equals("foo").toString();
    }

    public String getString() {
        return new String("foo");
    }
}

//import java.util.Objects;

package com.google.devtools.staticanalysis;

public class Test {
    public boolean foo() {
        return Objects.equals(getString(), "foo".toString());
    }

    public String getString() {
        return new String("Foo");
    }
}
Static Analysis at Google

- The gist: Many simple precise checks
  - What else could one do?
Static Analysis at Google

- The gist: Many simple precise checks
  - What else could one do?

- Infer at Facebook
  - Built around separation logic; geared heavily towards tracking resources
    - Null-pointer dereferences, resource leaks, unintended data access
  - Google claims this won’t (easily) scale to their multi-billion line mono-repo
Static Analysis at Google

- The gist: Many simple precise checks
  - What else could one do?
- Use AI?
  - Rule-mining from previous reviews
    - Detects typical vulnerabilities, bad patterns
  - Mostly fairly simple ML (details limited)
Static Analysis at Google

- The gist: Many simple precise checks
  - What else could one do?
- Use AI?
  - Microsoft’s IntelliSense in VSCode
  - Mostly refactorings, code completions
  - Trained on large volumes of code
Static Analysis at Google

● The gist: Many simple precise checks
  ○ What else could one do?

● Use AI?
  ○ Shameless plug: AI is rapidly learning to program. If this interests you, come do research with me :)

Programma, cura te ipsum
Summary

● We all constantly make mistakes
  ○ Static analysis captures common issues
  ○ Choose suitable abstractions; consider trade-offs
    ■ E.g., dynamic vs. static typing; sound vs. precise

● At big-tech-scale, automated checks are key
  ○ Help normalize coding standards
  ○ Even rare bugs are common at scale
  ○ But: social factors are very important
HW6 Feedback

https://rb.gy/itzmja