Carnegie Mellon University

Static and Dynamic Analysis

17-313, Foundations of Software Engineering, Spring 2023

S3D Software and Societal Systems Department

Learning Goals

- Gain an understanding of the relative strengths and weaknesses of static and dynamic analysis
- Examine several popular analysis tools and understand their use cases
- Understand how analysis tools are used in large open source software

S3D

Activity: Analyze the Python program statically

```
def n2s(n: int, b: int):
  if n <= 0: return '0'
  r = ''
  while n > 0:
    u = n % b
    if u >= 10:
      u = chr(ord('A') + u-10)
    n = n // b
    r = str(u) + r
  return r
```

1. What are the set of data types taken by variable `u` at any point in the program?
2. Can the variable `u` be a negative number?
3. Will this function always return a value?
4. Can there ever be a division by zero?
5. Will the returned value ever contain a minus sign '-'?

S3D

What static analysis can and cannot do

- Type-checking is well established
  - Set of data types taken by variables at any point
  - Can be used to prevent type errors (e.g. Java) or warn about potential type errors (e.g. Python)

- Checking for problematic patterns in syntax is easy and fast
  - Is there a comparison of two Java strings using `==`?
  - Is there an array access `a[i]` without an enclosing bounds check for `i`?

- Reasoning about termination is impossible in general
  - Halting problem

- Reasoning about exact values is hard, but conservative analysis via abstraction is possible
  - Is the bounds check before `a[i]` guaranteeing that `I` is within bounds?
  - Can the divisor ever take on a zero value?
  - Could the result of a function call be `42`?
  - Will this multi-threaded program give me a deterministic result?
  - Be prepared for “MAYBE”

- Verifying some advanced properties is possible but expensive
  - CI-based static analysis usually over-approximates conservatively

S3D

The Bad News: Rice's Theorem

Every static analysis is necessarily incomplete, unsound, undecidable, or a combination thereof

> "Any nontrivial property about the language recognized by a Turing machine is undecidable."

Henry Gordon Rice, 1953

Static Analysis is well suited to detecting certain defects

- **Security:** Buffer overruns, improperly validated input…
- **Memory safety:** Null dereference, uninitialized data…
- **Resource leaks:** Memory, OS resources…
- **API Protocols:** Device drivers; real time libraries; GUI frameworks
- **Exceptions:** Arithmetic/library/user-defined
- **Encapsulation:**
  - Accessing internal data, calling private functions…
- **Data races:**
  - Two threads access the same data without synchronization

Activity: Analyze the Python program dynamically

```
def n2s(n: int, b: int):
  if n <= 0: return '0'
  r = ''
  while n > 0:
    u = n % b
    if u >= 10:
      u = chr(ord('A') + u-10)
    n = n // b
    r = str(u) + r
  return r

print(n2s(12, 10))
```

1. What are the set of data types taken by variable `u` at any point in the program?
2. Did the variable `u` ever contain a negative number?
3. For how many iterations did the while loop execute?
4. Was there ever be a division by zero?
5. Did the returned value ever contain a minus sign '-'?

S3D

Dynamic analysis reasons about program executions

- Tells you properties of the program that were definitely observed
  - Code coverage
  - Performance profiling
  - Type profiling
  - Testing

- In practice, implemented by program *instrumentation*
  - Think “Automated logging”
  - Slows down execution speed by a small amount

S3D

Static Analysis

- Requires only source code
- Conservatively reasons about all possible inputs and program paths
- Reported warnings may contain false positives
- Can report all warnings of a particular class of problems
- Advanced techniques like verification can prove certain complex properties, but rarely run in CI due to cost

Dynamic Analysis

- Requires successful build + test inputs
- Observes individual executions
- Reported problems are real, as observed by a witness input
- Can only report problems that are seen. Highly dependent on test inputs. Subject to false negatives
- Advanced techniques like symbolic execution can prove certain complex properties, but rarely run in CI due to cost

S3D

Static Analysis Tools

Tools for Static Analysis



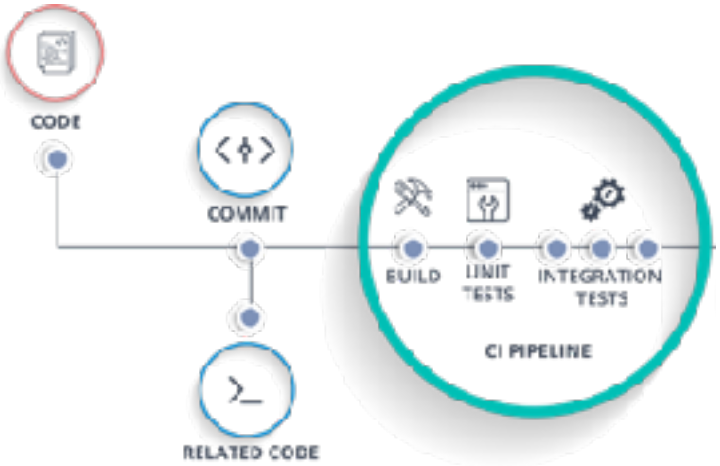


Static analysis can be applied to all attributes

- Find bugs
- Refactor code
- Keep your code stylish!
- Identify code smells
- Measure quality
- Find usability and accessibility issues
- Identify bottlenecks and improve performance

S3D

Static analysis is a key part of continuous integration



Travis CI

GitHub Actions

sonarqube

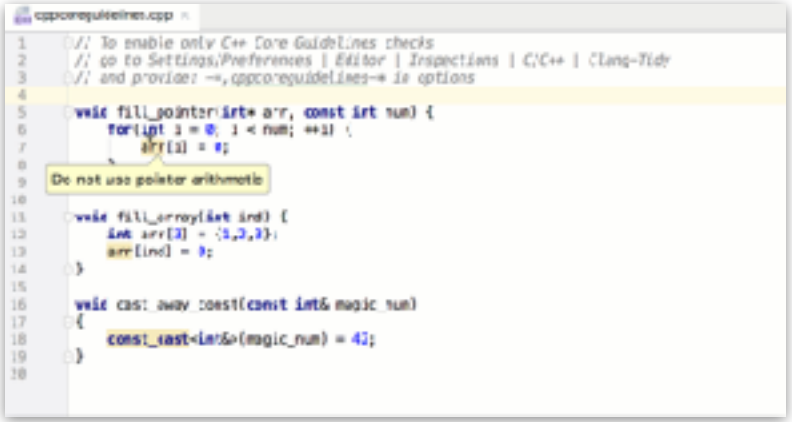
S3D

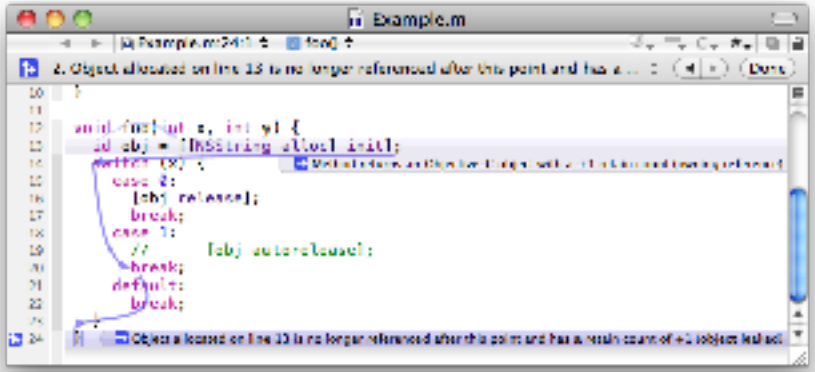
Static analysis is a growing industry

https://www.sdxcentral.com/articles/news/snyk-secures-150m-snags-1b-valuation/2020/01/
https://techcrunch.com/2019/09/18/github-acquires-code-analysis-tool-semmle/
https://github.com/marketplace

S3D

Static analysis is also integrated into IDEs





https://clang-analyzer.llvm.org

S3D

What makes a good static analysis tool?

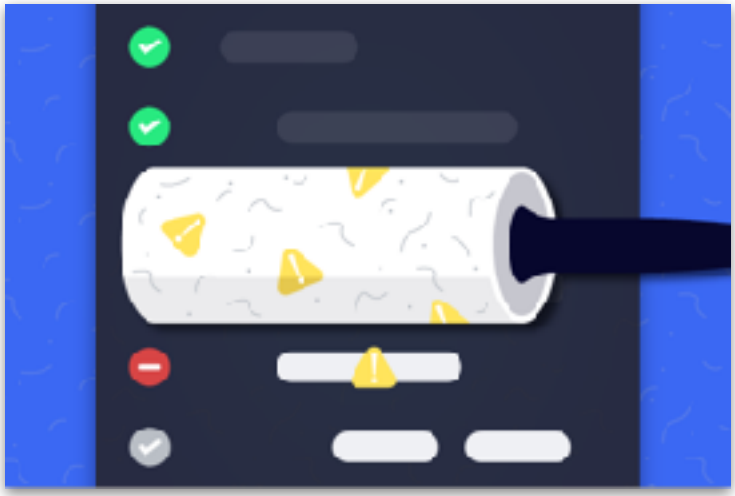
- Static analysis should be **fast**
  - Don’t hold up development velocity
  - This becomes more important as code scales
- Static analysis should report **few false positives**
  - Otherwise developers will start to ignore warnings and alerts, and quality will decline
- Static analysis should be **continuous**
  - Should be part of your continuous integration pipeline
  - Diff-based analysis is even better -- don’t analyse the entire codebase; just the changes
- Static analysis should be **informative**
  - Messages that help the developer to quickly locate and address the issue
  - Ideally, it should suggest or automatically apply fixes

https://cacm.acm.org/magazines/2010/2/69354-a-few-billion-lines-of-code-later/fulltext

S3D

Linters

Cheap, fast, and lightweight static source analysis



https://www.perforce.com/blog/qac/what-lint-code-and-why-linting-important

S3D

Linters for Maintainability

Use linters to improve maintainability

Why? We spend more time reading code than writing it.

- Developers spend most of their time maintaining code
  - Various estimates of the exact %, some as high as 80%
- Code is ownership is usually shared
- The original owner of some code may move on
- Code conventions make it easier for other developers to quickly understand your code

S3D

Use Style Guidelines to facilitate communication

- Indentation
- Comments
- Line length
- Naming
- Directory structure
- ...

Guidelines are inherently opinionated, but **consistency** is the important point.
Agree to a set of conventions and stick to them.

https://www.chicagomanualofstyle.org/ | https://google.github.io/styleguide/ | https://www.python.org/dev/peps/pep-0008

S3D

Use linters to enforce style guidelines

Don’t rely on manual inspection during code review!

https://checkstyle.sourceforge.io/

S3D

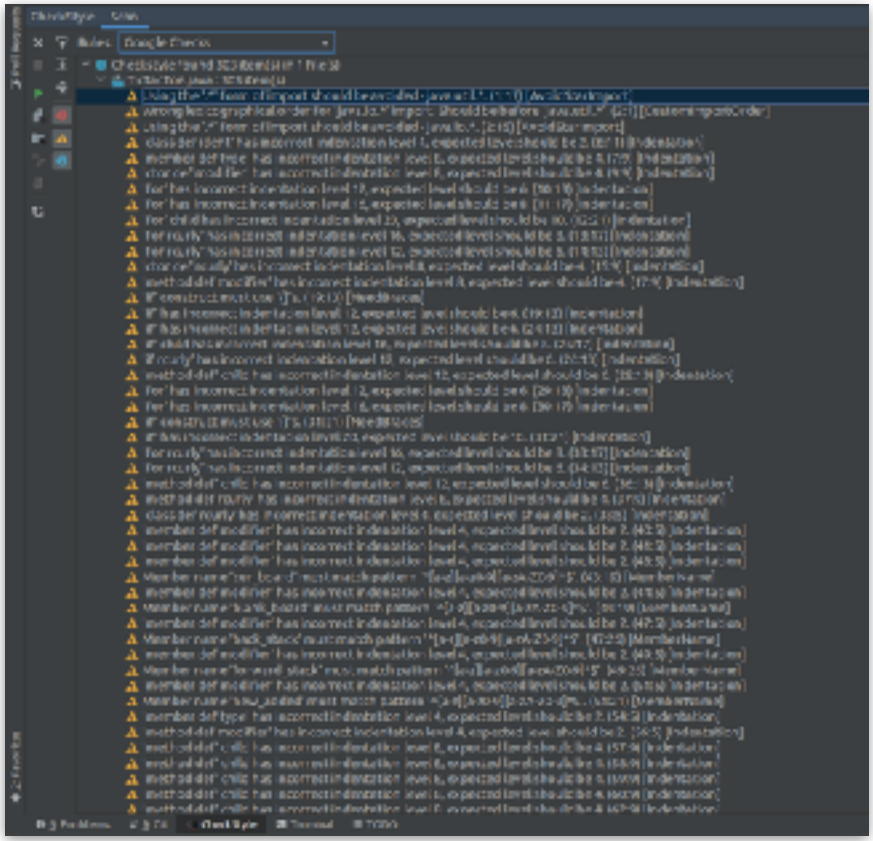
Example: CheckStyle

```
<module name="WhitespaceAround">
  <property name="allowEmptyConstructors" value="true"/>
  <property name="allowEmptyLambdas" value="true"/>
  <property name="allowEmptyMethods" value="true"/>
  <property name="allowEmptyTypes" value="true"/>
  <property name="allowEmptyLoops" value="true"/>
  <property name="ignoreEnhancedForColon" value="false"/>
  <property name="tokens"
            value="ASSIGN, BAND, BAND_ASSIGN, BCOR, BOR_ASSIGN, BSR, BSR_ASSIGN, BXOR,
                   BXOR_ASSIGN, COLON, DIV, DIV_ASSIGN, DO_WHILE, EQUAL, GE, GT, LAMBDA, LAND,
                   LCURLY, LE, LITERAL_CATCH, LITERAL_DO, LITERAL_ELSE, LITERAL_FINALLY,
                   LITERAL_FOR, LITERAL_IF, LITERAL_RETURN, LITERAL_SWITCH, LITERAL_SYNCHRONIZED,
                   LITERAL_TRY, LITERAL_WHILE, LOR, LT, MINUS, MINUS_ASSIGN, MOD, MOD_ASSIGN,
                   NOT_EQUAL, PLUS, PLUS_ASSIGN, QUESTION, RCURLY, SL, SLIST, SL_ASSIGN, SR,
                   SR_ASSIGN, STAR, STAR_ASSIGN, LITERAL_ASSERT, TYPE_EXTENSION_AND"/>
  <message key="ws.notFollowed"
           value="WhitespaceAround: ''{0}'' is not followed by whitespace. Empty blocks may only
  <message key="ws.notPreceded"
           value="WhitespaceAround: ''{0}'' is not preceded with whitespace."/>
</module>
```

```
<module name="Indentation">
  <property name="basicOffset" value="2"/>
  <property name="braceAdjustment" value="2"/>
  <property name="caseIndent" value="2"/>
  <property name="throwsIndent" value="4"/>
  <property name="lineWrappingIndentation" value="4"/>
  <property name="arrayInitIndent" value="2"/>
</module>
```

...

https://github.com/checkstyle/checkstyle/blob/master/src/main/resources/google_checks.xml



```java
@Override
public boolean equals(Object o) {
    if (o == this)
        return true;
```

```java
private Board cur_board;
private Board blank_board;
```

```java
public static void main(String[] args) throws Exception {
    TicTacToe tictactoe = new TicTacToe();
```

Integrate style checking into your CI

```
plugins {
    id 'checkstyle'
}

...

checkstyle {
  ignoreFailures = true
  toolVersion = "6.7"
  sourceSets = [sourceSets.main]
}
```

Travis CI

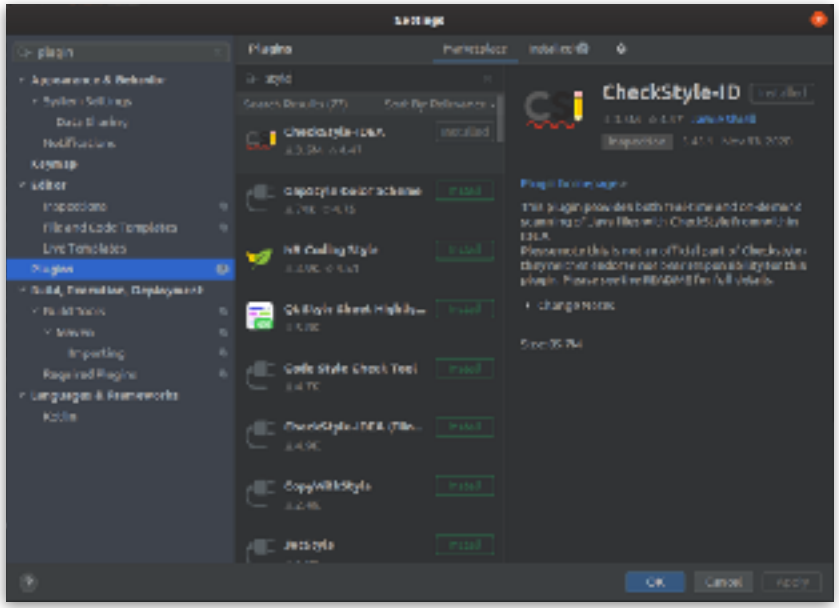
GitHub Actions

```
name: Tox lint checking
on: [pull_request]
jobs:
  build:
    runs-on: ubuntu-20.04
    steps:
    - uses: actions/checkout@v2
    - name: Install Python
      uses: actions/setup-python@v2
      with:
        python-version: 3.9.5
    - name: Install pipenv
      run: pip install pipenv==2021.5.29
    - id: cache-pipenv
      uses: actions/cache@v2
      with:
        path: ~/.local/share/virtualenvs
        key: ${{ runner.os }}-pipenv-${{ hashFiles('**/Pipfile.lock') }}
    - name: Install package
      if: steps.cache-pipenv.outputs.cache-hit != 'true'
      run: |
        pipenv install --dev
    - name: Flake8
      run: pipenv run flake8 src
    - name: MyPy
      run: pipenv run mypy src
```

https://docs.gradle.org/current/userguide/checkstyle_plugin.html

S3D

Automatically reformat your existing code
Developer time is valuable!



https://www.jetbrains.com/help/idea/reformat-and-rearrange-code.html

S3D

Take Home Message:
Style is an easy way to improve readability

- Everyone has their own opinion (e.g., tabs vs. spaces)
- Agree to a convention and stick to it
  - Use continuous integration to enforce it
- Use automated tools to fix issues in existing code

Pattern-Based Static Analyzers

Cheap and fast tools that scan Abstract Syntax Trees for common developer mistakes known as patterns

clang-tidy

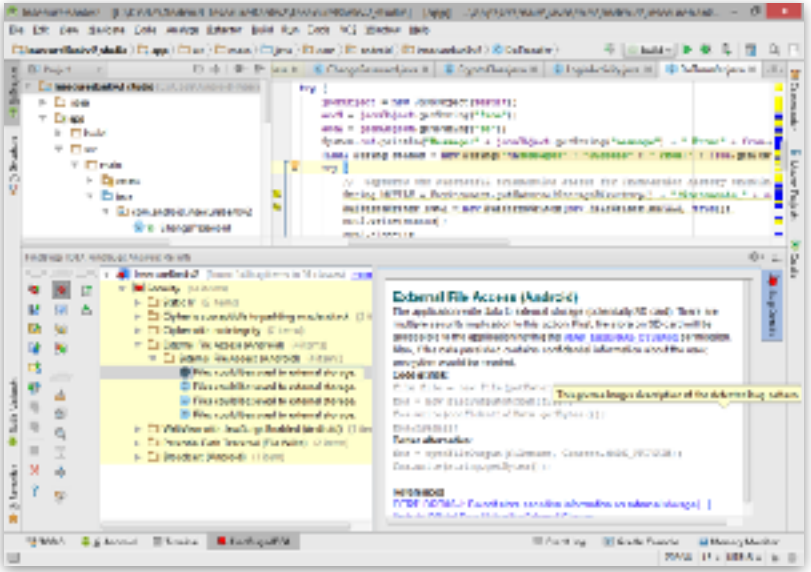
https://github.com/analysis-tools-dev/static-analysis

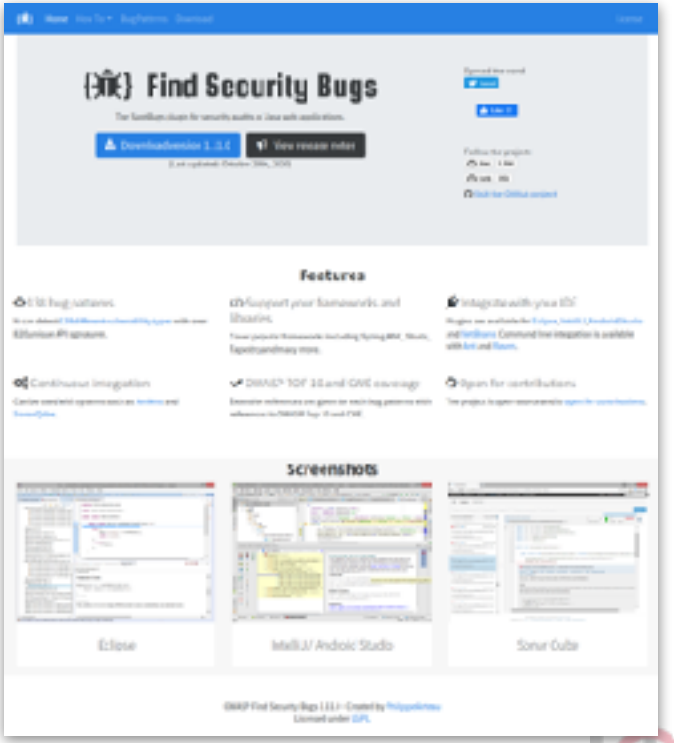
SpotBugs

- Bad Practice
- Correctness
- Performance
- Internationalization
- Malicious Code
- Multithreaded Correctness
- Security
- Dodgy Code

http://findbugs.sourceforge.net/bugDescriptions.html
https://spotbugs.readthedocs.io/en/latest/bugDescriptions.html

SpotBugs can be extended with plugins





https://find-sec-bugs.github.io/

Bad Practice:

```
String x = new String("Foo");
String y = new String("Foo");

if (x == y) {
  System.out.println("x and y are the same!");
} else {
  System.out.println("x and y are different!");
}
```

S3D

Bad Practice: `ES_COMPARING_STRINGS_WITH_EQ`

Comparing strings with ==

```
String x = new String("Foo");
String y = new String("Foo");

if (x == y) {
if (x.equals(y)) {
  System.out.println("x and y are the same!");
} else {
  System.out.println("x and y are different!");
}
```

Performance:

```
public static String repeat(String string, int times)
{
  String output = string;
  for (int i = 1; i < times; ++i) {
    output = output + string;
  }
  return output;
}
```

Performance: `SBSC_USE_STRINGBUFFER_CONCATENATION`

Method concatenates strings using + in a loop

```
public static String repeat(String string, int times)
{
  String output = string;
  for (int i = 1; i < times; ++i) {
    output = output + string;
  }
  return output;
}
```

The method seems to be building a String using concatenation in a loop. In each iteration, the String is converted to a StringBuffer/StringBuilder, appended to, and converted back to a String. **This can lead to a cost quadratic in the number of iterations, as the growing string is recopied in each iteration.**

Performance: `SBSC_USE_STRINGBUFFER_CONCATENATION`

Method concatenates strings using + in a loop

```
public static String repeat(String string, int times)
{
  StringBuffer output = new StringBuffer(string);
  for (int i = 1; i < times; ++i) {
    output.append(string);
  }
  return output.toString();
}
```

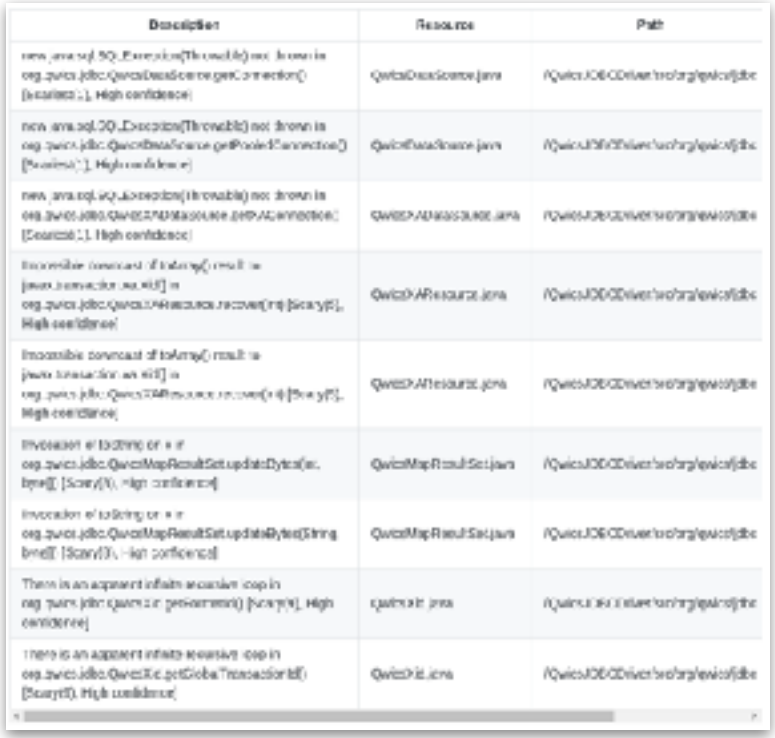
Performance: `SBSC_USE_STRINGBUFFER_CONCATENATION`

Method concatenates strings using + in a loop

```
public static String repeat(String string, int times)
{
  int length = string.length() * times;
  StringBuffer output = new StringBuffer(length);
  for (int i = 0; i < times; ++i) {
    output.append(string);
  }
  return output.toString();
}
```

Correctness: Lots of issues here!

```
public class QwicsXid implements Xid {
  private byte[] globalTransactionId;
  private byte[] branchQualifier;
  private int formatId;
  ...

  @Override
  public byte[] getBranchQualifier() {
    return this.branchQualifier;
  }
  @Override
  public int getFormatId() {
    return this.getFormatId();
  }
  @Override
  public byte[] getGlobalTransactionId() {
    return this.getGlobalTransactionId();
  }
}
```



https://github.com/pbrune1973/qwics/blob/3b4ec904468eaed89ea890b5ae97ebaaa8e4a3e6/workspace/QwicsJDBCDriver/src/org/qwics/jdbc/QwicsXid.java
https://github.com/pbrune1973/qwics/issues/7

Correctness:

```
@Override
public Connection getConnection() throws SQLException {
  QwicsConnection con = new QwicsConnection(host, port);
  try {
    con.open();
  } catch (Exception e) {
    new SQLException(e);
  }
  return con;
}
```

https://github.com/pbrune1973/qwics/blob/3b4ec904468eaed89ea890b5ae97ebaaa8e4a3e6/workspace/QwicsJDBCDriver/src/org/qwics/jdbc/QwicsXid.java
https://github.com/pbrune1973/qwics/issues/7

Correctness:

```
@Override
public Connection getConnection() throws SQLException {
  QwicsConnection con = new QwicsConnection(host, port);
  try {
    con.open();
  } catch (Exception e) {
    throw new SQLException(e);
  }
  return con;
}
```

https://github.com/pbrune1973/qwics/blob/3b4ec904468eaed89ea890b5ae97ebaaa8e4a3e6/workspace/QwicsJDBCDriver/src/org/qwics/jdbc/QwicsXid.java
https://github.com/pbrune1973/qwics/issues/7

What are some of the problems with SpotBugs?

S3D

Google: Move static checks to the compiler

Developers can ignore warnings, but they can’t ignore build errors

clang-tidy

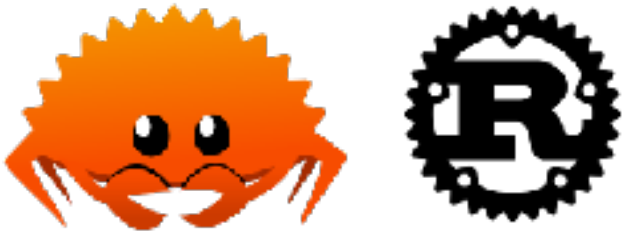
Error Prone

https://cacm.acm.org/magazines/2018/4/226371-lessons-from-building-static-analysis-tools-at-google/fulltext

S3D

New languages have embraced the same idea

Code smells will cause the build to fail (e.g., dead code)



Challenges

- The analysis must produce zero false positives
  - Otherwise developers won't be able to build the code!
- The analysis needs to be really fast
  - Ideally < 100 ms
  - If it takes longer, developers will become irritated and lose productivity
- You can't just "turn on" a particular check
  - Every instance where that check fails will prevent existing code from building
  - There could be thousands of violations for a single check across large codebases

S3D

Challenges

- The analysis must produce zero false positives
  - Otherwise developers won't be able to build the code!
- The analysis needs to be really fast
  - Ideally < 100 ms
  - If it takes longer, developers will become irritated and lose productivity
- You can't just "turn on" a particular check
  - Every instance where that check fails will prevent existing code from building
  - There could be thousands of violations for a single check across large codebases

S3D

Solution: Automatically patch existing bugs

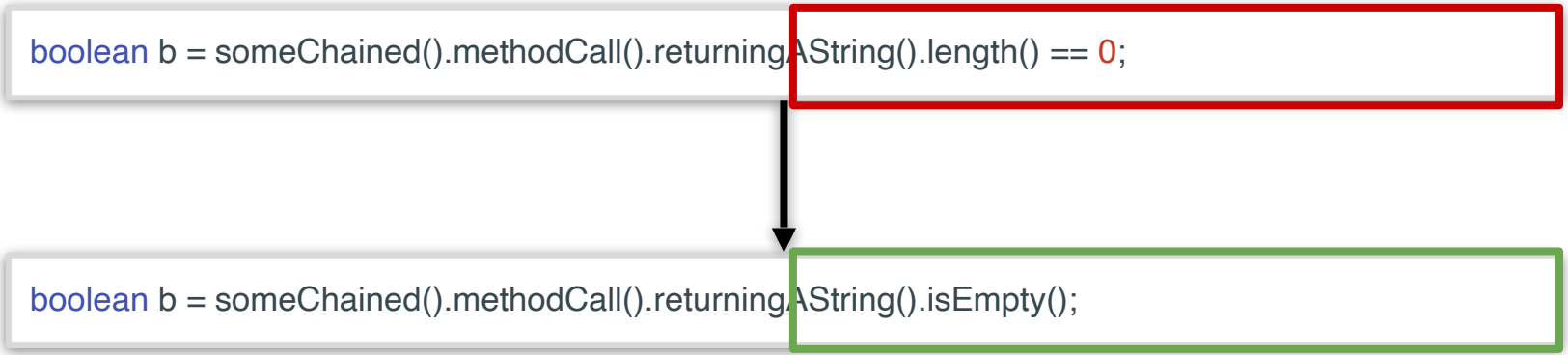
```
public class StringIsEmpty {
  @BeforeTemplate
  boolean equalsEmptyString(String string) {
    return string.equals("");
  }

  @BeforeTemplate
  boolean lengthEquals0(String string) {
    return string.length() == 0;
  }

  @AfterTemplate
  @AlsoNegation
  boolean optimizedMethod(String string) {
    return string.isEmpty();
  }
}
```

@BeforeTemplate finds String expressions that match the body of the method.

@AfterTemplate rewrites matching String expressions to match the body of the method.

https://errorprone.info/docs/refaster

Solution: Automatically patch existing bugs



https://errorprone.info/docs/refaster

S3D

Summary: Linters

- Linters are cheap and fast static analysis tools!
- Style checkers can improve readability of code
- Pattern-based bug detectors catch common developer mistakes
  - Code smells, performance issues, correctness, …
  - They don’t know the intent of the program, leading to occasional false positives
  - They reveal issues that are genuine, but which we don’t sufficiently care about
  - The best tools automatically fix detected issues
  - Each developer mistake needs its own analyzer / AST checker
  - They *complement* but don’t *replace* testing

S3D

Java Checker Framework

Uses annotations to detect common errors

- Uses a conservative analysis to prove the absence of certain defects *
  - Null pointer errors, uninitialized fields, certain liveness issues, information leaks, SQL injections, bad regular expressions, incorrect physical units, bad format strings, ...
  - C.f. SpotBugs which makes no safety guarantees
  - Assuming that code is annotated and those annotations are correct
- Uses annotations to enhance Java's type system

https://checkerframework.org/

Annotations can be applied to types and declarations

```
// return value
@InternedString intern() { ... }

// parameter
int compareTo(@NonNullString other) { ... }

// receiver ("this" parameter)
String toString(@TaintedMyClass this) { ... }

// generics: non-null list of interned Strings
@NonNullList<@InternedString> messages;

// arrays:  non-null array of interned Strings
@InternedString @NonNull[] messages;

// cast
myDate = (@InitializedDate) beingConstructed;
```

Detecting null pointer exceptions

- **@Nullable** indicates that an expression may be null
- **@NonNull** indicates that an expression must never be null
  - Rarely used because @NonNull is assumed by default
  - See documentation for other nullness annotations
- Guarantees that expressions annotated with @NonNull will **never** evaluate to null, forbids other expressions from being dereferenced

https://checkerframework.org/manual/#nullness-annotations

S3D

```java
import org.checkerframework.checker.nullness.qual.*;

public class NullnessExampleWithWarnings {
    public void example() {
        @NonNull String foo = "foo";
        String bar = null;

        foo = bar;
    }
}
```

```
import org.checkerframework.checker.nullness.qual.*;

public class NullnessExampleWithWarnings {
    public void example() {
        @NonNull String foo = "foo";
        String bar = null;

        foo = bar;
    }
}
```

`@Nullable` is applied by default

```
import org.checkerframework.checker.nullness.qual.*;

public class NullnessExampleWithWarnings {
    public void example() {
        @NonNull String foo = "foo";
        String bar = null;

        foo = bar;
    }
}
```

`@Nullable` is applied by default

Error: [assignment.type.incompatible] incompatible types in assignment.
found : @Initialized **@Nullable** String
required: @UnknownInitialization **@NonNull** String

```
import org.checkerframework.checker.nullness.qual.*;

public class NullnessExampleWithWarnings {
  public void example() {
    @NonNull String foo = "foo";
    String bar = null;   // @Nullable

    if (bar != null) {
      foo = bar;
    }
  }
}
```

bar is refined to `@NonNull`

Is there a bug?

```
public String getDay(int dayIndex) {
  String day = null;
  switch (dayIndex) {
    case 0: day = "Monday";
    case 1: day = "Tuesday";
    case 2: day = "Wednesday";
    case 3: day = "Thursday";
  }
  return day;
}

public void example() {
  @NonNull String dayName = getDay(4);
  System.out.println("Today is " + dayName);
}
```

Is there a bug? Yes.

```
public String getDay(int dayIndex) {
  String day = null;
  switch (dayIndex) {
    case 0: day = "Monday";
    case 1: day = "Tuesday";
    case 2: day = "Wednesday";
    case 3: day = "Thursday";
  }
  return day;
}

public void example() {
  @NonNull String dayName = getDay(4);
  System.out.println("Today is " + dayName);
}
```

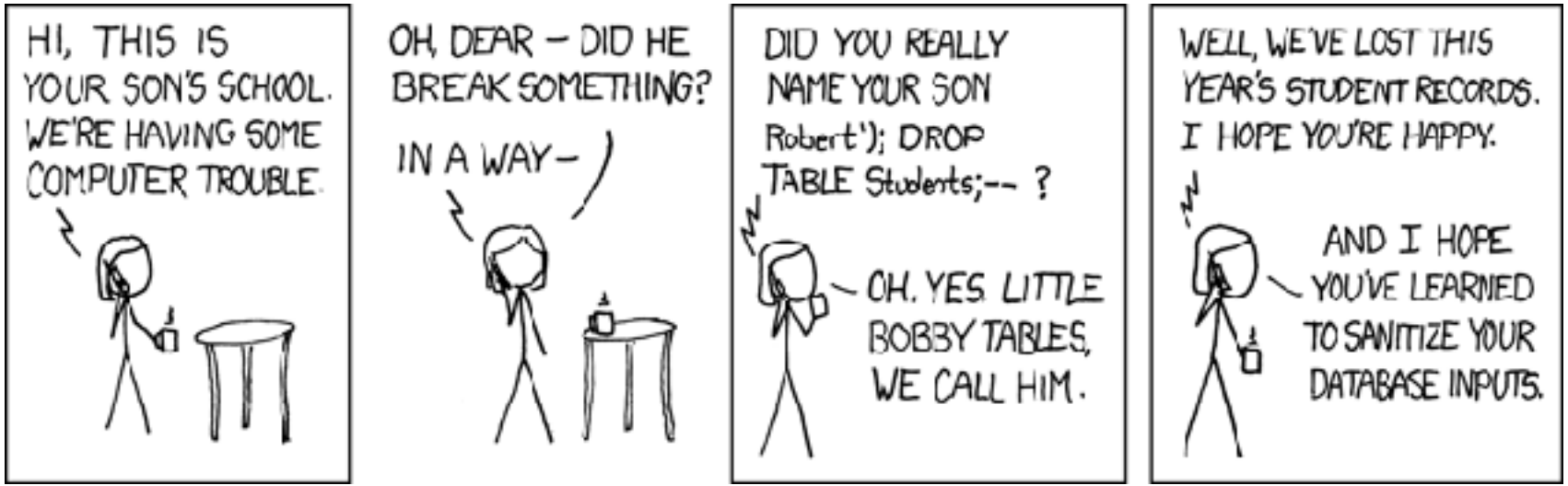
Error: [return.type.incompatible] incompatible types in return.
type of expression: @Initialized **@Nullable** String
method return type: @Initialized **@NonNull** String

Taint Analysis

Prevents untrusted (tainted) data from reaching sensitive locations (sinks)

- Tracks flow of sensitive information through the program
- Tainted inputs come from arbitrary, possibly malicious sources
  - User inputs, unvalidated data
- Using tainted inputs may have dangerous consequences
  - Program crash, data corruption, leak private data, etc.
- We need to check that inputs are sanitized before reaching sensitive locations

Classic Example: SQL Injection



https://xkcd.com/327

Classic Example: SQL Injection

```
void processRequest() {
  String input = getUserInput();
  String query = "SELECT ... " + input;
  executeQuery(query);
}
```

Classic Example: SQL Injection

```
void processRequest() {
  String input = getUserInput();
  String query = "SELECT ... " + input;
  executeQuery(query);
}
```

Tainted input arrives from an untrusted source

Tainted input flows to a sensitive sink

Classic Example: SQL Injection

```
void processRequest() {
  String input = getUserInput();
  input = sanitizeInput(input);
  String query = "SELECT ... " + input;
  executeQuery(query);
}
```

Taint is removed by sanitizing the data

We can now safely execute query on untainted data

Taint Checker: `@Tainted` and `@Untainted`

```
void processRequest() {
  @Tainted String input = getUserInput();
  executeQuery(input);
}

public void executeQuery(@Untainted String input) {
  // ...
}

@Untainted public String validate(String userInput) {
  // ...
}
```

Taint Checker: `@Tainted` and `@Untainted`

```
void processRequest() {
  @Tainted String input = getUserInput();
  executeQuery(input);
}

public void executeQuery(@Untainted String input) {
  // ...
}

@Untainted public String validate(String userInput) {
  // ...
}
```

Indicates that data is tainted

Argument ***must*** be untainted

Guarantees that return value is untainted

Taint Checker: `@Tainted` and `@Untainted`

```
void processRequest() {
  @Tainted String input = getUserInput();
  executeQuery(input);
}

public void executeQuery(@Untainted String input) {
  // ...
}

@Untainted public String validate(String userInput) {
  // ...
```

Indicates that data is tainted

Argument ***must*** be untainted

Guarantees that return value is untainted

Does this compile?

```
void processRequest() {
  @Tainted String input = getUserInput();
  input = validate(input);
  executeQuery(input);
}

public void executeQuery(@Untainted String input) {
  // ...
}

@Untainted public String validate(String userInput) {
  // ...
}
```

Input becomes `@Untainted`

Does this program compile?

```
void processRequest() {
  @Tainted String input = getUserInput();
  if (input.equals("little bobby drop tables")) {
    input = validate(input);
  }
  executeQuery(input);
}
```

Does this program compile? No.

```
void processRequest() {
  @Tainted String input = getUserInput();
  if (input.equals("little bobby drop tables")) {
    input = validate(input); // @Untainted
  }
  executeQuery(input); // @Tainted
}
```

Remember the Mars Climate Orbiter incident from 1999?

NASA's Mars Climate Orbiter (cost of $327 million) was lost because of a discrepancy between use of metric unit Newtons and imperial measure Pound-force.

https://www.simscale.com/blog/2017/12/nasa-mars-climate-orbiter-metric/

Units Checker identifies physical unit inconsistencies

- Guarantees that operations are performed on the same kinds and units
- Kind annotations
  - `@Acceleration, @Angle, @Area, @Current, @Length, @Luminance, @Mass, @Speed, @Substance, @Temperature, @Time`
- SI unit annotation
  - `@m, @km, @mm, @kg, @mPERs, @mPERs2, @radians, @degrees, @A, ...`

https://www.nist.gov/pml/weights-and-measures/metric-si/si-units

```
import static org.checkerframework.checker.units.UnitsTools.m;
import static org.checkerframework.checker.units.UnitsTools.mPERs;
import static org.checkerframework.checker.units.UnitsTools.s;

void demo() {
  @m int x;
  x = 5 * m;

  @m int meters = 5 * m;
  @s int seconds = 2 * s;

  @mPERs int speed = meters / seconds;
  @m int foo = meters + seconds;
  @s int bar = seconds - meters;
}
```

```
import static org.checkerframework.checker.units.UnitsTools.m;
import static org.checkerframework.checker.units.UnitsTools.mPERs;
import static org.checkerframework.checker.units.UnitsTools.s;

void demo() {
  @m int x;
  x = 5 * m;

  @m int meters = 5 * m;
  @s int seconds = 2 * s;

  @mPERs int speed = meters / seconds;
  @m int foo = meters + seconds;
  @s int bar = seconds - meters;
}
```

`@m` indicates that `x` represents meters

To assign a unit, multiply appropriate unit constant from `UnitTools`

Does this program compile?

```
import static org.checkerframework.checker.units.UnitsTools.m;
import static org.checkerframework.checker.units.UnitsTools.mPERs;
import static org.checkerframework.checker.units.UnitsTools.s;

void demo() {
  @m int x;
  x = 5 * m;

  @m int meters = 5 * m;
  @s int seconds = 2 * s;

  @mPERs int speed = meters / seconds;
  @m int foo = meters + seconds;
  @s int bar = seconds - meters;
}
```

`@m` indicates that `x` represents meters

To assign a unit, multiply appropriate unit constant from `UnitTools`

Does this program compile? No.

```
import static org.checkerframework.checker.units.UnitsTools.m;
import static org.checkerframework.checker.units.UnitsTools.mPERs;
import static org.checkerframework.checker.units.UnitsTools.s;

void demo() {
  @m int x;
  x = 5 * m;

  @m int meters = 5 * m;
  @s int seconds = 2 * s;

  @mPERs int speed = meters / seconds;
  @m int foo = meters + seconds;
  @s int bar = seconds - meters;
}
```

Addition and subtraction between meters and seconds is physically meaningless

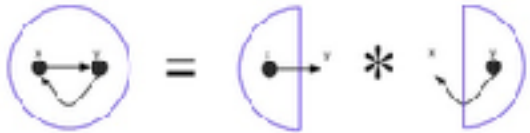
Checker Framework: Limitations

- **Can only analyze code that is annotated**
  - Requires that dependent libraries are also annotated
  - Can be tricky, but not impossible, to retrofit annotations into existing codebases
- Only considers the signature and annotations of methods
  - Doesn’t look at the implementation of methods that are being called
- Dynamically generated code
  - Spring Framework
- Can produce false positives!
  - Byproduct of necessary approximations

S3D

Infer: What if we didn’t need annotations?

- Focused on memory safety bugs
  - Null pointer dereferences, memory leaks, resource leaks, ...
- Compositional interprocedural reasoning
  - Based on separation logic and bi-abduction
- Scalable and fast
  - Can run incremental analysis on changed code
- **Does not require annotations**
- **Supports multiple languages**
  - Java, C, C++, Objective-C
  - Programs are compiled to an intermediate representation



https://fbinfer.com/

https://engineering.fb.com/2017/09/06/android/finding-inter-procedural-bugs-at-scale-with-infer-static-analyzer/

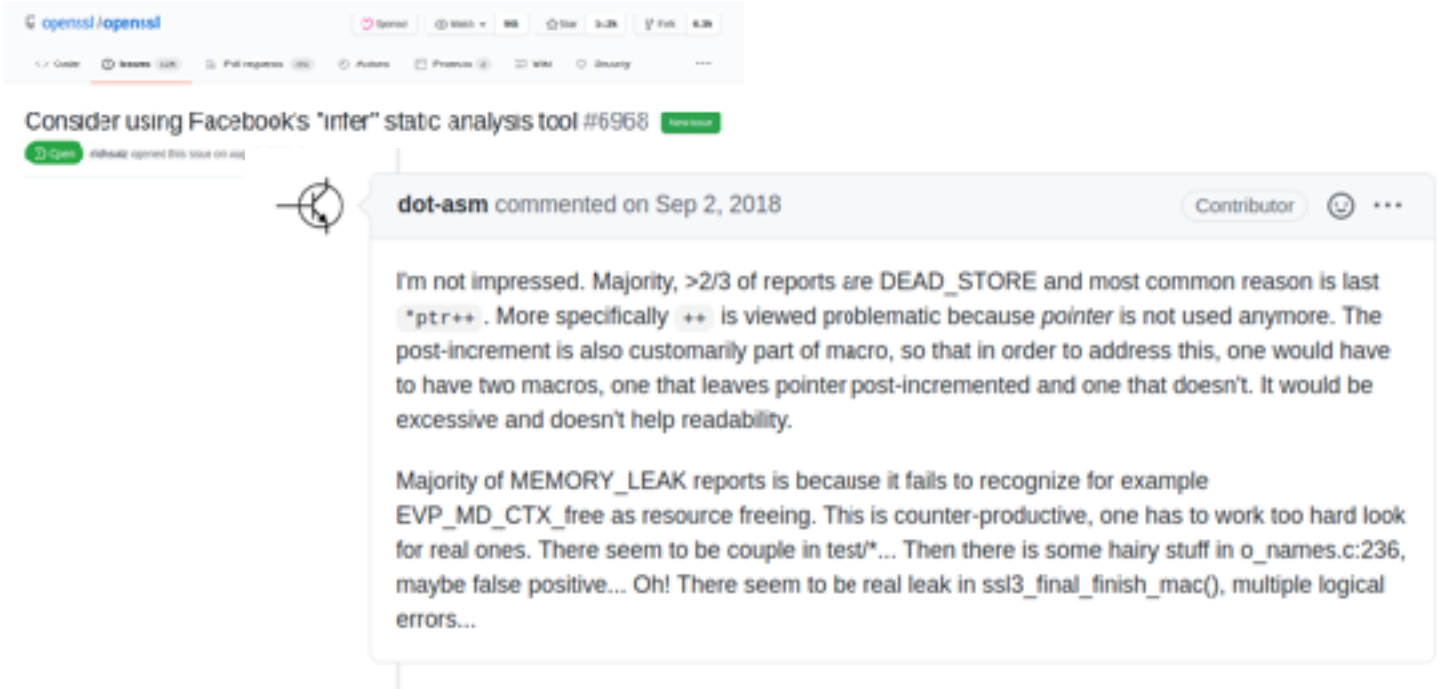
Infer: Hello World!

```
// Hello.java
class Hello {
  int test() {
    String s = null;
    return s.length();
  }
}
```

```
$ infer run -- javac Hello.java
…
Hello.java:5: error: NULL_DEREFERENCE
 object s last assigned on line 4 could be null and is dereferenced at line 5
```

https://fbinfer.com/docs/hello-world

Beware of the inevitable false positives!



openssl / openssl

Consider using Facebook's "infer" static analysis tool #6968

dot-asm commented on Sep 2, 2018 — Contributor

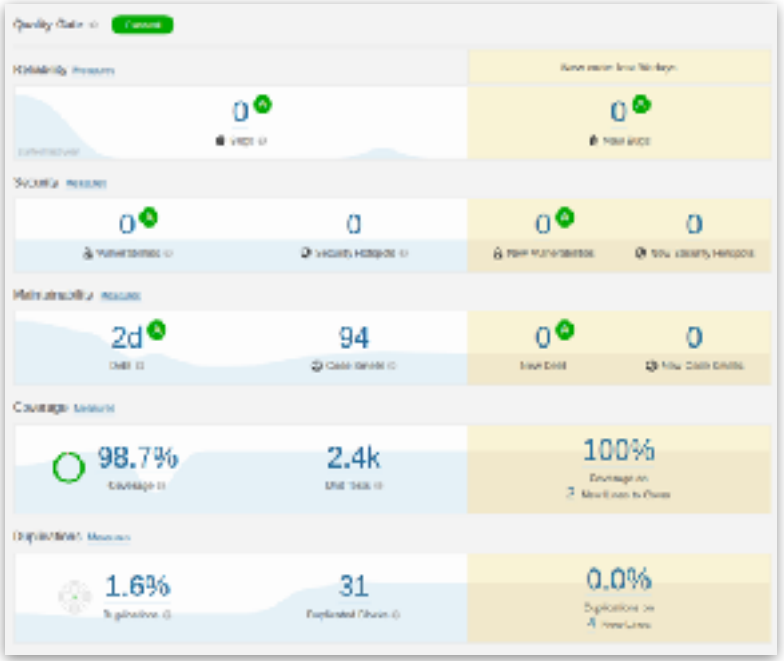
I'm not impressed. Majority, >2/3 of reports are DEAD_STORE and most common reason is last `*ptr++`. More specifically `++` is viewed problematic because *pointer* is not used anymore. The post-increment is also customarily part of macro, so that in order to address this, one would have to have two macros, one that leaves pointer post-incremented and one that doesn't. It would be excessive and doesn't help readability.

Majority of MEMORY_LEAK reports is because it fails to recognize for example EVP_MD_CTX_free as resource freeing. This is counter-productive, one has to work too hard look for real ones. There seem to be couple in test/*... Then there is some hairy stuff in o_names.c:236, maybe false positive... Oh! There seem to be real leak in ssl3_final_finish_mac(), multiple logical errors...

https://github.com/openssl/openssl/issues/6968

Analysis Dashboards

A holistic approach to quality: SonarQube

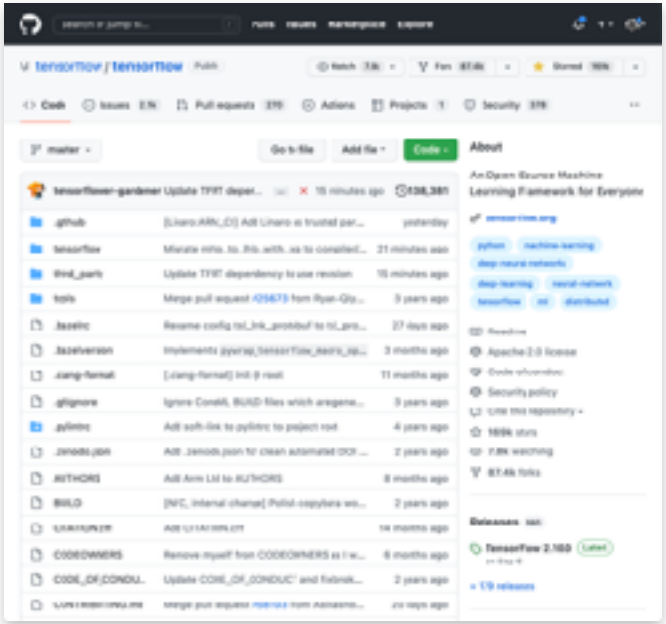


sonarcloud

sonarqube

https://sonarcloud.io

Let’s look at a real project using SonarQube: TensorFlow



TensorFlow

Fork 87.4k Starred 169k

https://sonarcloud.io/summary/overall?id=htefera_Tensorflow

What analysis tools should I use?

The best QA strategies employ a combination of tools

How Many of All Bugs Do We Find? A Study of Static Bug Detectors

Andrew Habib
andrew.a.habib@gmail.com
Department of Computer Science
TU Darmstadt
Germany

Michael Pradel
michael@binaervarianz.de
Department of Computer Science
TU Darmstadt
Germany

ABSTRACT

Static bug detectors are becoming increasingly popular and are widely used by professional software developers. While most work on bug detectors focuses on whether they find bugs at all, and on how many false positives they report in addition to a few true warnings, the inverse question is often neglected: How many of all real-world bugs do static bug detectors find? This paper addresses this question by studying the results of applying three widely used static bug detectors to an extended version of the Defects4J dataset that consists of 15 Java projects with 594 known bugs. To decide which of these bugs the tools detect, we use a novel methodology that combines an automatic analysis of warnings and bugs with a manual validation of each candidate of a detected bug. The results of the study show that: (i) static bug detectors find a non-negligible amount of all bugs, (ii) different tools are mostly complementary to each other, and (iii) current bug detectors miss the large majority of the studied bugs. A detailed analysis of bugs missed by the static detectors shows that some bugs could have been found by variants of the existing detectors, while others are domain-specific problems that do not match any existing bug pattern. These findings help potential users of such tools to assess their utility, motivate and outline directions for future work on static bug detection, and provide a basis for future comparisons of static bug detection with other bug finding techniques, such as manual and automated testing.

International Conference on Automated Software Engineering (ASE '18), September 3–7, 2018, Montpellier, France. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3238147.3240469

1 INTRODUCTION

Finding software bugs is an important but difficult task. For average industry code, the number of bugs per 1,000 lines of code has been estimated to range between 0.5 and 25 [21]. Even after years of deployment, software still contains unnoticed bugs. For example, studies of the Linux kernel show that the average bug remains in the kernel for a surprisingly long period of 1.5 to 1.8 years [8, 26]. Unfortunately, a single bug can cause serious harm, even if it has been subsisting for a long time without doing so as evidenced by examples of software bugs that have caused huge economic losses and even killed people [17, 28, 46].

Given the importance of finding software bugs, developers rely on several approaches to reveal programming mistakes. One approach is to identify bugs during the development process, e.g., through pair programming or code review. Another direction is testing, ranging from purely manual testing over semi-automated testing, e.g., via manually written but automatically executed unit tests, to fully automated testing, e.g., with UI-level testing tools. Once the software is deployed, runtime monitoring can reveal so far missed bugs, e.g., collect information about abnormal runtime

Tool	Bugs
Error Prone	8
Infer	5
SpotBugs	18
Total:	31
Total of 27 unique bugs	

Figure 4: Total number of bugs found by all three static checkers and their overlap.

https://software-lab.org/publications/ase2018_static_bug_detectors_study.pdf

Summary

- Linters are cheap, fast, but imprecise analysis tools
  - Can be used for purposes other than bug detection (e.g., style)
- Conservative analyzers can demonstrate the absence of particular defects
  - At the cost of false positives due to necessary approximations
  - Inevitable trade-off between false positives and false negatives
- The best QA strategy involves multiple analysis and testing techniques
  - The exact set of tools and techniques depends on context

S3D