End to End Quality with the Sonar Ecosystem and the Water Leak Metaphor

G. Ann Campbell
@GAnnCampbell | ann.campbell@SonarSource.com
@SonarLint | @SonarQube | @SonarSource
SonarLint
Leak Period
Quality Gate
20+ Languages

Static Analysis
What is Static Analysis?

Analyzing code, without executing it!
Detecting Bugs, Vulnerabilities, and Code Smells
Why use Static Analysis

Catch new problems ASAP

- the longer it takes to catch a bug, the more it costs
- no one writes perfect code every time
- rule description and precise issue location cut research time
Why use Static Analysis

Changing A might have added bugs in B

- peer review misses new issues in untouched code
- static analysis is machine-assisted code review; it looks at every file every time
Why use Static Analysis

Provide coaching

- language best practices
- team coding style
SonarSource’s Toolbox
Only two things are infinite, the universe and human stupidity, and I am not sure about the former.
Only two things are infinite, the universe and human stupidity, and I am not sure about the former.

Albert E.
Only two things are infinite, the universe and human stupidity, and I am not sure about the former.

Albert E.
Only two things are infinite, the universe and human stupidity, and I am not sure about the former.
Object myObject = new Object();

if(a) { myObject = null; }

...

if(!a) { ... }

else { myObject.toString(); }
Object myObject = new Object();

if (a) {
    myObject = null;
}

...

if (!a) {
    ...
}

else {
    myObject.toString();
} //NPE
Object myObject = new Object();

if(a) { myObject = null; }

else { myObject.toString(); } //NPE
Object myObject = new Object();

if (a) {
    myObject = null;
}
...
if (!a) {
    ...  
}
else {
    myObject.toString();  //NPE
}
Beyond Semantic: Symbolic Execution

... 

if( !a ) {
    ...
}

} else {

    myObject.toString(); // NPE

}
Beyond Semantic: Symbolic Execution

Program State#1
myObject != null
a = false

Program State#2
myObject = null
a = true

... if(!a) {

} else {

myObject.toString(); // NPE

}
Beyond Semantic: Symbolic Execution

```java
... if (!a) {
    ...
} else {
    myObject.toString(); // NPE
}
```
What is Static Analysis?

Analyzing code, without executing it. by (symbolically) executing all possible paths!
Symbolic Execution Almost Everywhere

- SonarAnalyzers for C#, C/C++, Java, and JS
  - Dereferences of Null Pointers
  - Unconditionally True/False (sub)conditions
  - Division by zero
  - Resource leaks
    - Unclosed resources (Java)
    - Unreleased memory (C/C++)
  - Double free (C/C++)
Fewer slides, more code!
Full Cycle

IDE

SonarQube

Gradle

Full Analysis

TC

npm
Full Cycle

IDE

SonarQube
Fix the Leak

✔ SonarLint
Leak Period
Quality Gate
Reimbursing the Debt
This is Hard

- Total amount of Technical Debt can be depressing
- How to get a budget to fix old Technical Debt?
- Risk of injecting functional regression
- This is not fun!
## Project Homepage: Leak Period

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leak Period: since 6.5</td>
<td></td>
</tr>
<tr>
<td>started 2 months ago</td>
<td></td>
</tr>
<tr>
<td>New Bugs</td>
<td>5</td>
</tr>
<tr>
<td>New Vulnerabilities</td>
<td>0</td>
</tr>
<tr>
<td>New Debt</td>
<td>25d</td>
</tr>
<tr>
<td>New Code Smells</td>
<td>687</td>
</tr>
<tr>
<td>Coverage on</td>
<td>83.6%</td>
</tr>
<tr>
<td>3.3k New Lines to Cover</td>
<td></td>
</tr>
<tr>
<td>Duplications on</td>
<td>0.0%</td>
</tr>
<tr>
<td>44k New Lines</td>
<td></td>
</tr>
</tbody>
</table>
Fix the Leak

- SonarLint
- Leak Period
- Quality Gate
# Quality Gate

**SonarQube way**

## Conditions

Only project measures are checked against thresholds. Sub-projects, directories and files are ignored. [More]

<table>
<thead>
<tr>
<th>Metric</th>
<th>Over Leak Period</th>
<th>Operator</th>
<th>Warning</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverage on New Code</td>
<td>Always</td>
<td>is less than</td>
<td></td>
<td>80.0%</td>
</tr>
<tr>
<td>Maintainability Rating on New Code</td>
<td>Always</td>
<td>is worse than</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Reliability Rating on New Code</td>
<td>Always</td>
<td>is worse than</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Security Rating on New Code</td>
<td>Always</td>
<td>is worse than</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>
Project Homepage: Quality Gate

Quality Gate: Failed

- Coverage on New Code: 83.6% is less than 85.0%
- Reliability Rating on New Code: is worse than A
Quality Gate

Passed

Warning

Skipped Unit Tests
Over Leak Period
is greater than 0

Failed

Coverage on New Code
is less than 85.0%

Reliability Rating on New Code
is worse than A
Fix the Leak

- SonarLint
- Leak Period
- Quality Gate