



# Static Code Analysis



Dario Campagna



# Static Code Analysis

Static code (or program) analysis is the analysis of computer software that is performed without actually executing programs.

- Identification of coding errors
  - Individuation of duplicated code
  - Computation of software metrics
  - Formal methods (e.g., Hoare logic, Model Checking)

```
59 //is the element inside the visible window
60 var a = w.scrollLeft();
61 var b = w.scrollTop();
62 var o = t.offset();
63 var x = o.left;
64 var y = o.top;
65
66 var ax = settings.accX;
67 var ay = settings.accY;
68 var th = t.height();
69 var wh = w.height();
70 var tw = t.width();
71 var ww = w.width();
72
73 if (y + th + ay >= b &&
74     y <= b + wh + ay &&
75     x + tw + ax >= a &&
76     x <= a + ww + ax) {
77
78     //trigger the custom event
79     if (!t.appeared) t.trigger('appear', settings.data);
80
81 } else {
82
83     //it scrolled out of view
84     t.appeared = false;
85 }
86
87 //create a modified fn with some additional logic
88 var modifiedFn = function() {
89
90     //mark the element as visible
91     t.appeared = true;
92
93     //is this supposed to happen only once?
94     if (settings.one) {
95
96         //remove the check
97         $fn.unbind('scroll', check);
98         $fn.appear.checks--;
99     }
100 }
```

# Automated Tools

Automated tools that perform static code analysis can identify issues that reduce code readability, maintainability, quality.

- IDE code inspection functionalities (e.g., IntelliJ Idea)
- [SonarQube](#)

The screenshot shows a SonarQube Quality Gate report. It has a green header bar with the text "Quality Gate Passed" and "All conditions passed". Below this, there are four sections: RELIABILITY (0 Bugs), SECURITY (0 Vulnerabilities, 1 Hotspots), MAINTAINABILITY (4 Code Smells, 5 Debt min), and a detailed code review section. The code review section shows a snippet of Java code with several annotations: `246 if (Provider.class == roleTypeClass) {`, `247 Type providedType = ReflectionUtils.getLastTypeGenericArgument(dependencyDescriptor);`, `248 2 class providedClass = 1 ReflectionUtils.getTypeClass(providedType);`, `249 250 if (this.componentInstance.hasComponent(providedType, dependencyDescriptor);`, `251 || 3 providedClass.isAssignableFrom(List.class) || providedClass.isA`. A red callout box highlights a warning at line 250: "A 'NullPointerException' could be thrown; 'providedClass' is nullable here." It also shows a "Bug" icon and "Major" status. The code review continues with `252 continue;` and `253 }`.