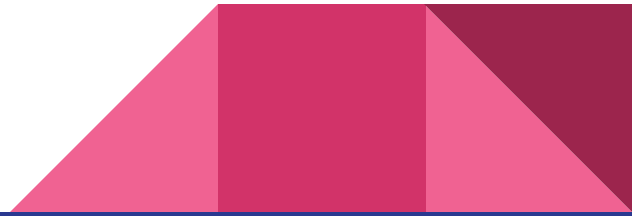
Statical and Dynamical Software Analysis

Sergii Sosnytskyi

Software Analysis Overview

- It is a fundamental question in every engineering discipline - Will our design work in reality as we expected?
- The answer comes from analyzing designs using knowledge about nature that will carry out the designs.
- The same question applies to computer software.



Program analysis versus other engineering areas

Name	**Computing area**	**Other engineering areas**
Object	Software	Machine/building/circuit/chemical process design
Execution	Computer	Nature
Question	Work as intended	Work as intended
Knowledge	Program analysis	Thermodynamic equations, and other principles

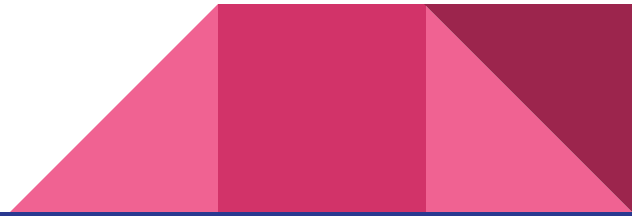
Areas to Analyze

- **Domain-specific analyses** - certain analyses are aimed at specific families of programs
- **Non-domain-specific analyses** - some analyses are designed without focus on a particular family of programs of the target language
- **Program-level** analyses are run on the source code of programs (e.g., written in C or Java) or on executable program binaries and typically involve a front end similar to a compiler's that constructs the syntax trees of programs from the program source or compiled files
- **Model-level** analyses consider a different input language that aims at modeling the semantics of programs

Families of Program Analysis Techniques

Testing: Checking a Set of Finite Executions

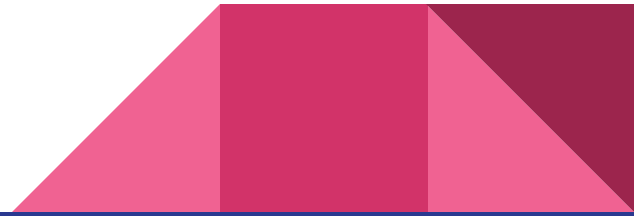
- Easy to automate
- Not robust
- Complete since a failed testing run will produce an execution that is incorrect



Families of Program Analysis Techniques

Assisted Proof: Relying on User-Supplied Invariant

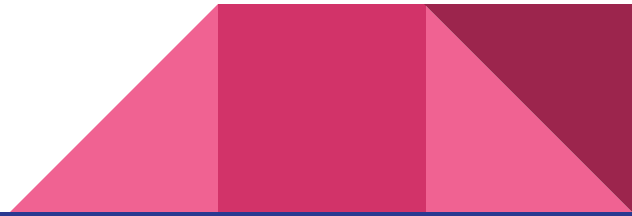
- They are not fully automatic and often require the most tedious logical arguments to come from the human user.
- In practice, they are robust with respect to the model of the program semantics used for the proof, and they are also complete up to the abilities of the proof assistant to verify proofs.



Families of Program Analysis Techniques

Model Checking: Exhaustive Exploration of Finite Systems

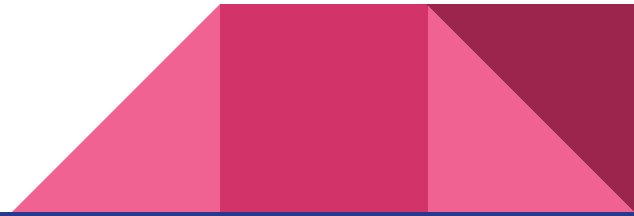
- Automatic
- Robust and complete with respect to the model



Families of Program Analysis Techniques

Static Analysis: Automatic, Robust, and Incomplete Approach

- It is automatic
- It produces robust results, as they compute a conservative description of program behaviors, using a limited set of logical properties.
- It is generally incomplete because they cannot represent all program properties and rely on algorithms that enforce termination of the analysis even when the input program may have infinite executions.

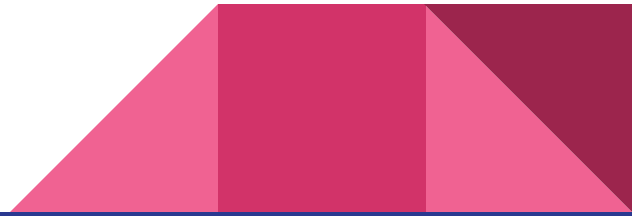


An overview of program analysis techniques

Techniques	Automatic	Robust	Complete	Object	Execution
Testing	Yes	No	Yes	Program	Dynamic
Assisted proving	No	Yes	Yes/No	Model	Static
Model checking of finite-state model	Yes	Yes	Yes	Finite Model	Static
Model checking at program level	Yes	Yes	No	Program	Static
Conservative static analysis	Yes	Yes	No	Program	Static
Bug finding	Yes	No	No	Program	Static

SQALE Model Indexes

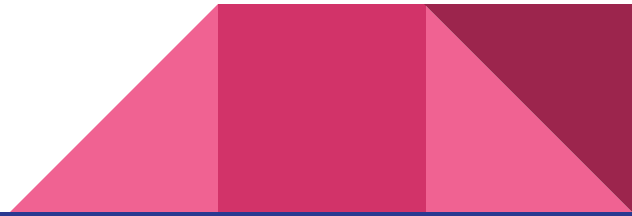
- Testability Index : STI
- Reliability Index : SRI
- Changeability Index : SCI
- Efficiency Index : SEI
- Security Index : SSI
- Maintainability Index : SMI
- Portability Index : SPI
- Reusability Index : SRuI



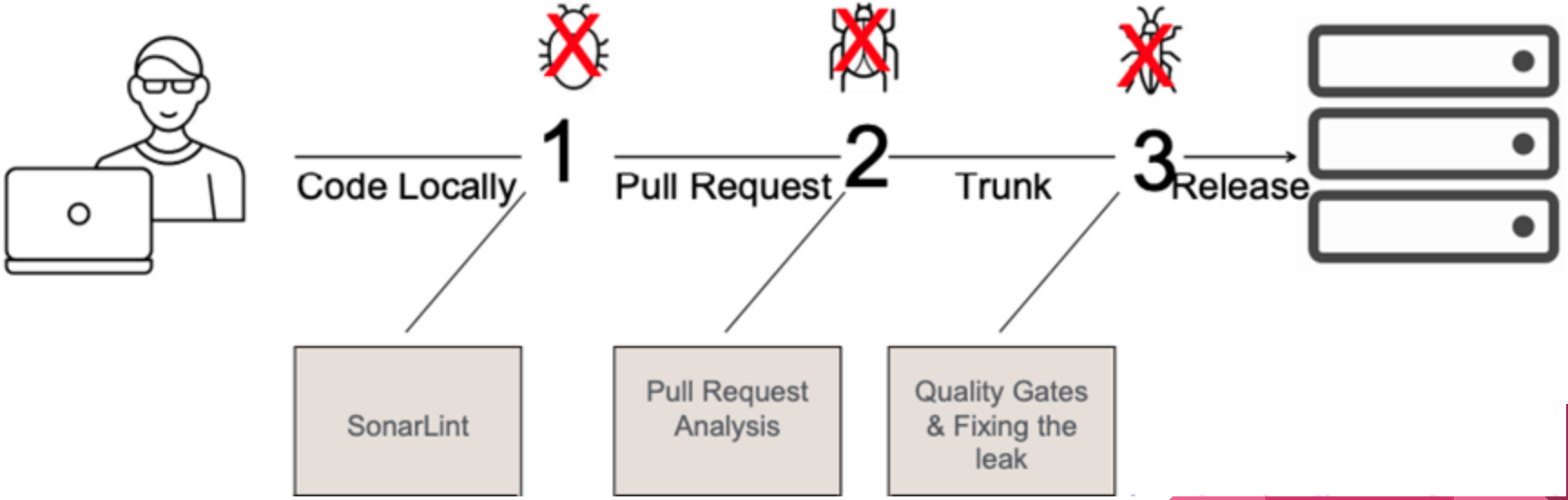
Continuous Code Analysis with the SonarQube

The SonarQube uses an evolved SQALE model.
Bugs, Vulnerabilities and Code Smells are main metrics to measure.

- Bugs: Code that is demonstrably wrong or highly likely to yield unexpected behaviour.
- Vulnerabilities: Code that is potentially vulnerable to exploitation by hackers.
- Code Smells: Will confuse maintainers or give them pause. Not only ratings, but also approximate remediation efforts.



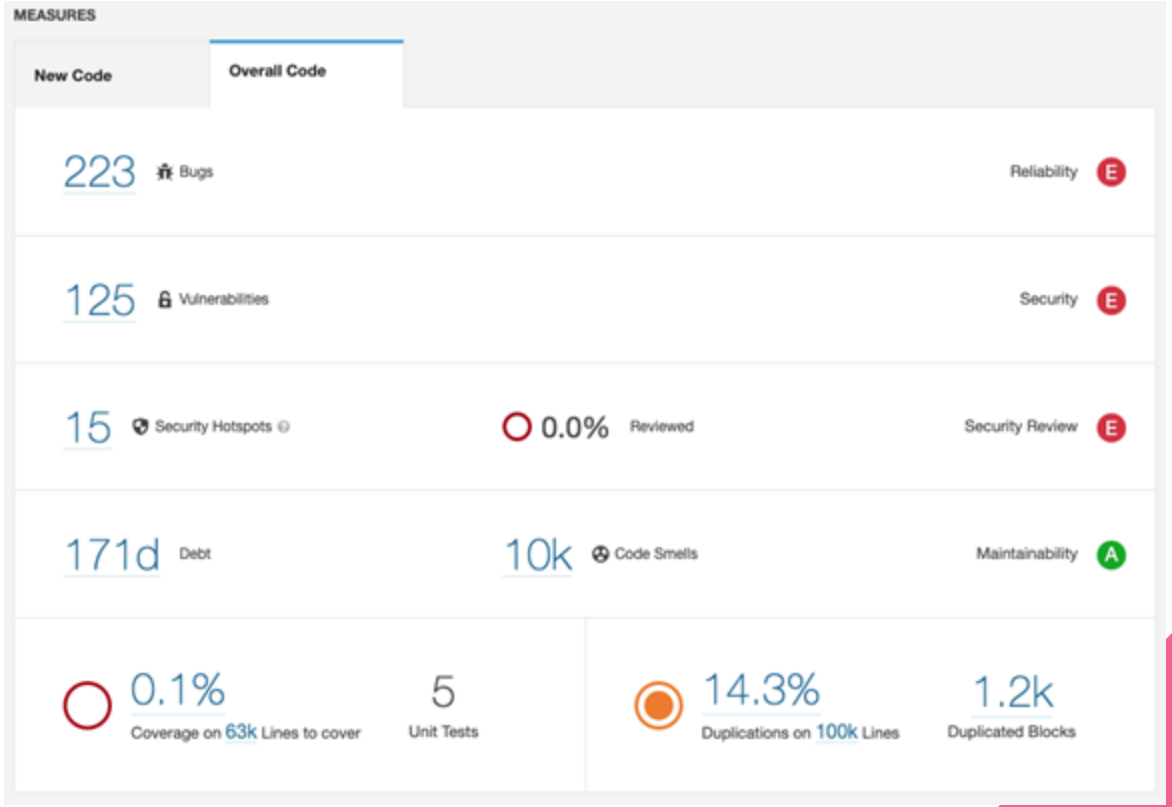
Three Lines of Analysis



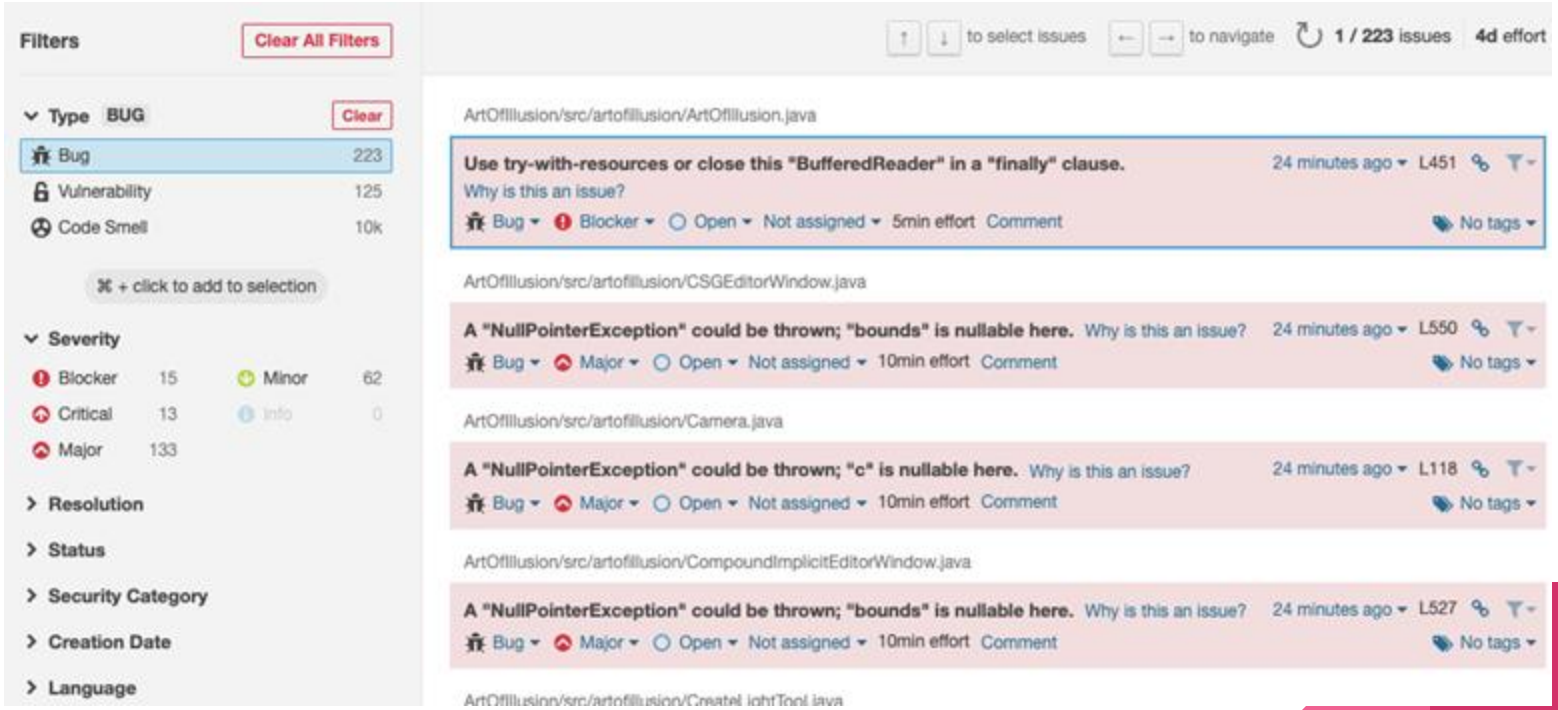
Importance of Unit Testing

```
import org.junit.Test;

import static org.junit.Assert.*;

 @Test
    public void helloWorldCheck() {
        System.out.println("* UtilsJUnit4Test: test method 1 - helloWorldCheck()");
        assertEquals("Hello, world!", Utils.concatWords("Hello", ", ", "world", "!"));
    }
```

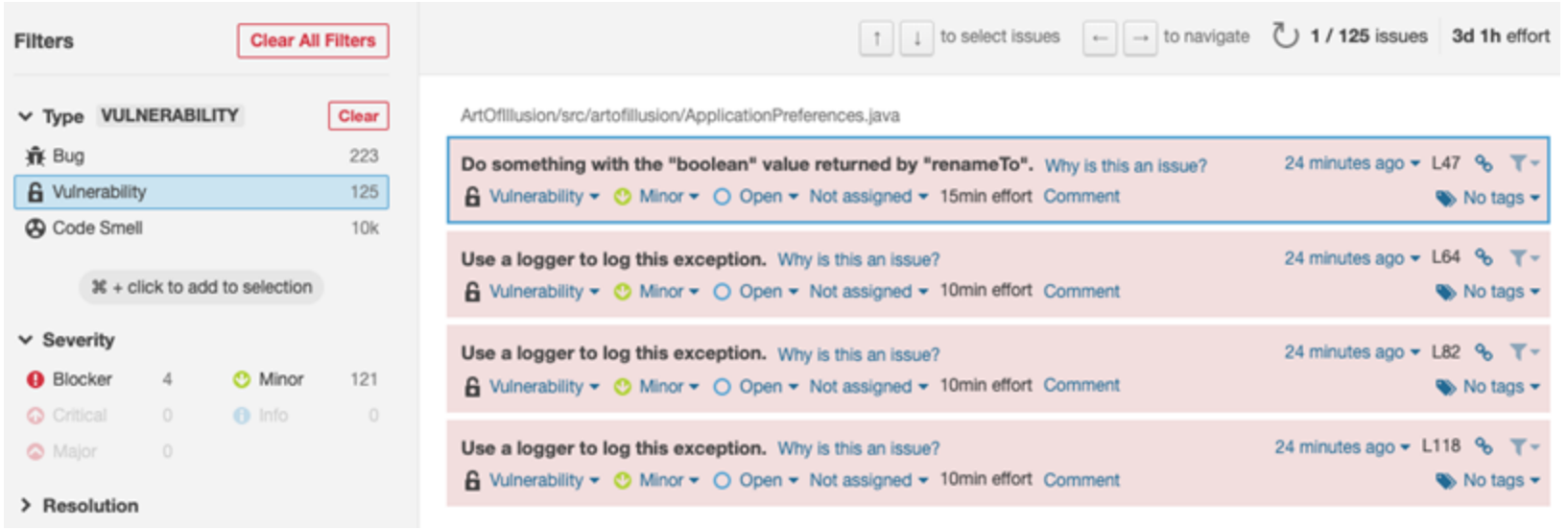
SonarQube Overall Code Analysis Results



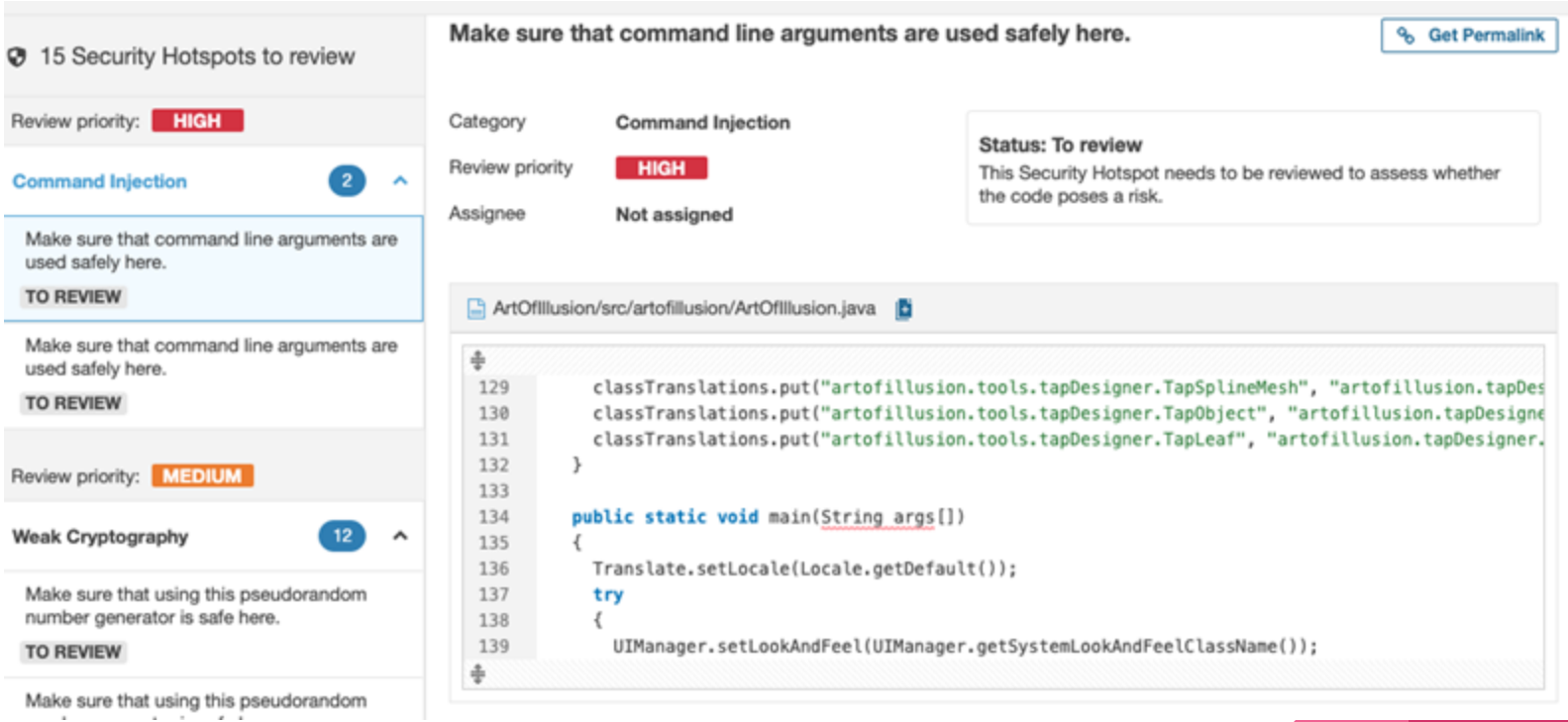
SonarQube Bug Analysis



SonarQube Vulnerabilities Analysis



SonarQube Security Hotspots Analysis



Conclusions

- No technique can provide fully automatic, robust, and complete analyses
- Testing sacrifices robustness
- Assisted proving is not automatic
- Model-checking approaches can achieve robustness and completeness only with respect to finite models
- Static analysis gives up completeness
- Bug finding is neither robust nor complete.
- Nevertheless, combining a number of techniques with a model like SQALE it is feasible to build a comprehensive analysis approach.