SonarQube Training

Session #1 - Overview

A journey in the land of code quality and security

Agenda

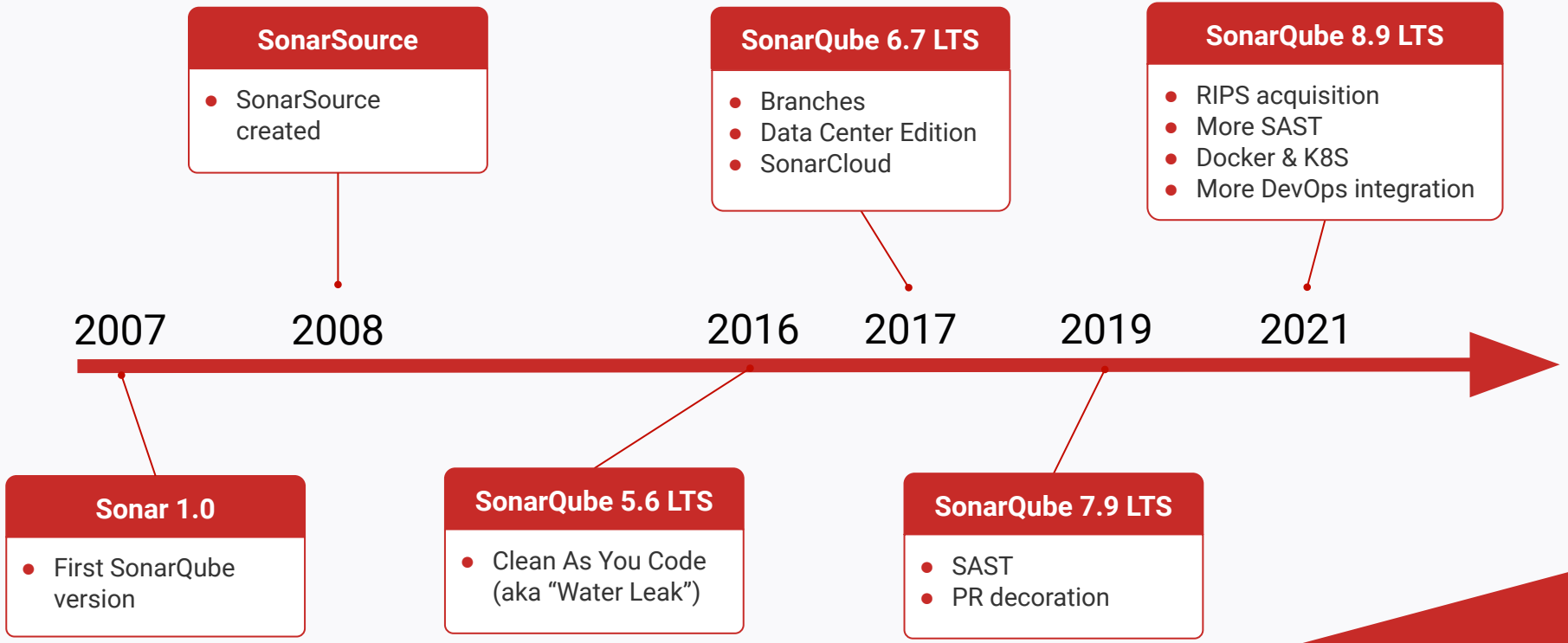
Session #1 - Overview

- SonarSource's approach to Code Quality and Security
- The *Clean As You Code* paradigm
- Security overview
- Demo of SonarSource's own production SonarQube platform
- Demo of SonarLint
- Platform walkthrough: Show your production platform, how you configured and used it so far
- Q&A

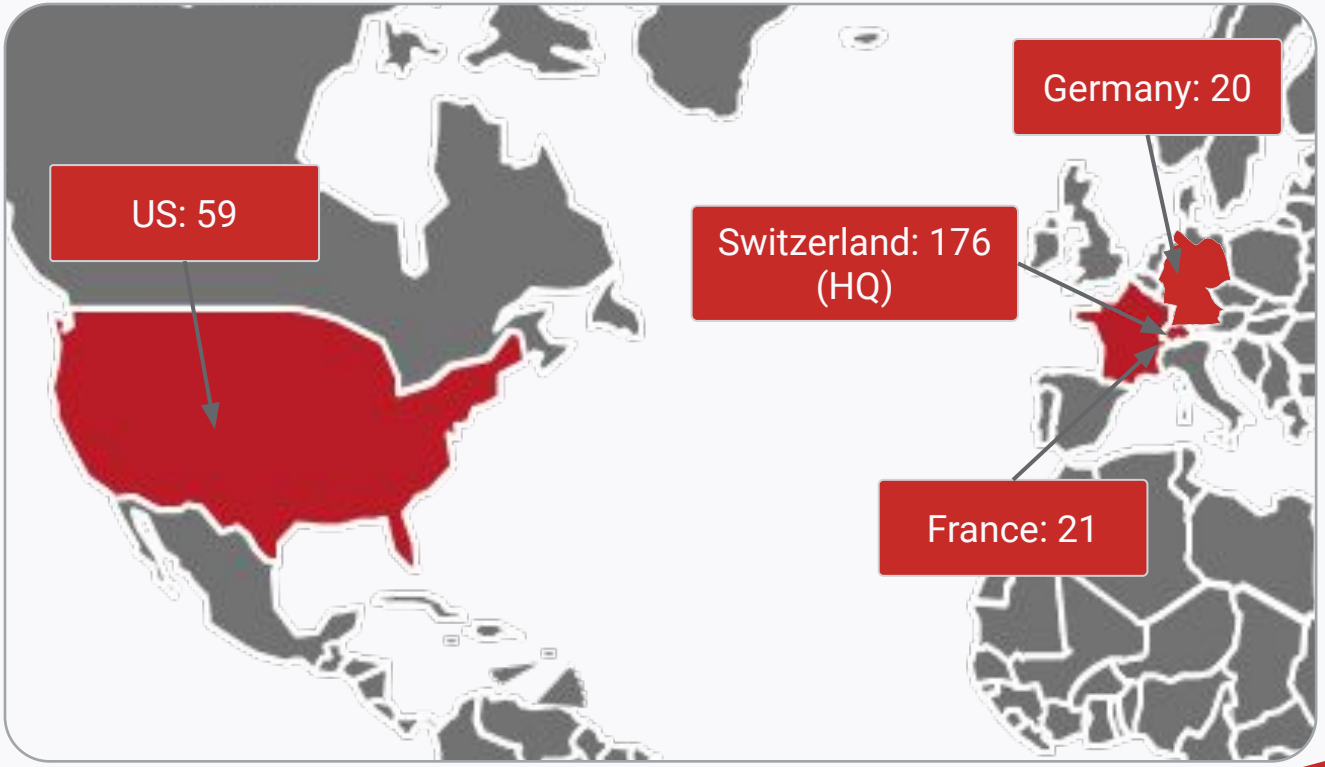
SonarSource’s approach

To code quality and security

History


SonarSource
SonarSource created
SonarQube 6.7 LTS
Branches
Data Center Edition
SonarCloud
SonarQube 8.9 LTS
RIPS acquisition
More SAST
Docker & K8S
More DevOps integration
2007
2008
2016
2017
2019
2021
Sonar 1.0
First SonarQube version
SonarQube 5.6 LTS
Clean As You Code (aka “Water Leak”)
SonarQube 7.9 LTS
SAST
PR decoration

Our Team



A lively community

	Today	Yesterday	Last 7	Last 30
Pageviews	9.1k	30.7k	140k	609k
User Visits	133	311	1.6k	7.0k
Time to first response	1	5	26	37
Likes	17	41	173	516
Flags	0	1	1	2
User-to-User (with r…	4	4	43	129

300,000+ live platforms

Users per Trust Level

0 7.4k | 1 3.2k | 2 69 | 3 1 | 4 177

https://community.sonarsource.com

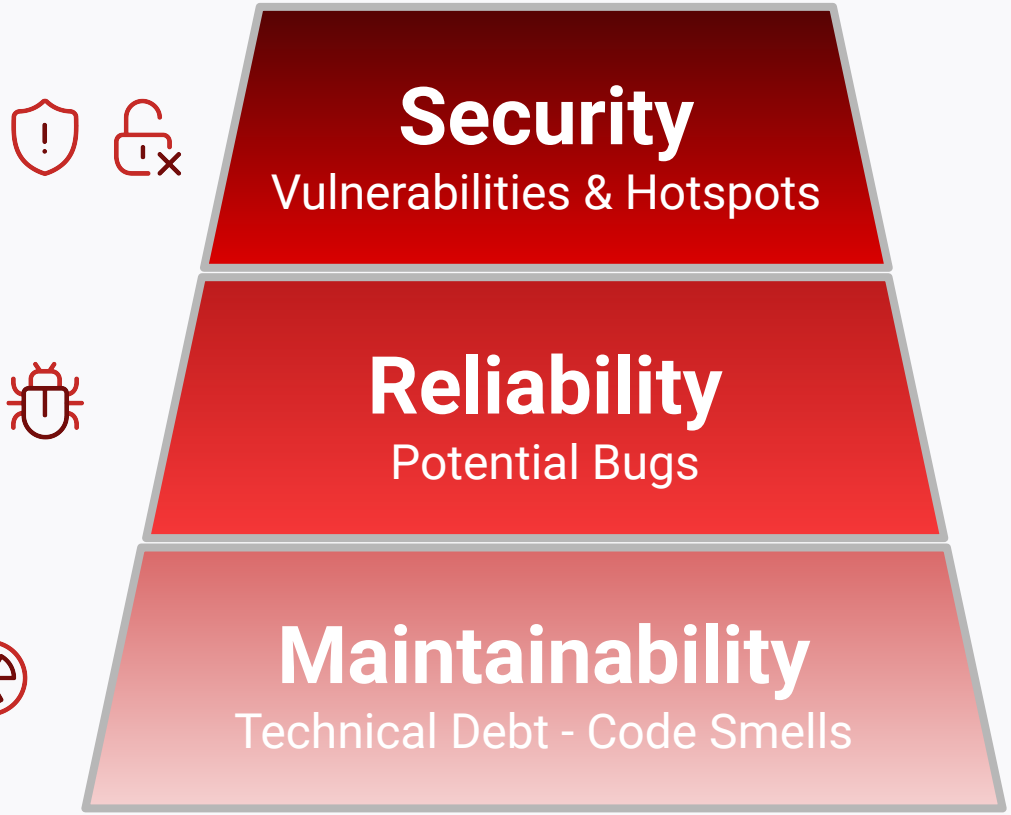
Software (and SonarQube) is everywhere

Customer diversity is good for our rules

Our mission

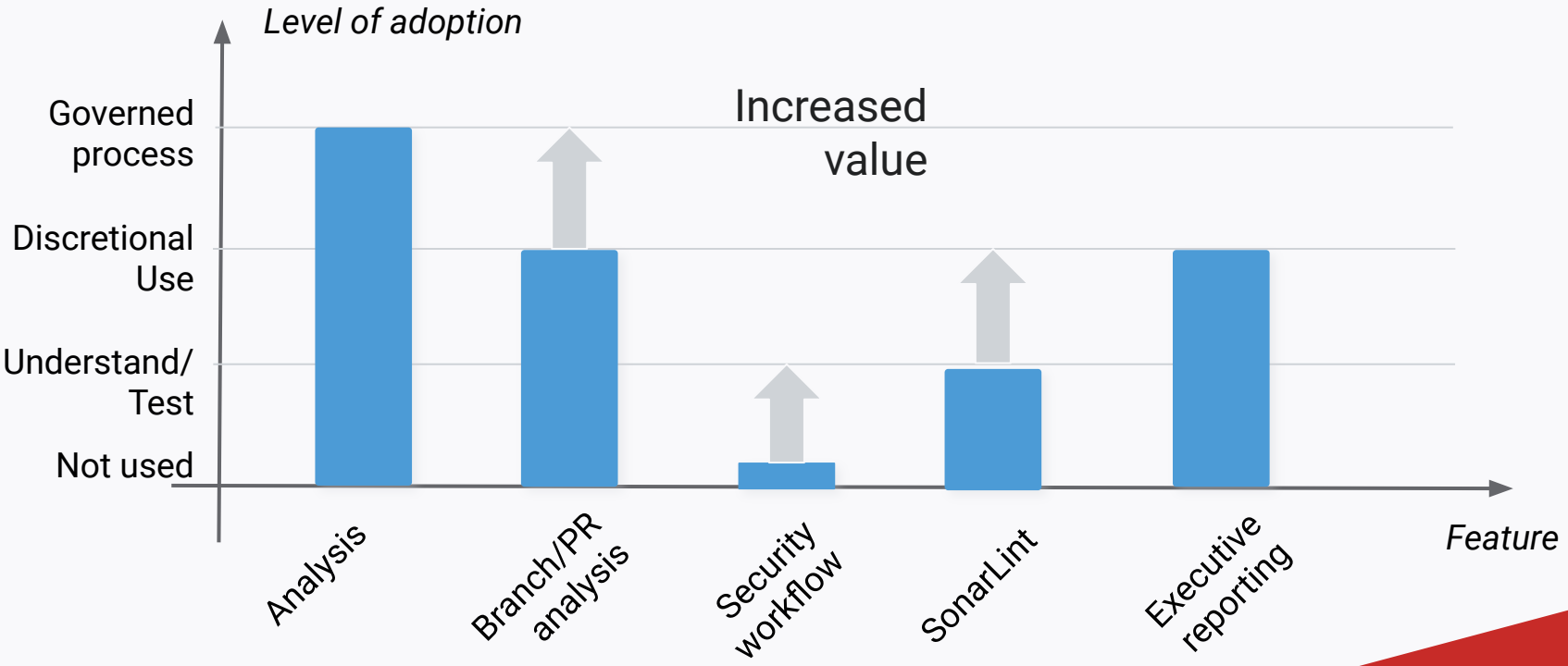
Every developer and development team uses SonarSource for code quality and security

SonarQube Quality Model



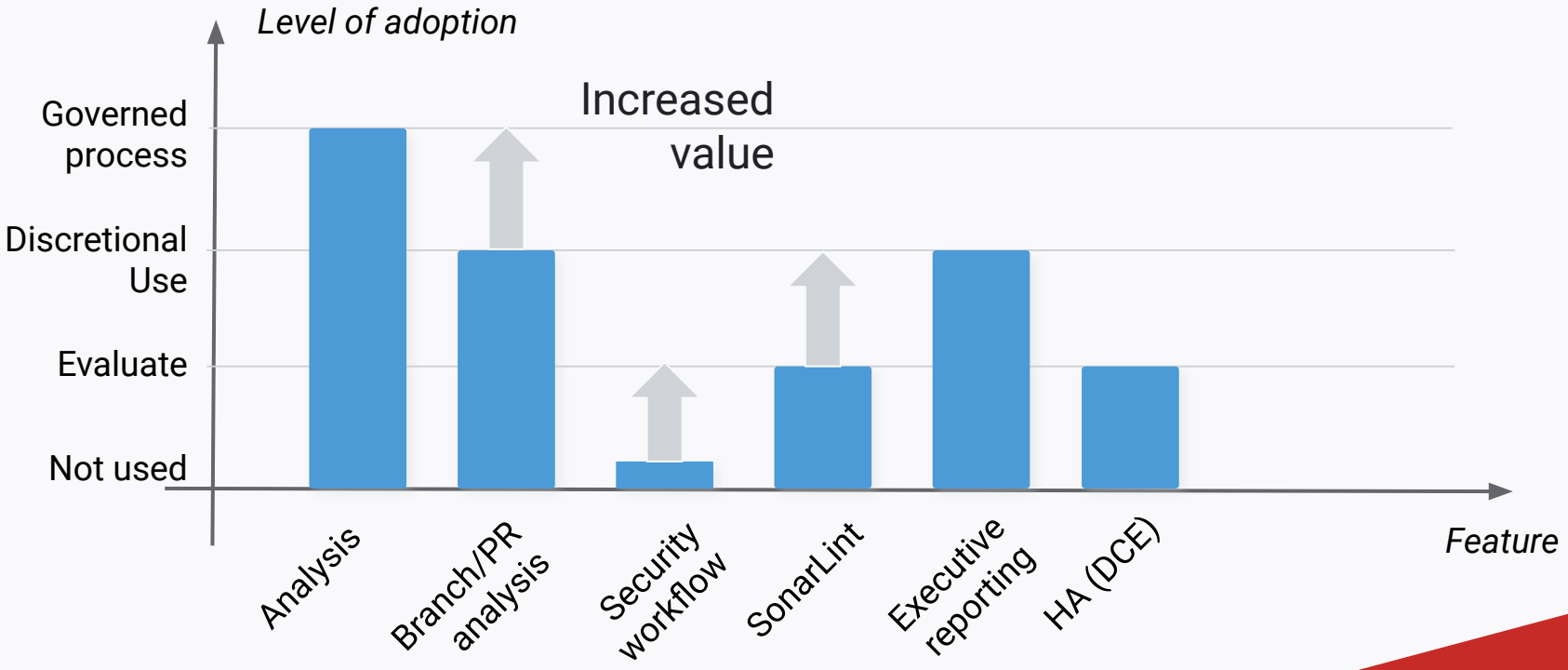
Getting the most of SonarQube

What you get from SonarQube depends on what you make of it



Getting the most of SonarQube

What you get from SonarQube depends on what you make of it

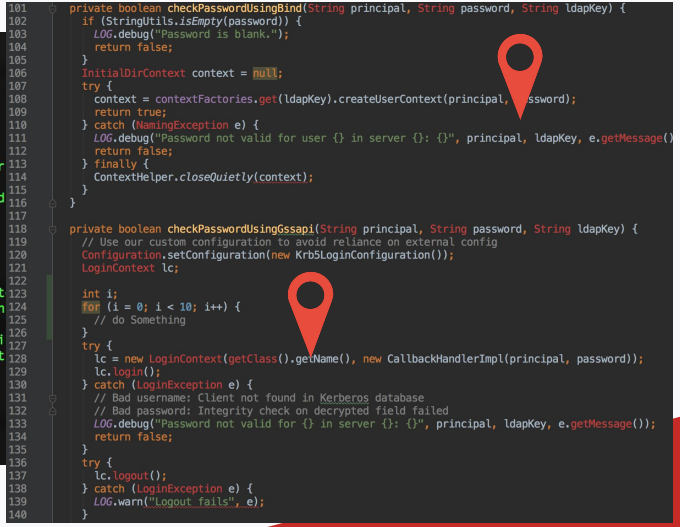


Key Feature - Quality Gate

Key Feature - Clean As You Code

- Track **new** issues
- Compute metrics on **new** code
  - Coverage
  - Duplications

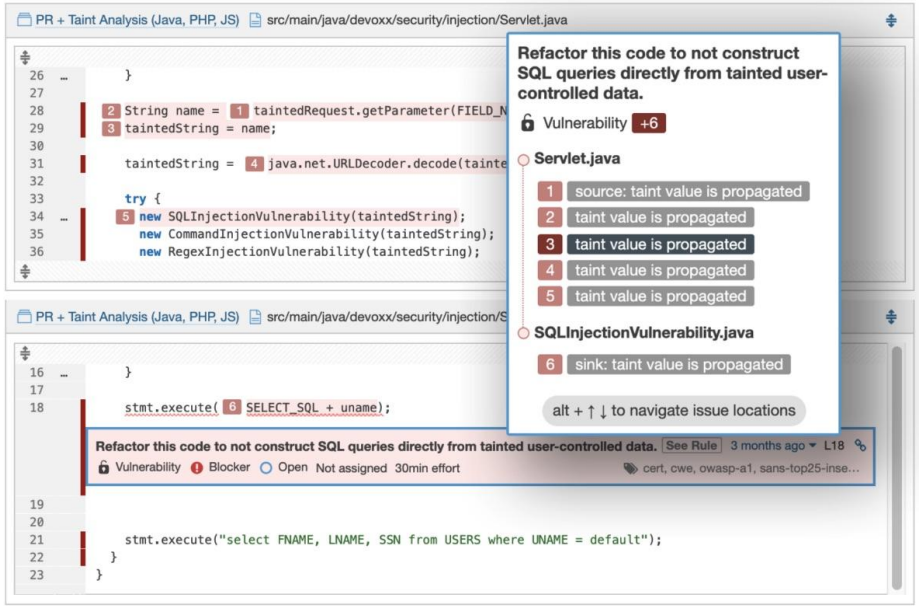
New Code



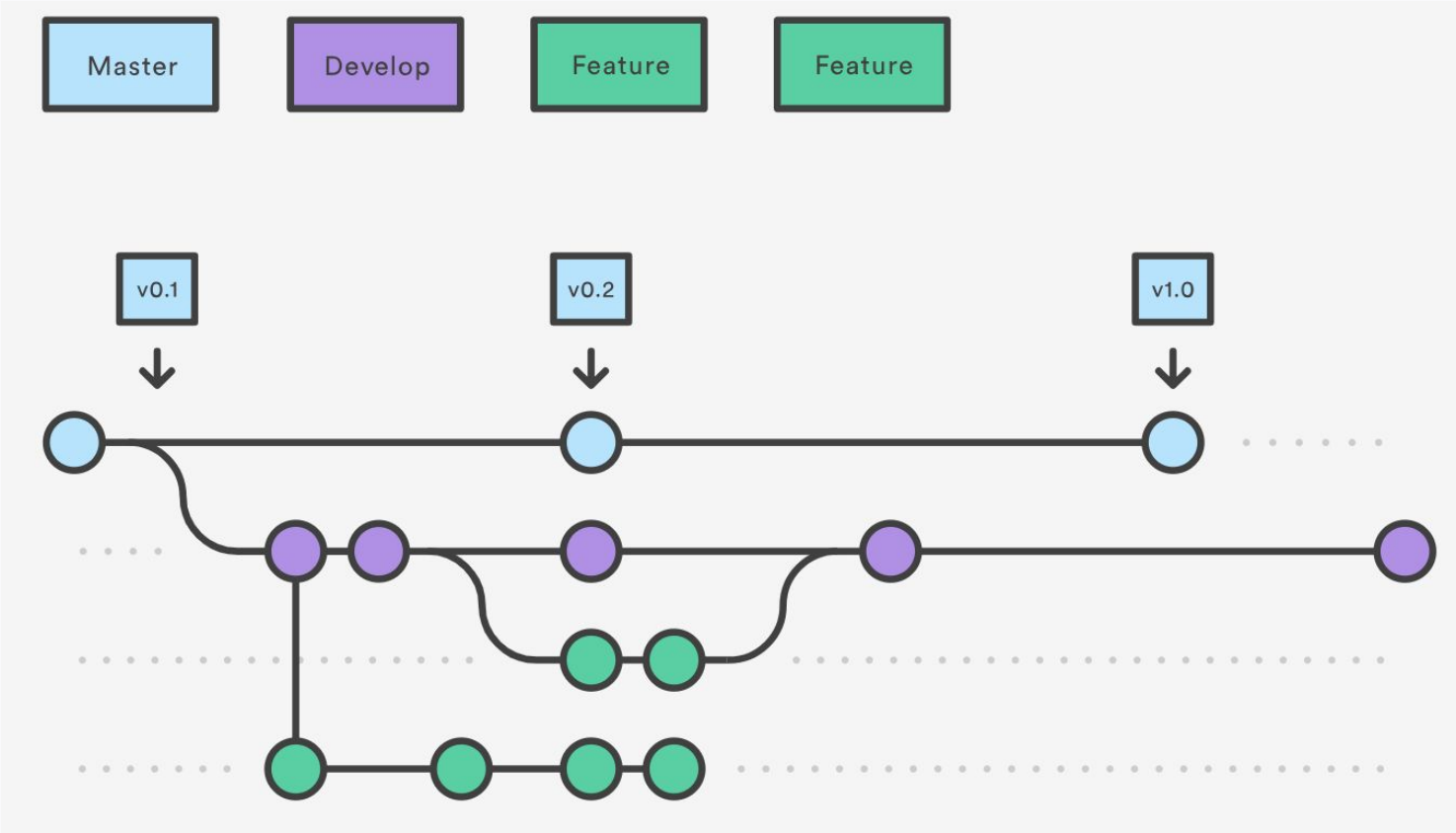
Legacy Code

Key Feature - SAST

- Detect Injection vulnerabilities
- SonarQube UI shows tainted data flow
- OWASP / CWE reporting
- Other vulnerabilities (data exposure, poor practices)
- Security Hotspots



Key Feature - Branch & PR support



Key Feature - Documented rules

Vulnerability Critical cert, cwe, owasp-a6, sans-top25-porous Available Since June 13, 2018 SonarAnalyzer (Java)
Constant/issue: 2min

The Advanced Encryption Standard (AES) encryption algorithm can be used with various modes. Some combinations are not secured:

- Electronic Codebook (ECB) mode: Under a given key, any given plaintext block always gets encrypted to the same ciphertext block. Thus, it does not hide data patterns well. In some senses, it doesn't provide serious message confidentiality, and it is not recommended for use in cryptographic protocols at all.
- Cipher Block Chaining (CBC) with PKCS#5 padding (or PKCS#7) is susceptible to padding oracle attacks.

In both cases, Galois/Counter Mode (GCM) with no padding should be preferred.

This rule raises an issue when a `Cipher` instance is created with either ECB or CBC/PKCS5Padding mode.

Noncompliant Code Example

```
Cipher c1 = Cipher.getInstance("AES/ECB/NoPadding"); // Noncompliant
Cipher c2 = Cipher.getInstance("AES/CBC/PKCS5Padding"); // Noncompliant
```

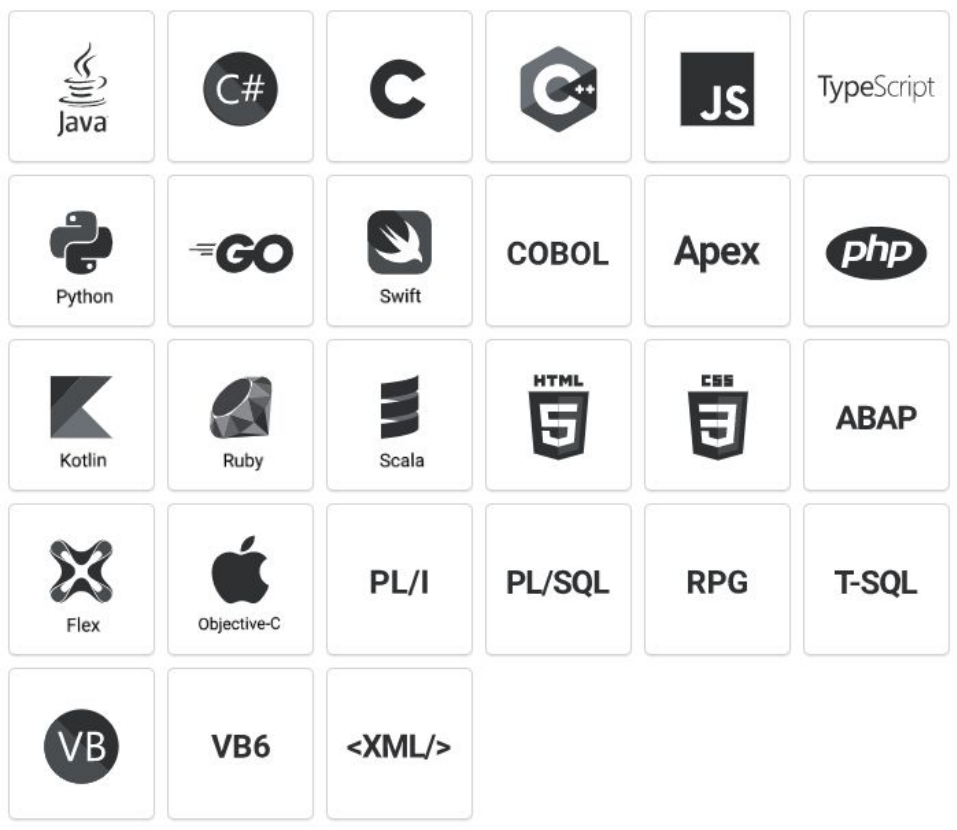
Compliant Solution

```
Cipher c = Cipher.getInstance("AES/GCM/NoPadding");
```

See

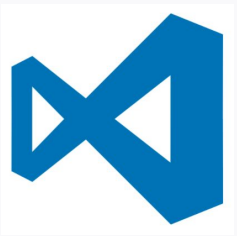
- MITRE, CWE-327 - Use of a Broken or Risky Cryptographic Algorithm
- OWASP Top 10 2017 Category A6 - Security Misconfiguration
- CERT, MSC61-J. - Do not use insecure or weak cryptographic algorithms
- SANS Top 25 - Porous Defenses

Key Feature - Language coverage

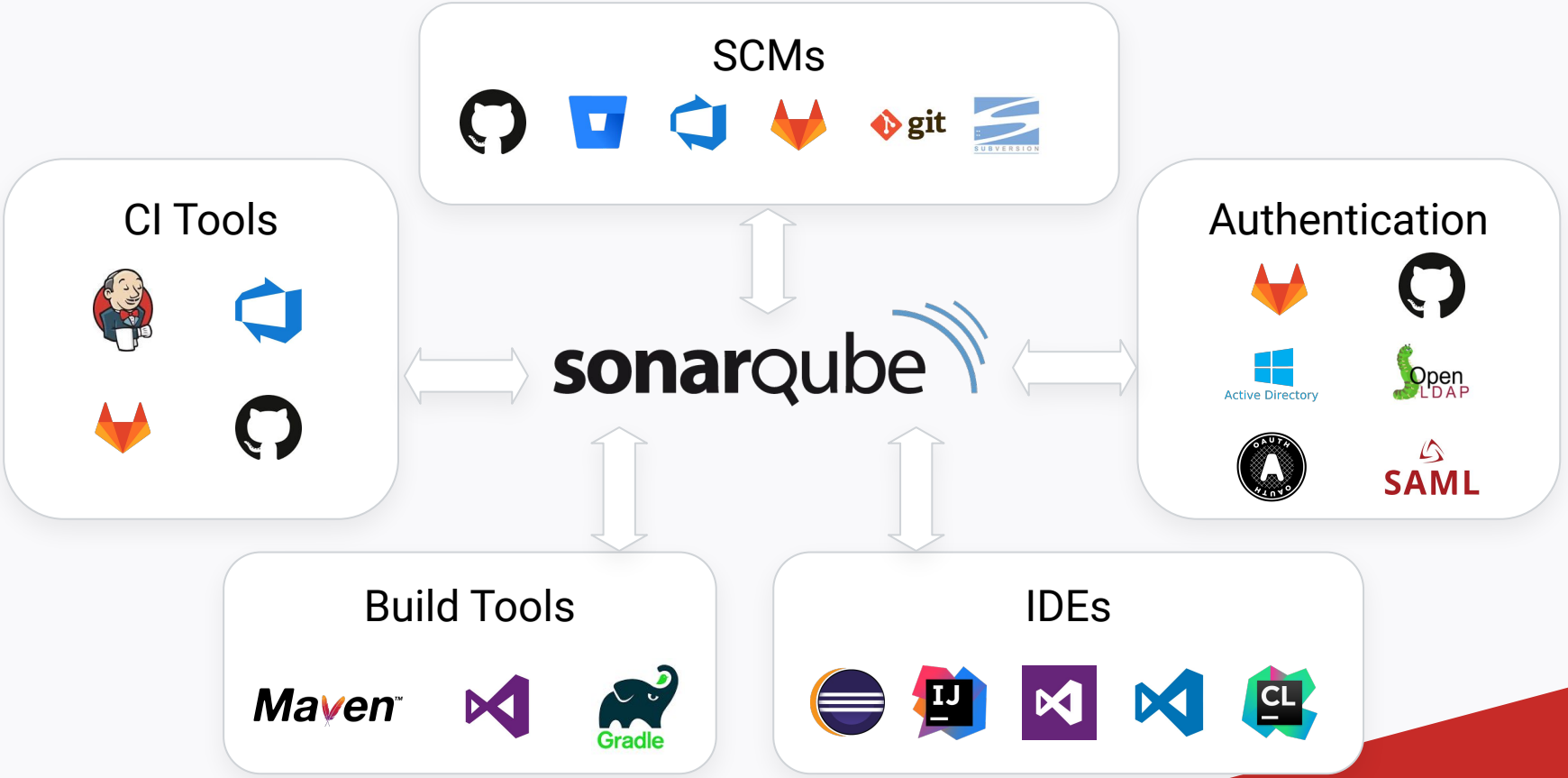


Key Feature - In IDE analysis

sonarlint



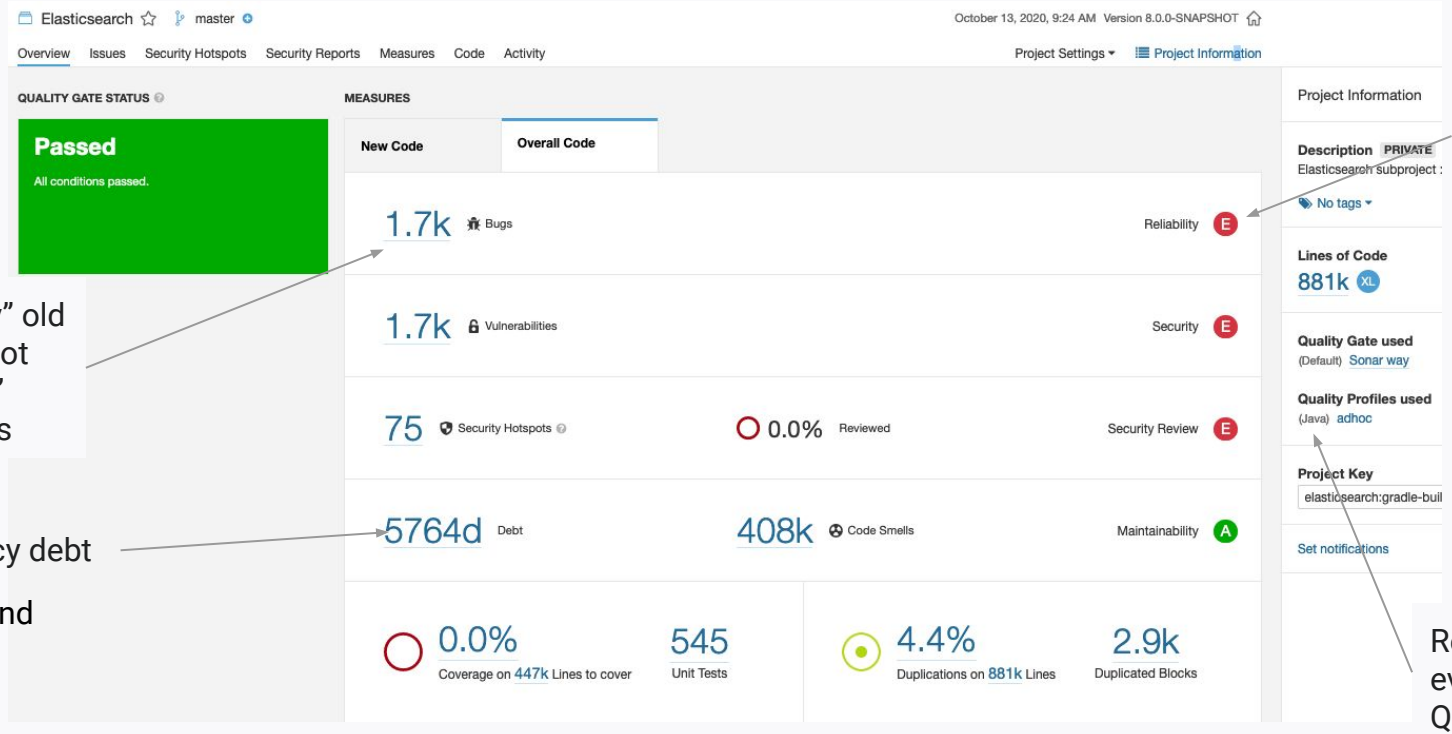
Key Feature - DevOps integration



Clean As You Code

Fundamentals

Caveats of traditional approaches



Fix “easy” old issues, not “difficult” new ones

Huge legacy debt

Where to find budget?

Risk of functional regression

Developer pushback

- Not my code
- Too many issues
- Boring!

Resistance to evolution of Quality Profiles

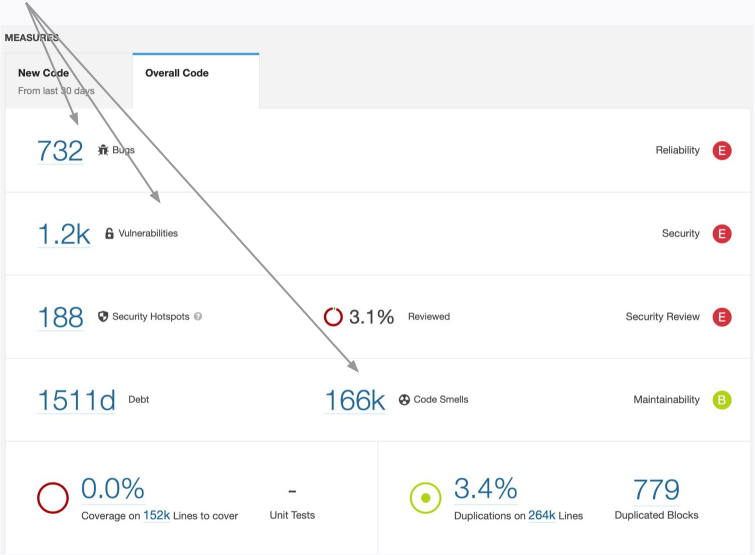
What is best practice?

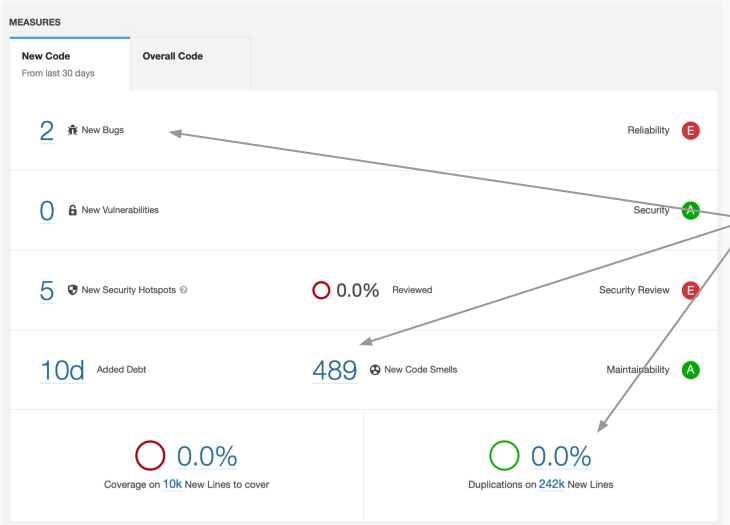
You take personal responsibility for *your* code

Huge legacy debt

Avoid being overwhelmed

A lot of work here

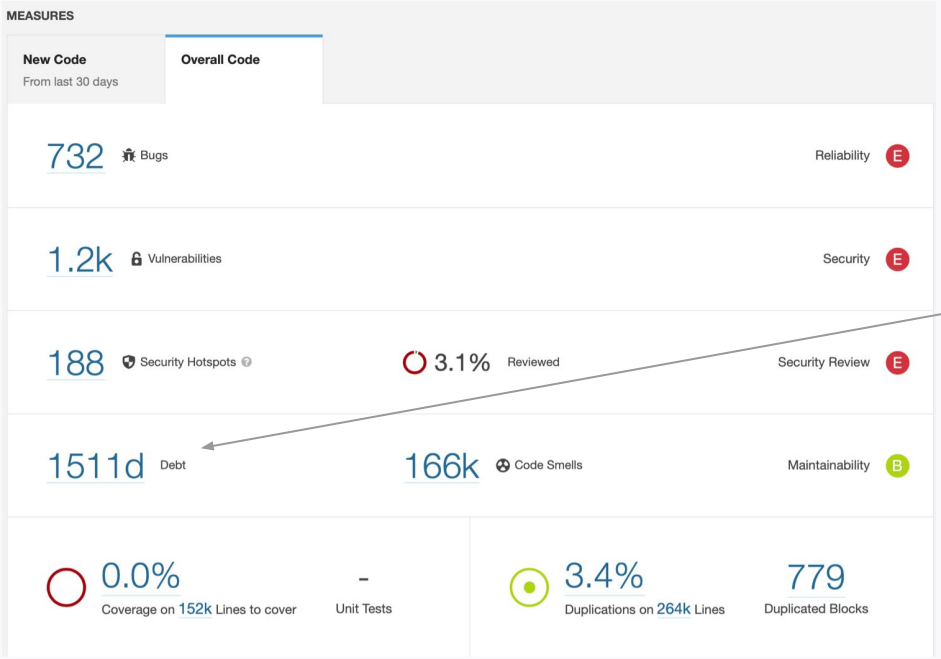




Not so much there

Huge legacy debt

Get rid of budget considerations



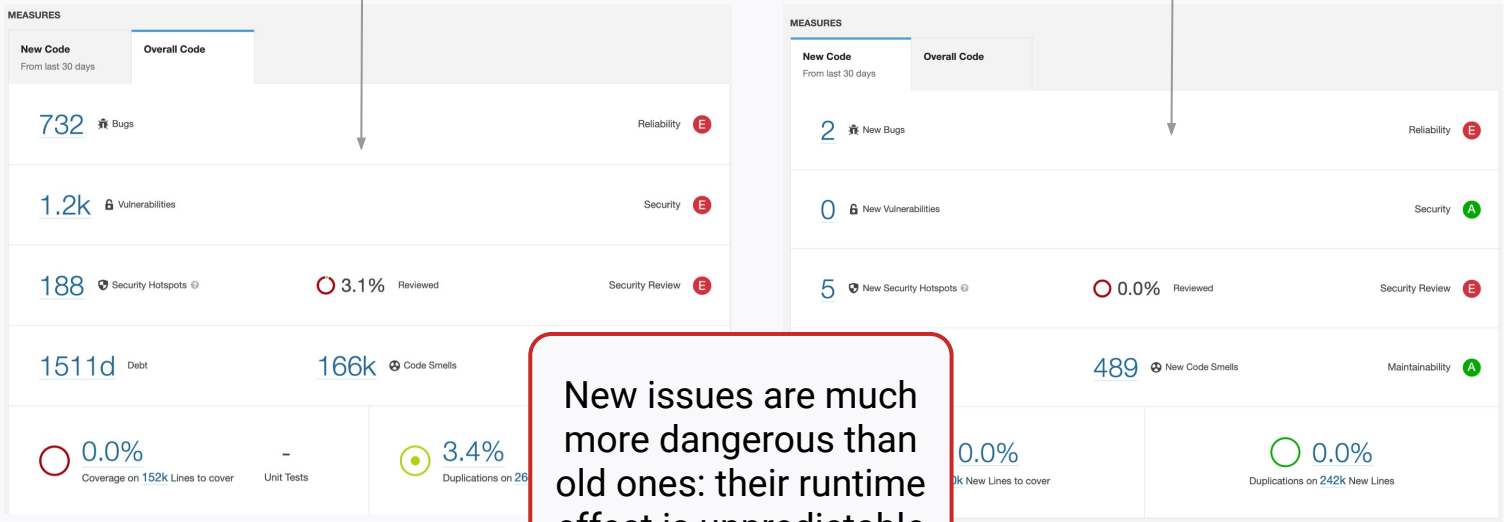
1511 days =
between $400K
and $1200K

Compensation

SonarQube will not let you fix an old issue to compensate for a new one

You **may** fix some of these

But you **MUST** fix these



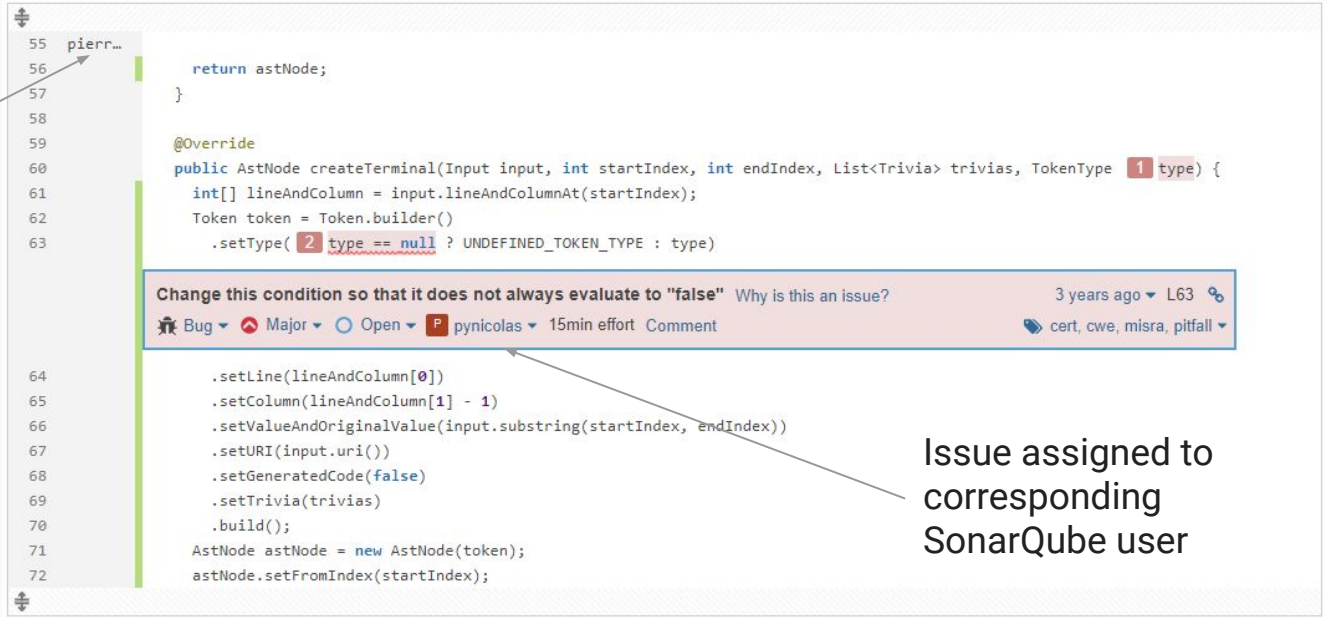
...also, be careful of functional regression here!

New issues are much more dangerous than old ones: their runtime effect is unpredictable

Developer Push Back

Issues assigned to developers that creates them

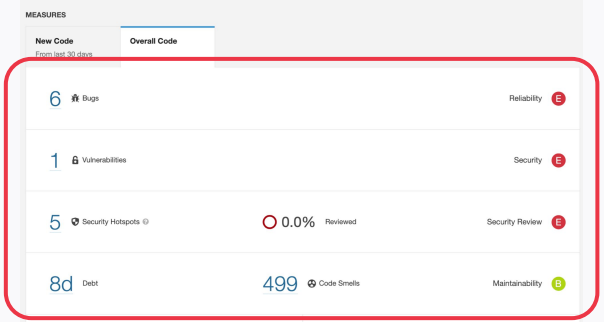
Blame data determines who added or modified this code

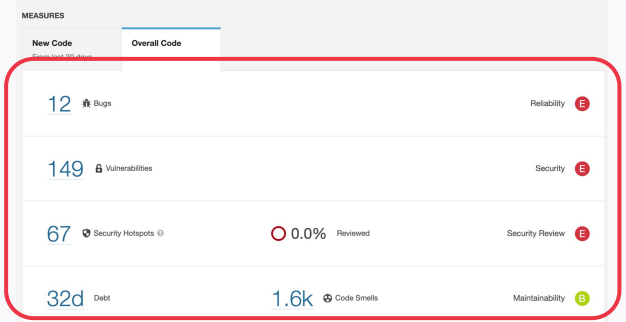


Issue assigned to corresponding SonarQube user

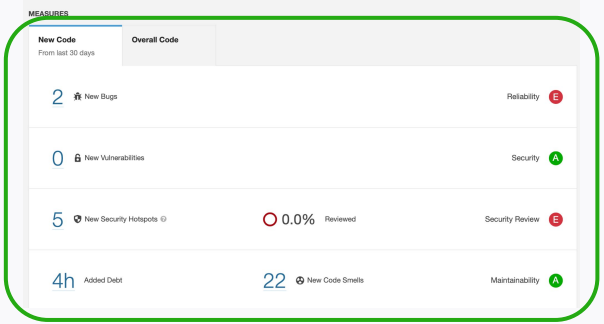
Quality Profile evolution

Making it easier to raise the bar or use new rules from new SQ versions

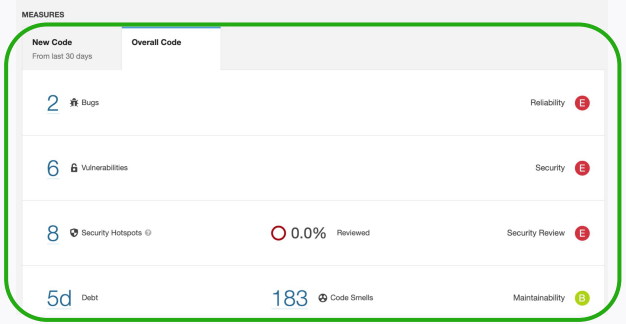




Significant increase in old code issues



My project with old quality profile



Same project with upgraded profile

Low or no increase in new code issues

Cleaning up the legacy code

Development lifecycle will renew old code into cleaner new code

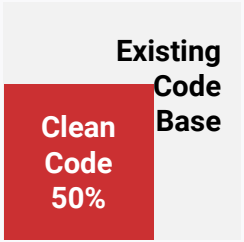
TODAY

Before SonarSource, the existing code base will contain an unknown percentage of bad code

1 YEAR WITH sonarsource

Every year, ~20% of the code base is changed; SonarSource is the Quality Gate for all new code

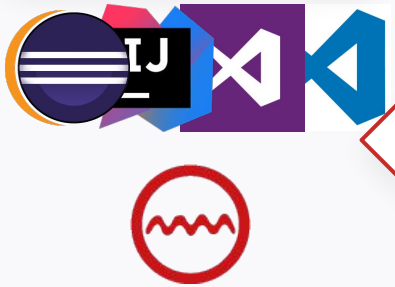
5 YEARS WITH sonarsource



In five years, SonarSource usage will systematically clean at least 50% of the code base

Going further

Shift left and shorten the feedback loop

On-the-fly analysis	Feature branch/ PR analysis	Main/Release branch analysis
		
2 sec	15 min	24h

Security overview

Security

What we do and don't do

SAST

Primarily for
Developers

Auditors

In the
DevSecOps
pipeline

DAST

SCA

Specific
process

Fast !

Security: What we detect

OWASP Top 10 Security Risk Categories

A1 **Injection**	A2 **Broken Auth**	A3 **Data Exposure**	A4 **XXE**
A5 **Broken Access Control**	A6 **Security Misconfiguration**	A7 **XSS**	A8 **Insecure Deserialization**
	A9 **Components with Vulnerabilities**	A10 **Insufficient Logging & Monitoring**	

Security in 8.9 LTS

Languages coverage and evolution

Web and Common Apps

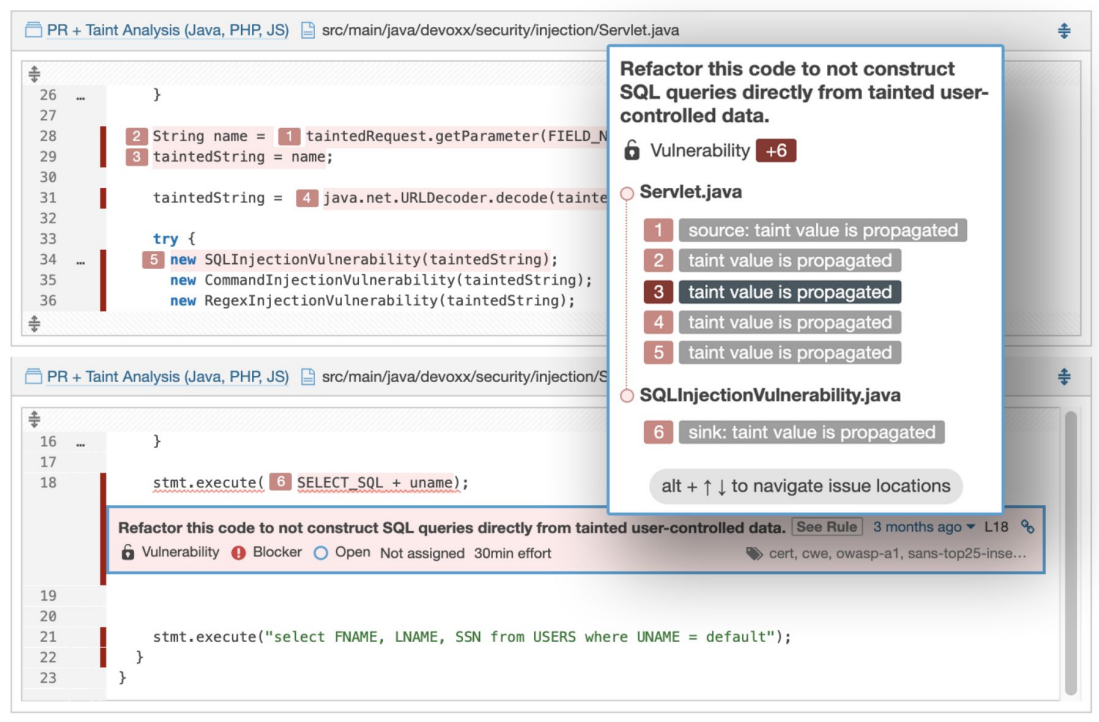
- Injections (taint) vulnerabilities
- Non injection vulnerabilities
- Security Hotspots


System & Embedded

- Buffer overflow Vulnerabilities
- Other non-injection Vulnerabilities
- Security Hotspots

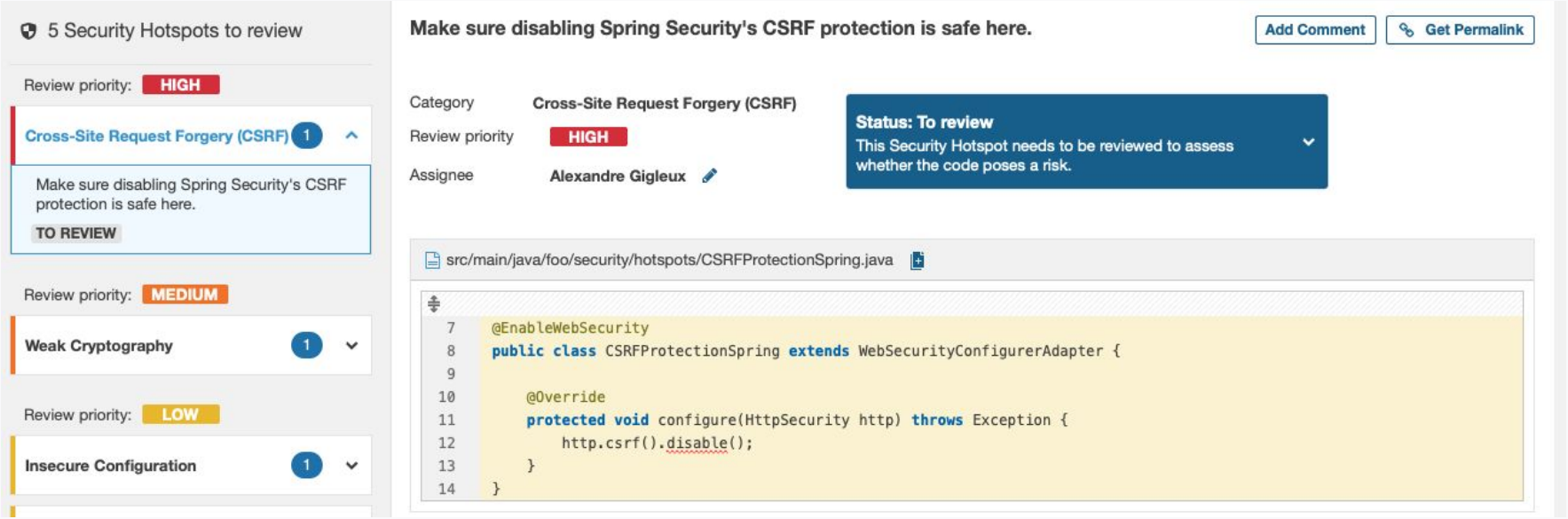

Vulnerabilities

Fix Security Risks



Security Hotspots

Review Security-sensitive code



sonarqube

sonarlint

Demo

Platform walkthrough