SonarQube Training

Developer Session

A journey in the land of code quality and security

Agenda

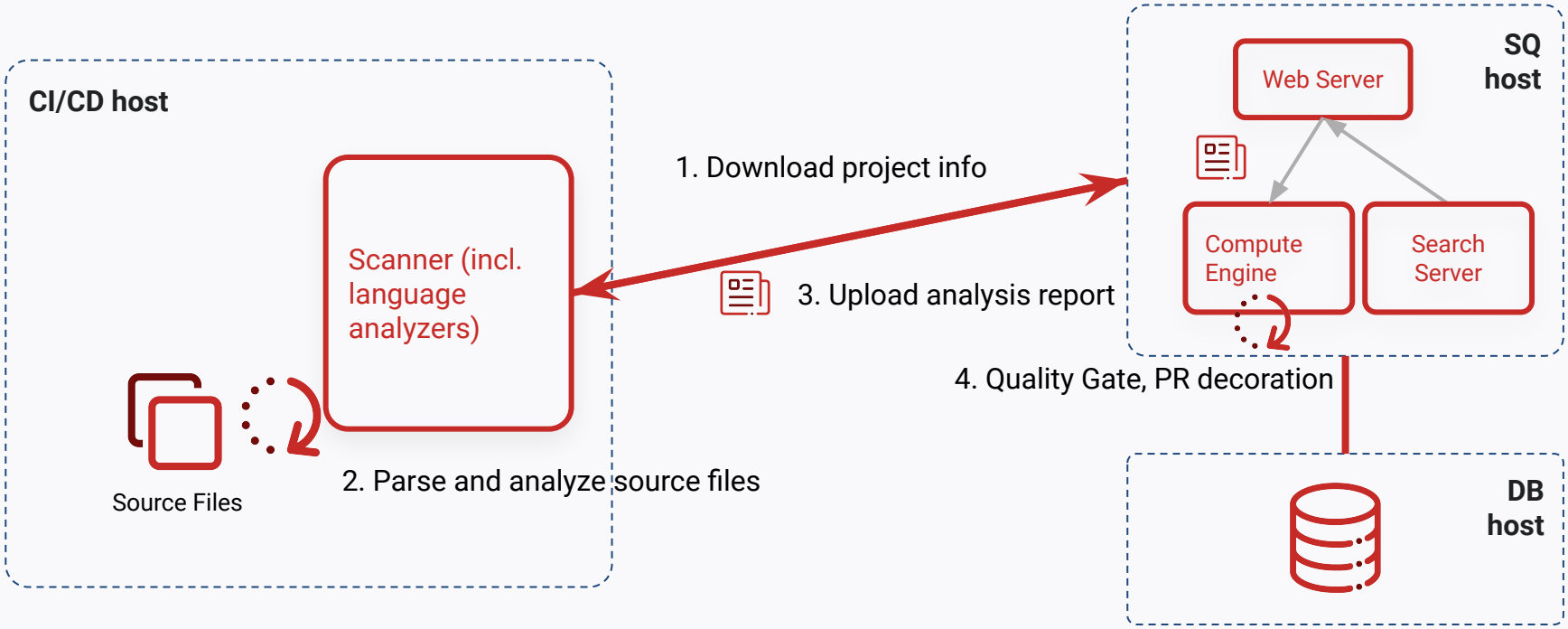
Developer Session

- Scanner integration
- Key metrics
- Rules and Quality Profiles
- Quality Gates
- Configuring Clean as you Code and the New Code Period
- Issue Management
- Working practically with SonarQube SAST results
- Branch Analysis / Pull Request decoration (Merge Request)
- SonarLint
- Q&A

Scanners

Scanning is simple and complex at the same time

Scanner workflow



Java-ish build systems

- Making it easy to scan using the same tool you use to build
- Each plugin provides a SonarQube scanning target
- Java analysis requires compile-time artefacts (retrieved from your config)
- Command line options added using `-D<option>=<value>`

Maven

Maven™

```
mvn sonar:sonar [options]
```

Gradle


```
gradlew sonarqube [options]
```

.NET scanning

- Special scanner must be used for .NET solutions
- Integrates with MSBuild and the Roslyn compiler
- Requires a full, clean build of your solution
- Command line options added using `/d:”<option>=<value>”`

.NET scanning

.NET Framework 4.6+ (Windows)

```
SonarQube.Scanner.MSBuild.exe begin /k:"projectKey" /d:"sonar.login=<AuthToken>" [optional parameters]

MSBuild.exe <path to solution.sln> /t:rebuild

SonarQube.Scanner.MSBuild.exe end /d:"sonar.login=<AuthToken>"
```

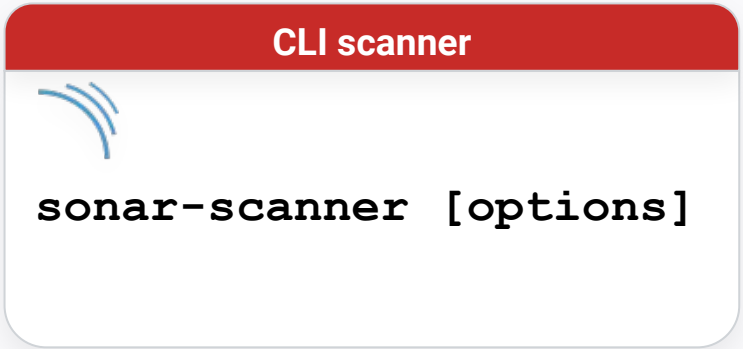
.NET Core Global Tool (Linux)

```
dotnet sonarscanner begin /k:"project-key" /d:"sonar.login=<AuthToken>" [optional parameters]

dotnet build <path to solution.sln>

dotnet sonarscanner end /d:"sonar.login=<AuthToken>"
```

Full parameter list: https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-msbuild/

Other types of projects: CLI scanner

- Invoked as a command line utility by your DevOps platform (or standalone)
- Used for languages where SonarQube doesn't need compile-time artefacts
  - Everything except Java and .NET
- Also used for C, C++, Objective-C (see later)
- Used in conjunction with **sonar-project.properties** config file or/and…
- …Command line options added using `-D<option>=<value>`



C, C++, Objective-C

Build Wrapper

```
build-wrapper --out-dir <some_dir> <build command>

sonar-scanner -Dsonar.cfamily.build-wrapper-output=<some_dir>
```

Compilation database

```
<build command that generates compile_commands.json>

sonar-scanner -Dsonar.cfamily.compile-commands=compile_commands.json
```

Compilation DB: Compilation database | SonarSource blog
Parameters: C/C++/Objective-C | SonarQube Docs
Example repos: https://github.com/sonarsource-cfamily-examples
Note: multi-threading and cached analysis options available

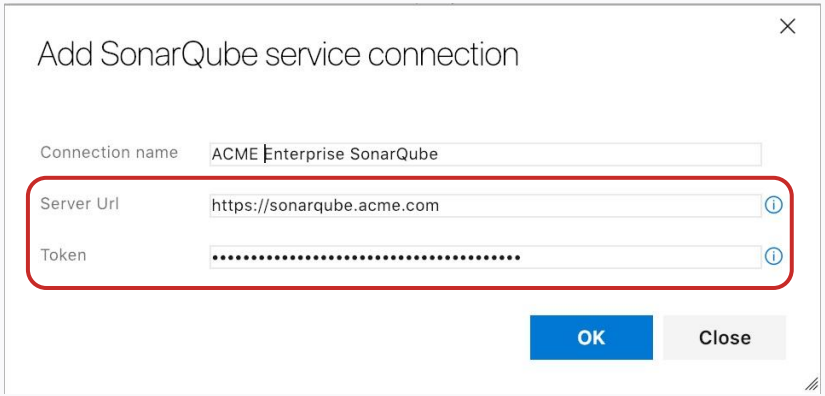
DevOps pipeline integrations

Dedicated tasks and pipeline support:

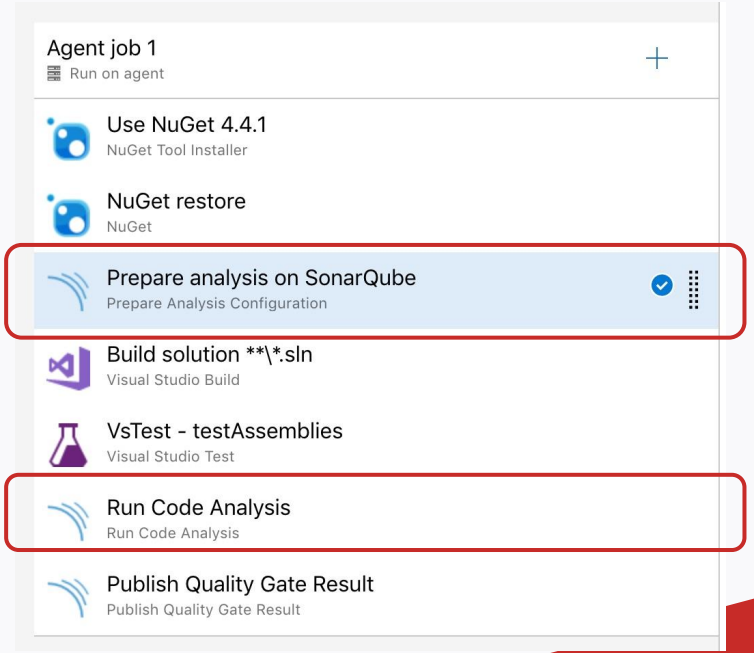
- GitHub Actions
- BitBucket Cloud Pipelines
- GitLab CI/CD Pipelines
- Azure DevOps extension
- Jenkins plugin

Azure DevOps Extension

1. **Define your service endpoints and credentials centrally**



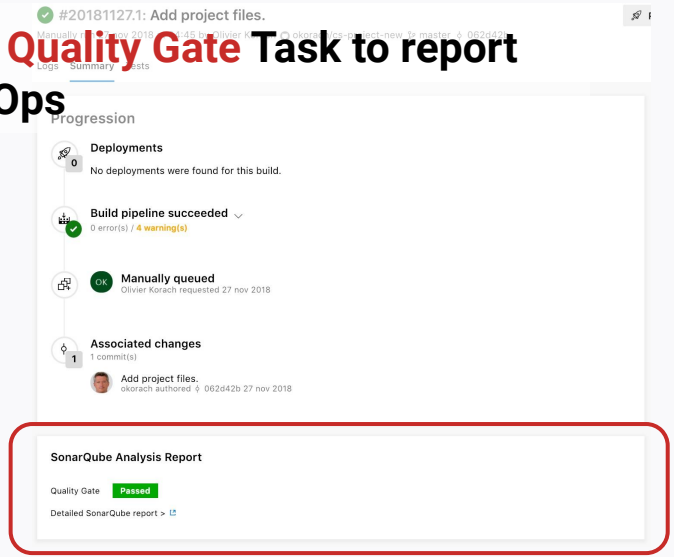
2. **Easy tasks to create your build jobs**

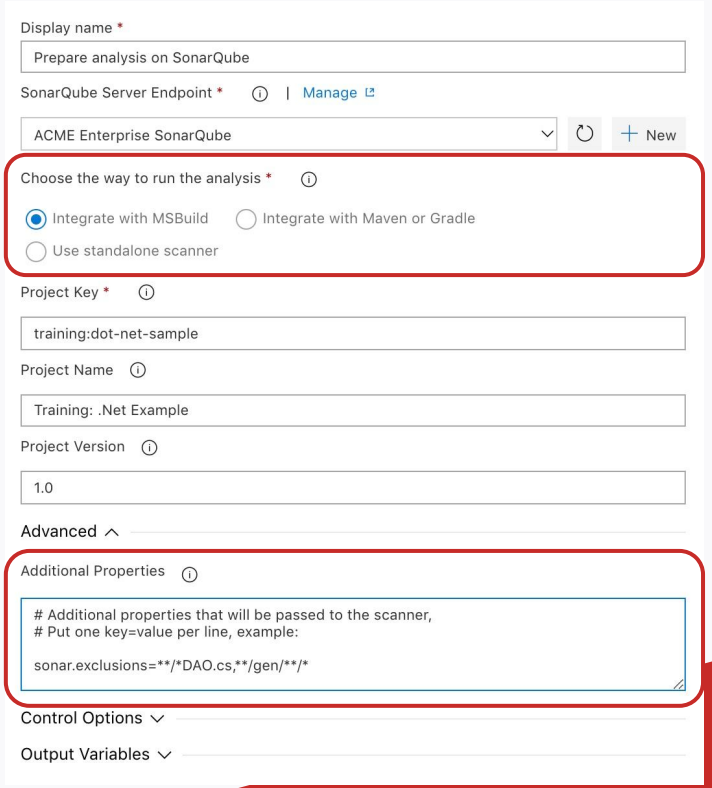


Azure DevOps Extension

Prepare Analysis Task to select MSBuild, Maven, Gradle or Standalone scanner and specify analysis properties

Optional Publish Quality Gate Task to report QG in Azure DevOps

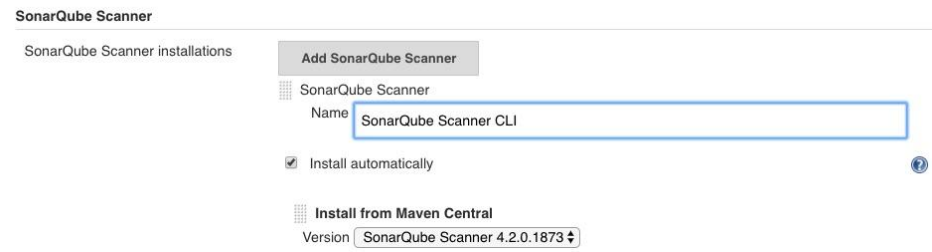




Jenkins plugin

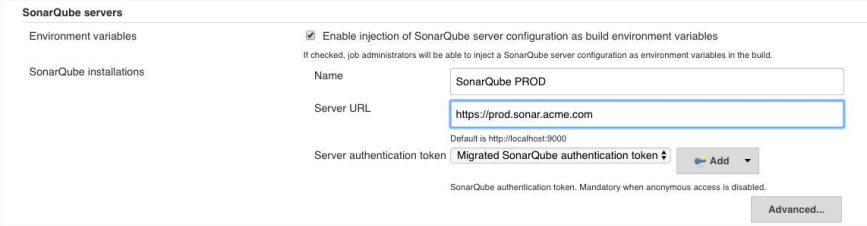
1. Install Scanners

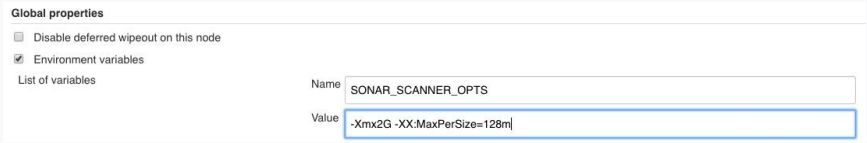
Global Tool Configuration



2. Configure Server

Configure System





Jenkins plugin

3. Define Step or Pipeline
Declarative pipeline example:

```
pipeline {
    agent any
    stages {
        stage(' SCM') {
            steps {
                git url: ' https://github.com/SonarSource/sonar-java ', branch: ' master'
            }
        }
        stage(' Build') {
            steps {
                withMaven() {
                    sh ' mvn clean org.jacoco:jacoco-maven-plugin:prepare-agent verify  '
                }
            }
        }
        stage('SonarQube analysis') {
            steps {
                withMaven() {
                    withSonarQubeEnv(‘SonarQube PROD’) {
                        sh 'mvn sonar:sonar'
                    }
                }
            }
        }
        stage('Quality Gate') {
            steps {
                timeout(time: 1, unit: 'HOURS') {
                    waitForQualityGate false
                }
            }
        }
    }
}
```

Scanning w/ GitLab - Scanner CLI

```
sonarqube-check:
  image:
    name: sonarsource/sonar-scanner-cli:latest
  variables:
    GIT_DEPTH: "0"
  cache:
    key: "${CI_JOB_NAME}"
    paths:
      - .sonar/cache
  script:
    - sonar-scanner -Dsonar.qualitygate.wait=true
  allow_failure: false
  only:
    - merge_requests
    - master
    - develop
```

SonarSource official scanner docker image embedding all latest scanner requirements (OS/JVM/Node.JS)

Force full depth clone
Shallow clone can have side effects

Define scanner plugin cache to not re-download for each pipeline execution

Fail pipeline when Quality Gate fails

Trigger on Merge Requests and key branches

GitHub Actions & docker scanner alternative

```
scan-cli:
   steps:
     - name: Checkout code
       uses: actions/checkout@v2
       with:
         fetch-depth: 0
     - name: Cache SonarQube plugins
       uses: actions/cache@v1
       with:
         path: ~/.sonar/cache
         key: ${{ runner.os }}-sonar
         restore-keys: ${{ runner.os }}-sonar
     - name: Analyze CLI comp
       uses: docker://sonarsource/sonar-scanner-cli:latest
       env:
         GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
         SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
         SONAR_HOST_URL: ${{ secrets.SONAR_HOST_URL }}
       with:
         sonar.qualitygate.wait: false
```

Force full depth clone
Shallow clone can have side effects

Define scanner plugin cache to not re-download for each pipeline execution

Run official scanner docker image embedding all latest scanner requirements (OS/JVM/Node.JS)

Secrets passed to scanner for analysis submission

No need to wait for QG
Decoration is asynchrononous

Scanning w/ GitHub Actions

```
name: SonarQube scan
jobs:
  scan-cli:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
      with:
        fetch-depth: 0
    - name: SonarQube Scan
      uses: sonarsource/sonarqube-scan-action@master
      env:
        SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
        SONAR_HOST_URL: ${{ secrets.SONAR_HOST_URL }}
```

Force full depth clone
Shallow clone can have side effects

Run scanner CLI GitHub Action

Pass GitHub secrets as env vars for scanner analysis submission

See: https://github.com/marketplace/actions/official-sonarqube-scan

Scanning w/ AWS CodeBuild

```
version: 0.2

phases:
  build:
    commands:
      - mvn sonar:sonar -Dsonar.login=$SONAR_LOGIN -Dsonar.host.url=$SONAR_HOST
```

Branch and PR/MR auto-detection

Simpler pipelines without need for conditional logic:

- GitHub
- GitLab
- Azure DevOps
- BitBucket Pipes
- Jenkins (using multi-branch plugin)
- CodeMagic

Failing pipelines

Your decision whether to fail pipelines:

- Jenkins: webhooks (lightweight, no polling)
- GitHub Actions: SonarQube Quality Gate Check
- BitBucket Pipelines: SonarQube Quality Gate Check
- Others: `sonar.qualitygate.wait=true`

Scanning Example

Narrowing the Focus

Generated and third-party source code

- Usually not under your control
- Often has many issues that ‘pollute’ your SonarQube projects
- General rule: don’t scan this code
- Or scan once in a separate project

Excluding code - *sonar.exclusions, sonar.test.exclusions*

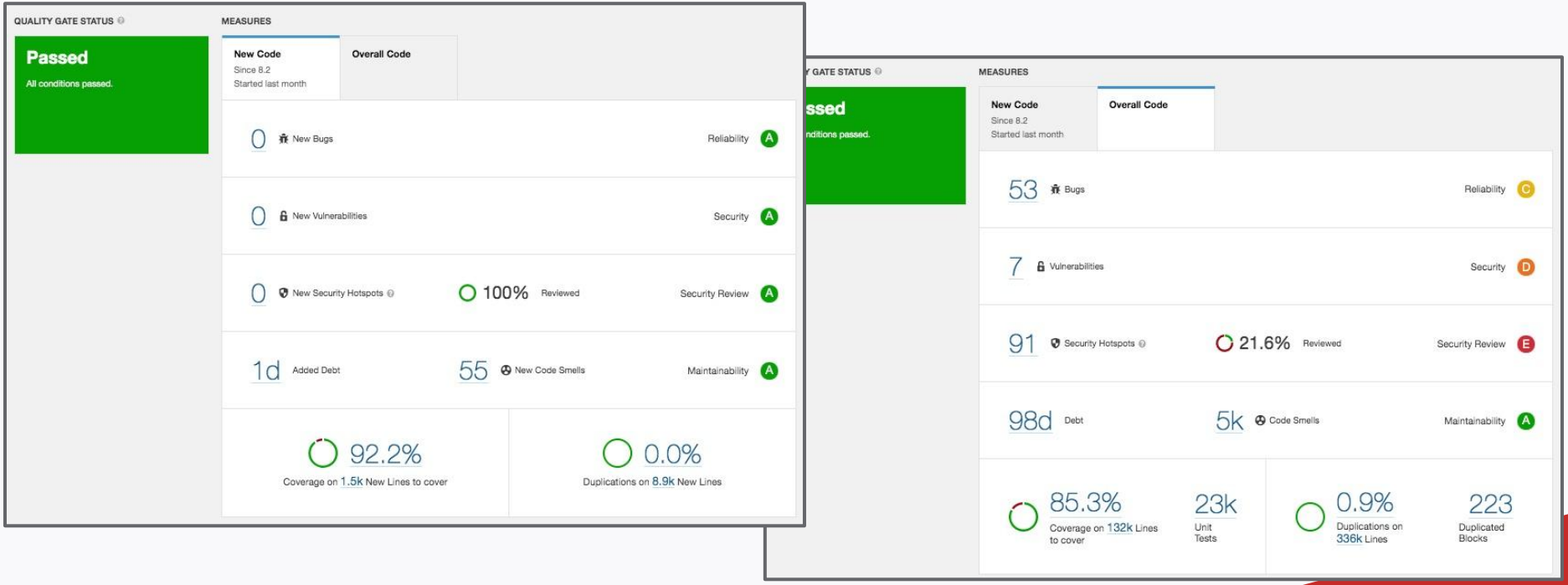
Including only some code - *sonar.inclusions, sonar.test.inclusions*

Key metrics

Focus on what matters first

Project key metrics

Focus on what matters first



Security and Reliability Ratings

A matter of severity

Rating represents the single highest Severity Vulnerability or Bug in the code period

Severity	Rating
Blocker	E
Critical	D
Major	C
Minor	B
Info (or No issues)	A

Tech Debt ratio and Maintainability Rating

A matter of code smell density

- Tech Debt (TD): Total effort to fix all CODE SMELL issues
- TD Ratio: Tech Debt / Effort to entirely rewrite the code
- Example: project with
  - **118** code smells → **1,120** minutes of Technical Debt,
  - **1,500** LoC → Total rewrite effort = 1500 x 30 min = **45,000** min
- **Tech Debt Ratio** = 1120 / 45000 = **2.5%**
- **Maintainability Rating**: A

Tech Debt Ratio	Maintainability Rating
0 - 5%	A
5 - 10%	B
10 - 20%	C
20 - 50%	D
> 50%	E

Security Review Rating

How are you doing on review?

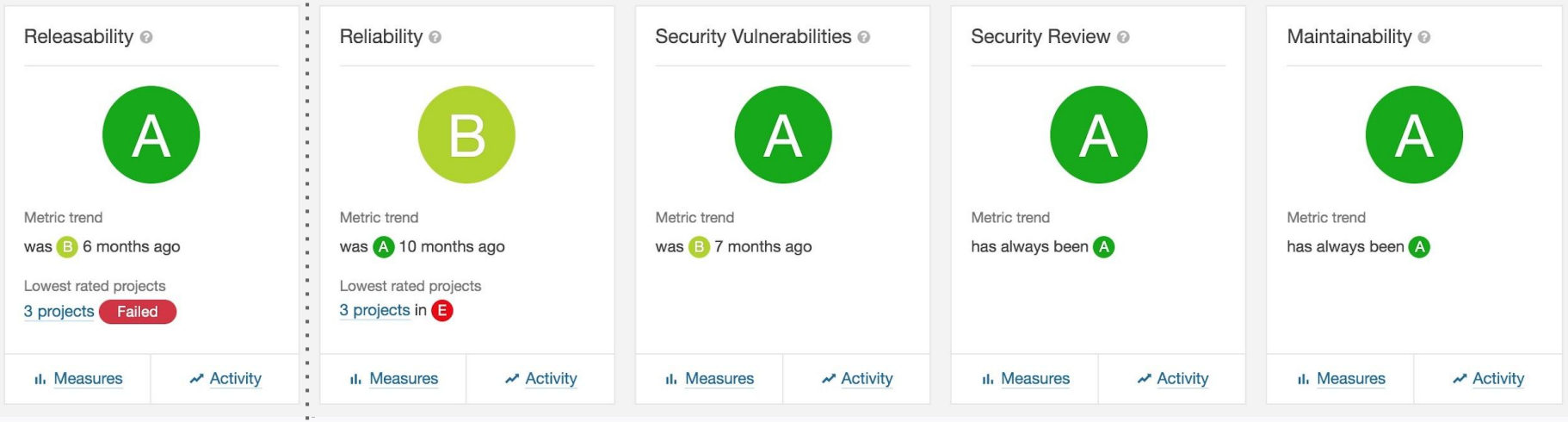
SonarQube 8.1+

Rating represents the percentage of hotspots reviewed

Separate ratings for overall code and New Code

Reviewed	Rating
< 30%	E
30 - 50%	D
50 - 70%	C
70 - 80%	B
>= 80%	A

Portfolio key metrics



% of projects Passed

- > 80% => A
- > 60% => B
- > 40% => C
- > 20% => D
- <= 20% => E

Average of underlying projects ratings

- Unweighted
- All projects under the portfolio at any depth (not portfolios and projects only 1 level below)

Complexity

- **Metric** exists on projects but barely useful
- What matters is to spot methods with excessive complexity
- Better achieved through complexity **rules**: An issue is raised when complexity is above threshold, pointing at the problematic method

Complexity

Cyclomatic = **Testability**

Cognitive = **Understandability**
(ie Maintainability too)

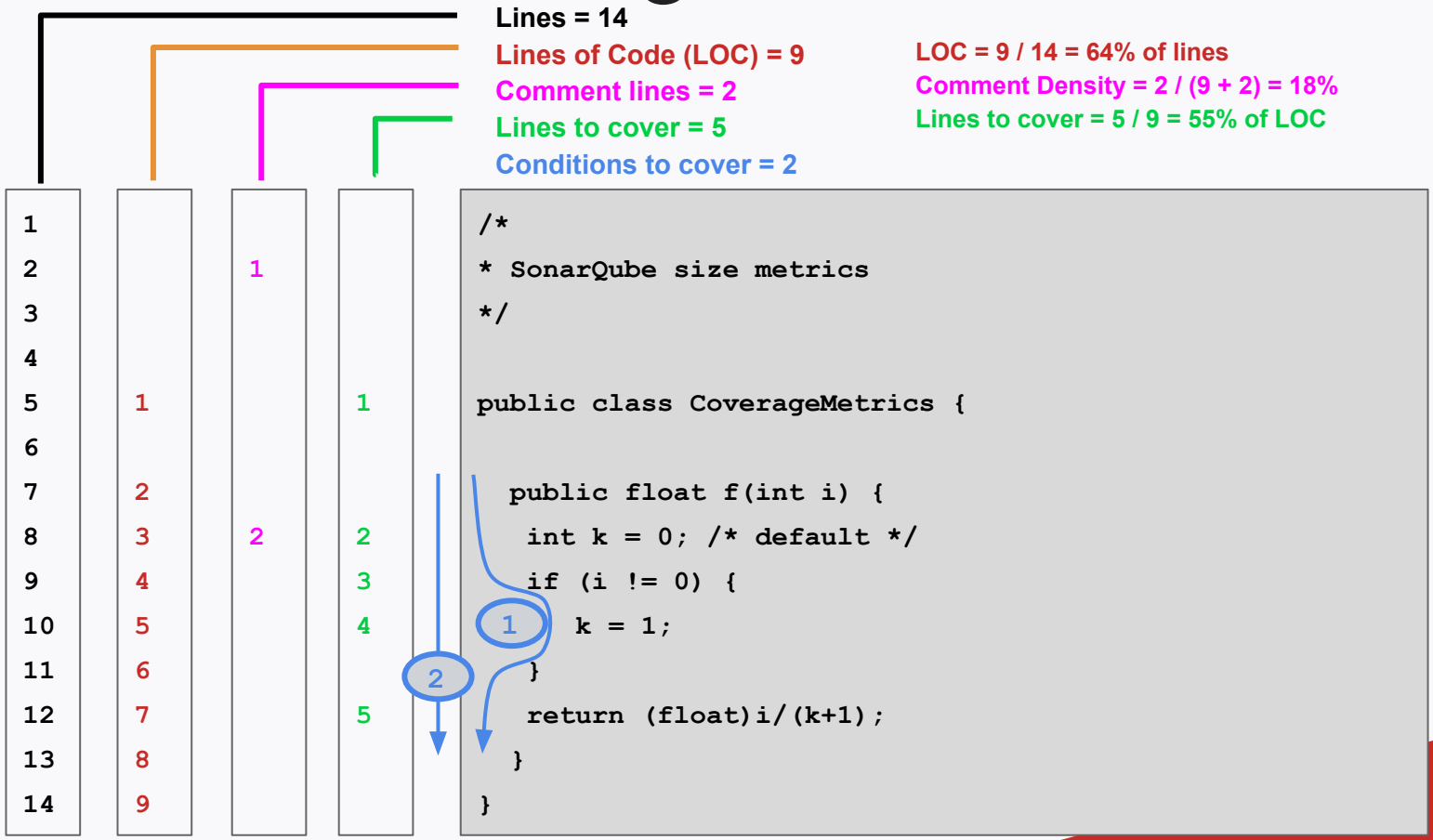
```
public static void highCycloLowCognitive(int i)
{
   switch (i) {
      case 1:
         doSomething();
         break;
      case 2:
         doSomethingElse();
         break;
      case 3:
         forgetIt();
         break;
      case 4:
         forgetItAgain();
         break;
      case 5:
         yetAnotherCase();
         break;
      case 6:
         soBoring();
         break;
      case 7:
         timeForVacations();
         break;
      case 8:
         eightCase();
         break;
      case 9:
         cantSeeTheEndOfIt();
         break;
      case 10:
         tenThisIsIt();
         break;
      default:
         doNothing();
         break;
   }
```

Cyclomatic Complexity = 11
Cognitive Complexity = 2

```
public static void highCycloVeryHighCognitive(int i)
{
   if ( i <= 5 ) {
      if ( i <= 2 ) {
         if ( i == 1 ) {
            doSomething();
         } else {
            doSomethingElse();
         }
      } else {
         if ( i > 3 ) {
            forgetItAgain();
         } else {
            forgetIt();
         }
      }
   } else {
      if ( i <= 7 ) {
         if ( i == 6 ) {
            soBoring();
         } else if ( i == 7 ) {
            timeForVacations();
         } else {
            // Never reached
            doNothing();
         }
      } else {
         if ( i == 8 ) {
            eightCase();
         } else if ( i == 9 ) {
            cantSeeTheEndOfIt();
         } else if ( i == 10 ) {
            tenThisIsIt();
         } else {
            doNothing();
         }
      }
   }
}
```

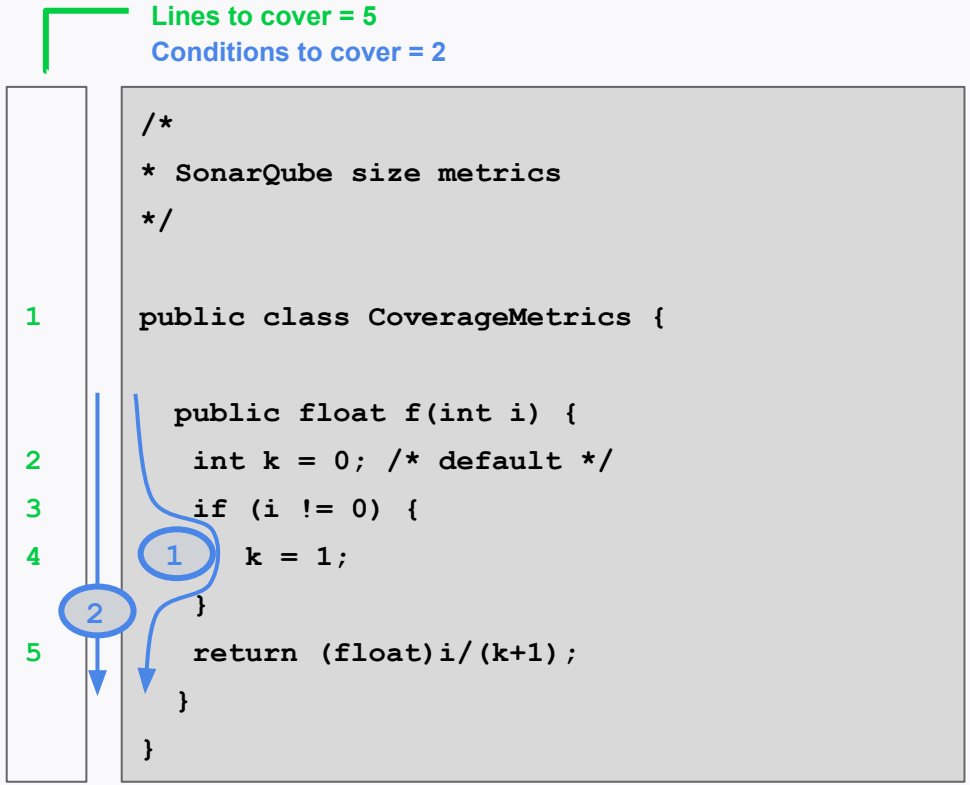
Cyclomatic Complexity = 11
Cognitive Complexity = 27

Line and coverage metrics



Coverage

Line coverage = Lines covered / Lines to cover
Condition coverage = (Cond true + cond false) / 2 * branches
(Overall) coverage = (Cond true + cond false + covered lines) / (2 * branches + exec lines)



Unit Test 1: `assertEquals(0.5, c.f(1), 0.0);`

Line coverage	= 5 / 5 = 100%
Cond coverage	= (1 + 0) / 2 = 50%
Overall coverage	= (1+0+5) / (2+5) = **85.7%**

Unit Test 2: `assertEquals(0.0, c.f(0), 0.0);`

Line coverage	= 4 / 5 = 80%
Cond coverage	= (0 + 1) / 2 = 50%
Overall coverage	= (0+1+4) / (2+5) = **71.4%**

UT1 + UT2

Line coverage	= 5 / 5 = 100%
Cond coverage	= (1 + 1) / 2 = 100%
Overall coverage	= (1+1+5) / (2+5) = **100%**

Rules and quality profiles

Defining your quality governance

Quality Profile best practices

- **One or a few QP** per language only
- Set an organizational baseline and use QP inheritance
- If using LATEST, revisit QP on a periodic basis

Quality Profile management

- Two key options
  - **Inherit** (from Sonar Way or your own baseline)
  - **Copy** (removes automatic inheritance)
- *Compare* function available to measure drift from baseline

Quality Gates

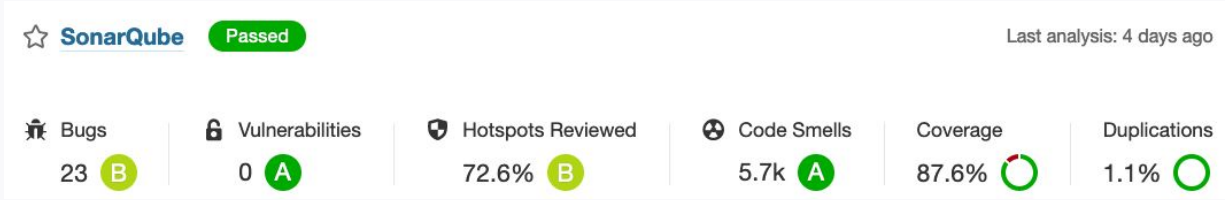
Defining your quality governance

Quality Gate best practices

- **One or a few Quality Gates** only
- Use metrics **on new code**
- **A** reliability rating (0 bugs)
- **A** security rating (0 vulnerabilities)
- **100%** hotspots reviewed
- Set thresholds that are challenging but reachable, then tighten criteria progressively

Quality Gates best practices

6 Key project metrics:



6 Key Quality Gate criteria…

(at least)

Conditions on New Code

Conditions on New Code apply to all branches and to Pull Requests.

Metric	Operator	Value
Coverage	is less than	80.0%
Duplicated Lines (%)	is greater than	3.0%
Maintainability Rating	is worse than	A
Reliability Rating	is worse than	A
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A

… On new code, to adhere to CAYC paradigm

What ? No criteria on legacy ?

Yes you can, but only:

- When your *new code* is well under control
- On very specific metrics

Conditions on New Code

Conditions on New Code apply to all branches and to Pull Requests.

Metric	Operator	Value	Edit	Delete
Coverage	is less than	80.0%		
Duplicated Lines (%)	is greater than	3.0%		
Maintainability Rating	is worse than	A		
Reliability Rating	is worse than	A		
Security Hotspots Reviewed	is less than	100%		
Security Rating	is worse than	A		

Conditions on Overall Code

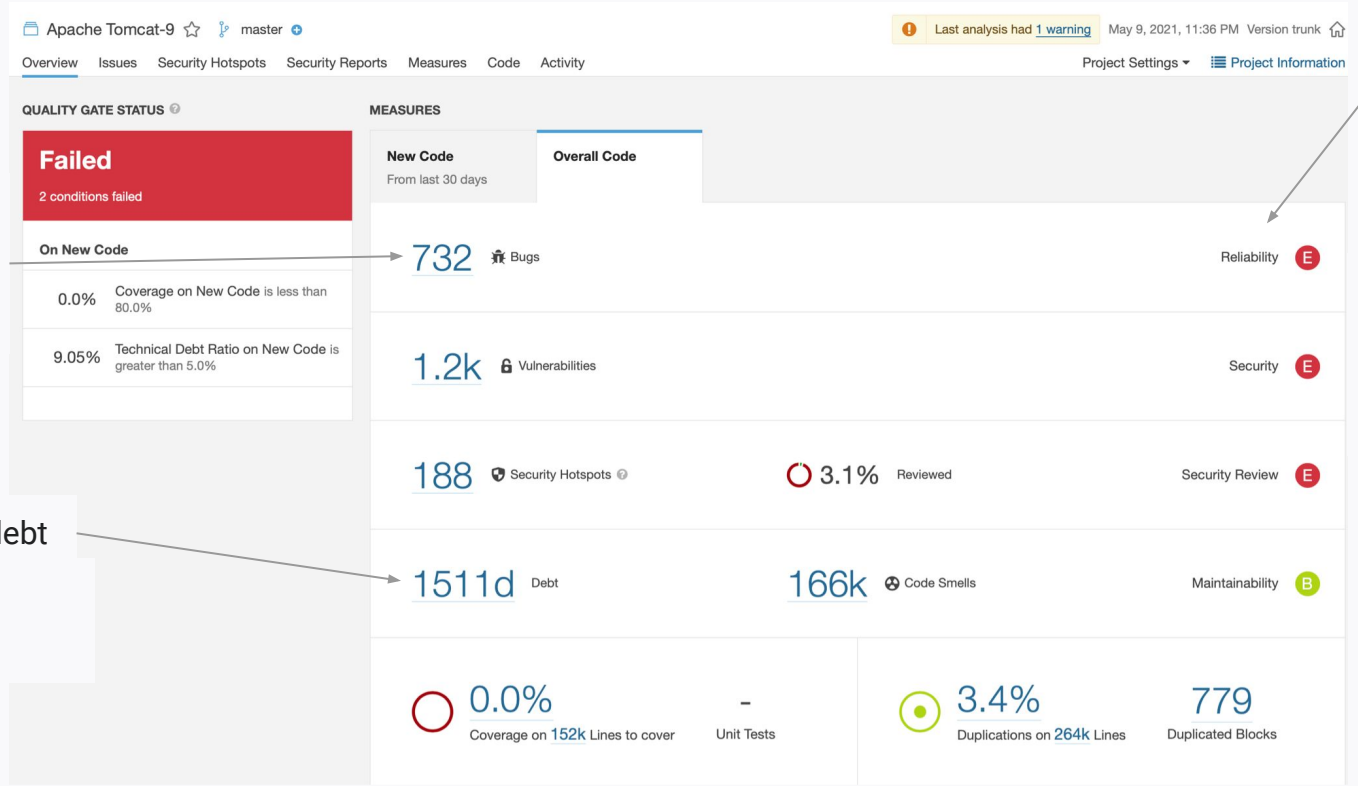
Conditions on Overall Code apply to branches only.

Metric	Operator	Value	Edit	Delete
Reliability Rating	is worse than	D		
Security Rating	is worse than	D		

Clean As You Code

Configuring the new code period

Caveats of traditional approaches



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

Risk of functional regression

Developer pushback

- Not my code
- Too many issues
- Boring!

Huge legacy debt

Where to find budget?

Resistance to evolution of Quality Profiles

What is best practice?

You take personal responsibility for *your* code

Setting the right new code period

Well tuned new code period leads to better code quality

New code period should be

- Long enough to have time to fix issues before they fall into the “legacy” period
- Short enough to enforce fixing issues before they accumulate too much
- Typically between 2 weeks and 3 months

New Code Example

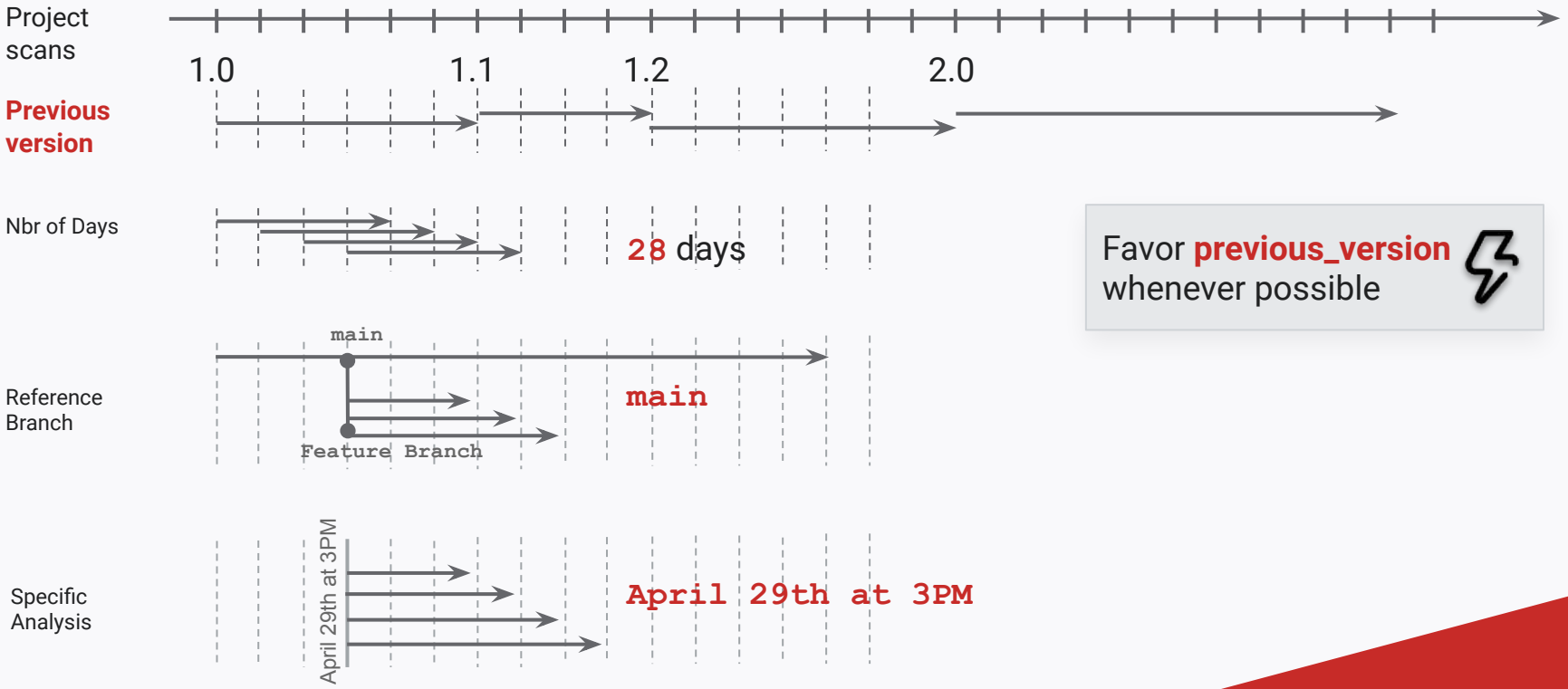
But… what's the new code ?

4 different configurations possible

- **Previous Version**
  - **Best option** if compatible with your versioning scheme
  - Align New Code period with releases/sprints
  - Fully automatic
- Reference Branch
  - Specific branch chosen as baseline
  - Changes considered as New Code
- Number of days
  - Sliding window for new code
- Specific Analysis
  - Specific Analysis in a given branch chosen as baseline
  - Changes considered as New Code
  - Can be set up at branch level only

But… what's the new code ?

4 different configurations possible



Favor **previous_version** whenever possible

CAYC and project versioning

- For smooth CAYC application, project versioning should be properly defined (`sonar.projectVersion`)
- Bad practices
  - Always provide same version
  - Change version for every build
  - Randomly change the version
  - Change version although the code fails the Quality Gate

CAYC corollary: Respect Quality Gates

- CAYC approach depends on strict enforcement of Quality Gate
- Don’t reset New Code period on a failed quality gate

Issue Management

Because false positives happen

Issue management

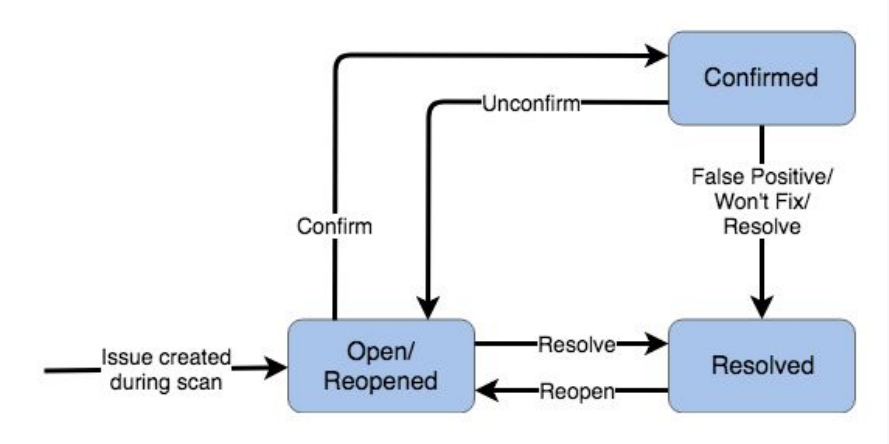
Resolving issues the right way

Automatic resolution

- Just fix the code 😀

Manual resolution

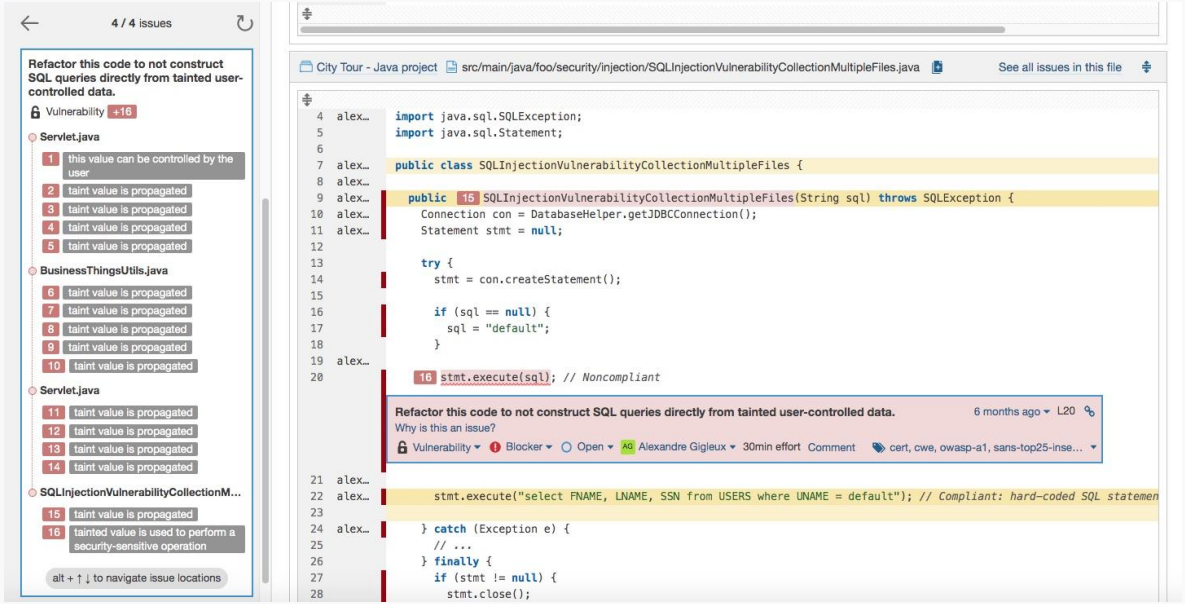
- *Administer issues* permission required
- False positive
- Won’t fix
- Never reopened (in that location)



Multiple issue locations

Managing complex data flow issues

- SonarQube highlights data flow
- Cross-file analysis
- Helps you decide where and how to fix



Working with Security

Injections reporting UI and Security hotspots workflow

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**	

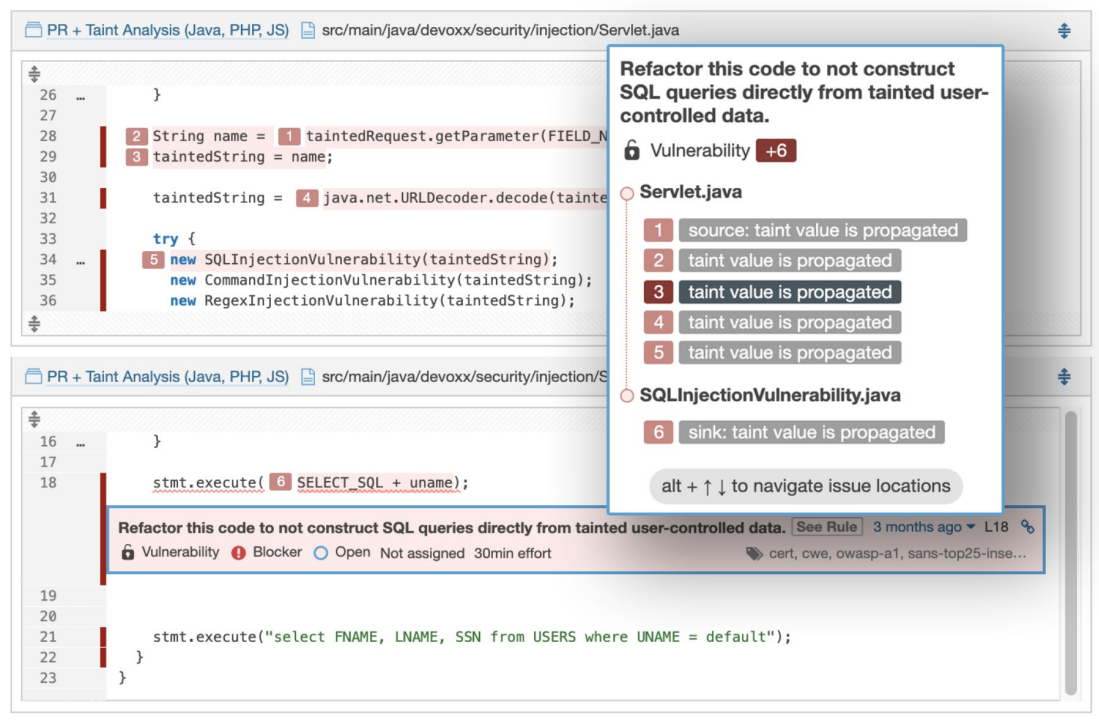
Security: What we detect

OWASP Top 10 Security Risk Categories

	Java	C#	PHP	Python	JS	TS	C/C++
A1 Injection (SQL, OS, LDAP...)	■	■	■	■	■	■	
A2 Broken Authentication	■	■	■	■	■	■	■
A3 Sensitive Data Exposure	■	■	■	■	■	■	■
A4 XML External Entities (XXE)	■	■	■	■	■	■	■
A5 Broken Access Control	■	■	■	■	■	■	■
A6 Security Misconfiguration	■	■	■	■	■	■	■
A7 Cross-Site Scripting (XSS)	■	■	■	■	■	■	
A8 Insecure Deserialization	■	■	■	■			
A9 Using Components with Known Vulnerabilities	■	■		■			■
A10 Insufficient Logging & Monitoring	■	■	■	■			

Vulnerabilities

Fix Security Risks



Security Hotspots

What are they ?

Analyzer has identified a security risk...

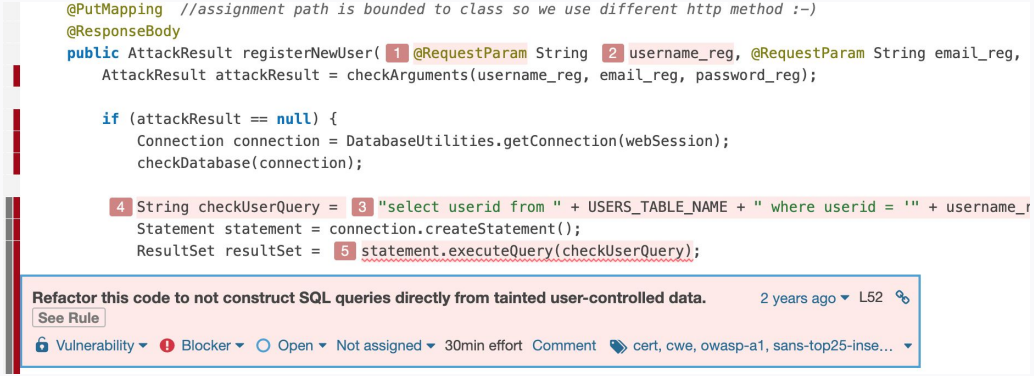
→ **VULNERABILITY**

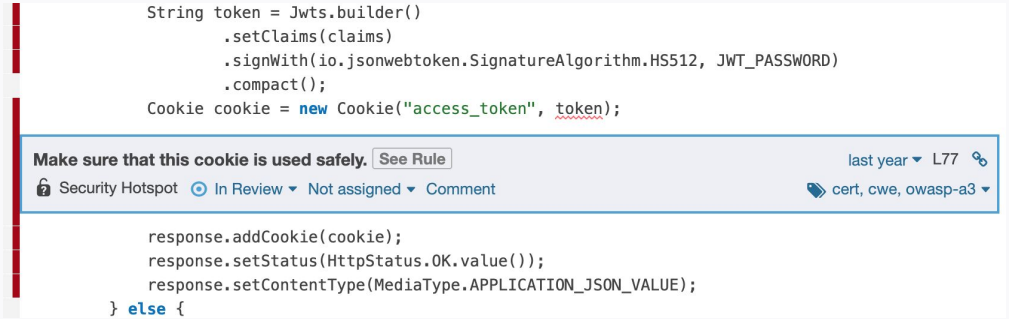
→ **FIX**

Developers should be aware they are writing security-sensitive code...

→ **HOTSPOT**

→ **REVIEW**





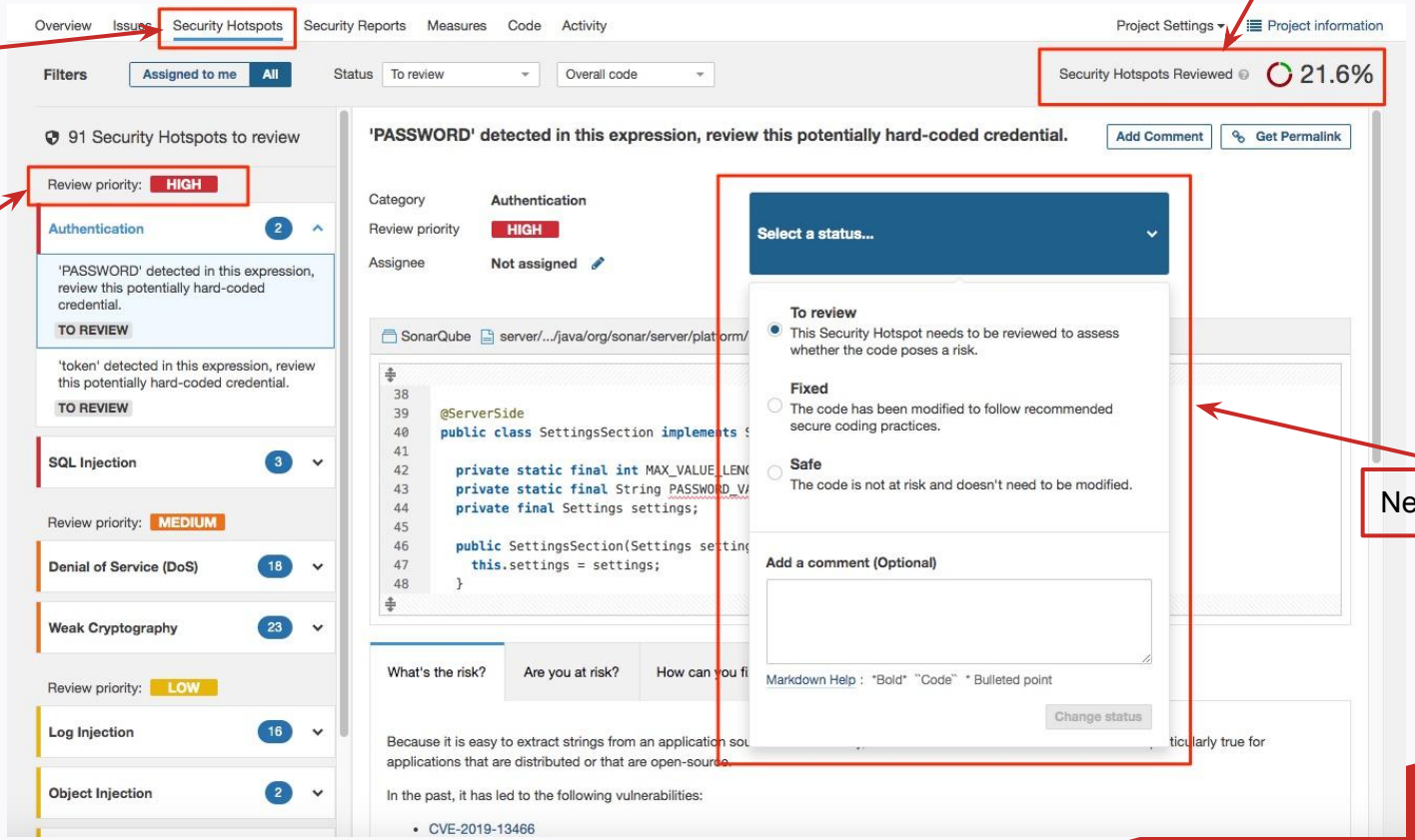
Hotspot lifecycle

Dedicated tab

Progress indicator

Sorted by priority

New workflow



Security Rating

Overview of code security

Projects

Highest severity Vulnerability in the code period

Portfolios

Average of the Security Ratings of included Projects

Severity	Rating
Blocker	E
Critical	D
Major	C
Minor	B
Info	A

Security Review Rating

How are you doing on Security Hotspot review?

Projects

Percentage of hotspots reviewed in the code period

Portfolios

Average of the Security Review Ratings of included Projects

Reviewed	Rating
< 30%	E
30 - 50%	D
50 - 70%	C
70 - 80%	B
>= 80%	A

Customization

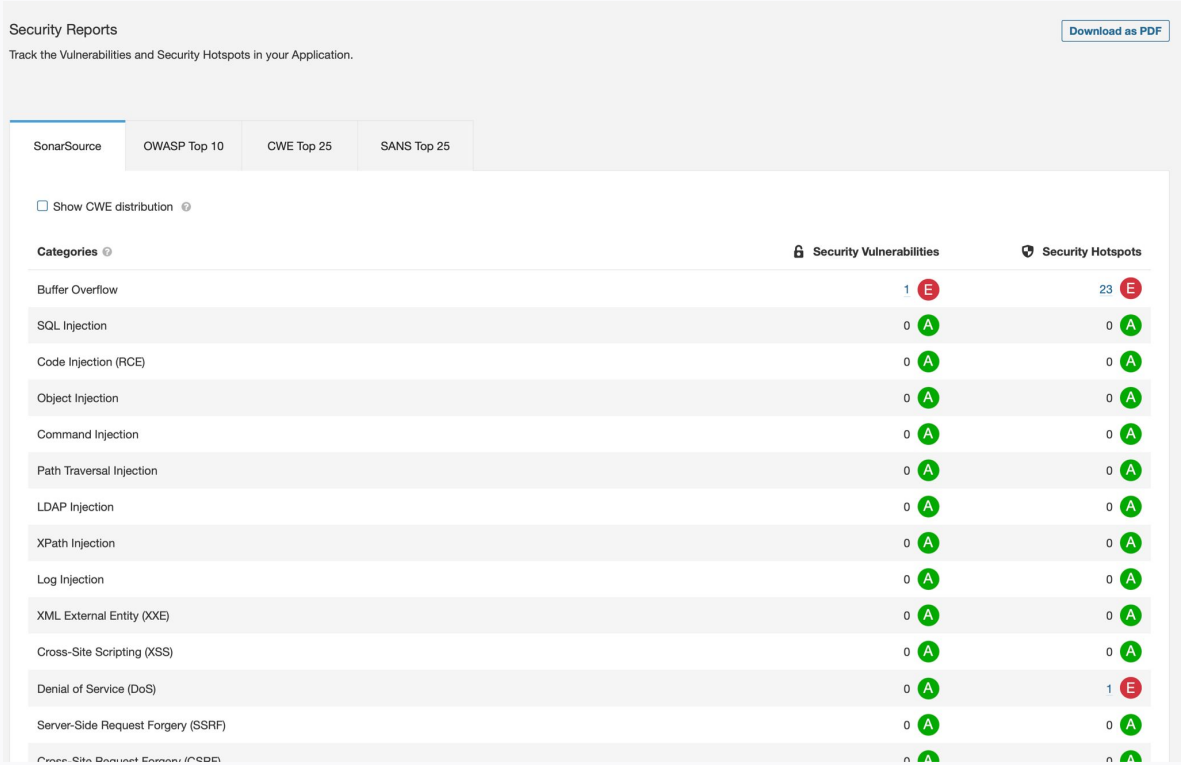
Support your Proprietary Frameworks

Make the Security Engine aware of your own frameworks by adding:

SOURCES	SANITIZERS	PASS-THROUGHS	SINKS
Input data, vulnerable to **Injection Attacks** (tainted data)	**Removes malicious content** from Input data	**Keep track of data** passing through libraries outside of your code	**Security-sensitive functions** that accept Input data

Compliance

OWASP/CWE Security Reports for Projects & Portfolios

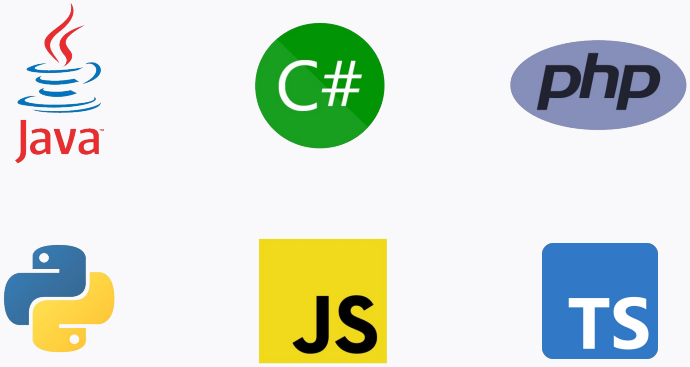


Security in 8.9 LTS

Languages coverage and evolution

Web Apps

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



System & Embedded

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

Security best practices

Elevate security governance with SonarQube

- Include Security Rating in Quality Gates
- Incorporate Security Hotspot reviews into your development cycle
- Enforce Security Hotspot reviews using the Security Review Rating Quality Gate criterion (8.3+)

Branch and PR analysis

Shift left in your feature branches

Why Use Branch Analysis in SonarQube?

- Issue metadata shared across branches
- Focus on issues created within each branch
- Branch-specific New Code Periods
- Licensing

Scanner Configuration

- `sonar.branch.name`
- `sonar.pullrequest.key`
- `sonar.pullrequest.branch`
- `sonar.pullrequest.base`

- Automated in Jenkins, Azure Devops, GitLab and GitHu

Issue Behavior

- Issues metadata is synched
  - From parent branch at creation
  - From PR to target Branch after merge
  - To reference branch after merge
- Comment added when issue is synced

“The issue has been copied from branch 'xxx' to branch ‘yyy’.”

Integrating Branch Analysis

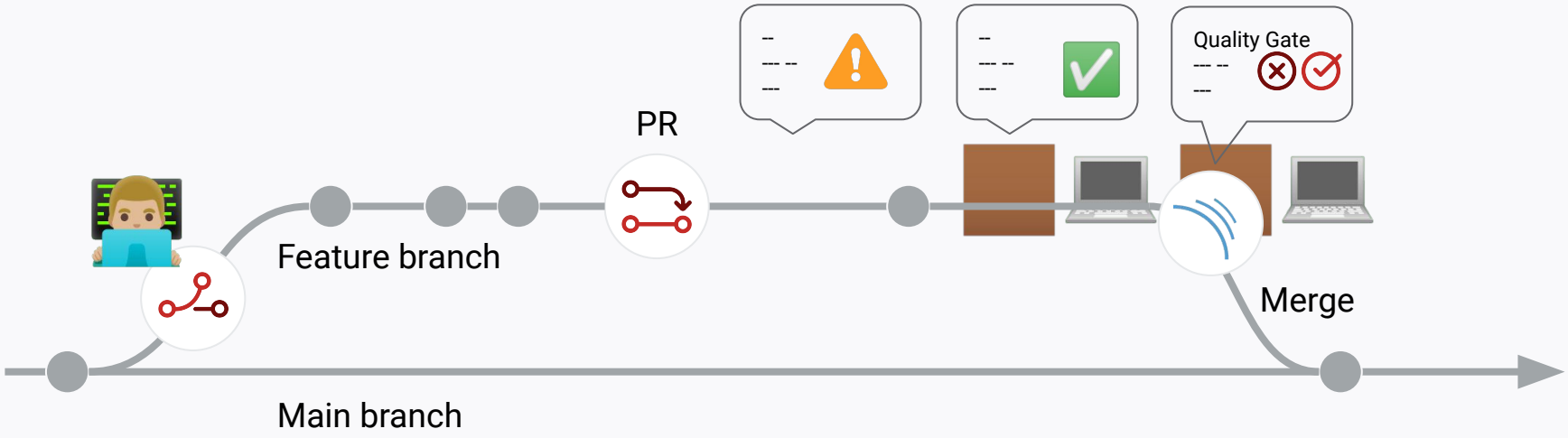
System setup

- Set global branch naming regex to organization standard
- Set lifespan of short-lived branches

Per-project setup

- Set up project scanning main (`'master'`) branch if not already existing
- Set project branch naming regex if varies from global setting
- Add branch builds to CI/CD
  - Retrieve `sonar.branch.name` and `sonar.branch.target` from build trigger
- Build out branch structure ensuring parent branches scanned before children
- Set leak period for long-lived branches is different from main.

Pull request workflow



Thanks to **SonarQube PR decoration**

- Developers know about issues in their code **before merging**
- Information about issues is **pushed in the SCM**
- It is possible to **enforce a policy** that PR quality gate must be passed before pull request can be merged

Example: GitHub PR Decoration

Integration steps

Global Settings

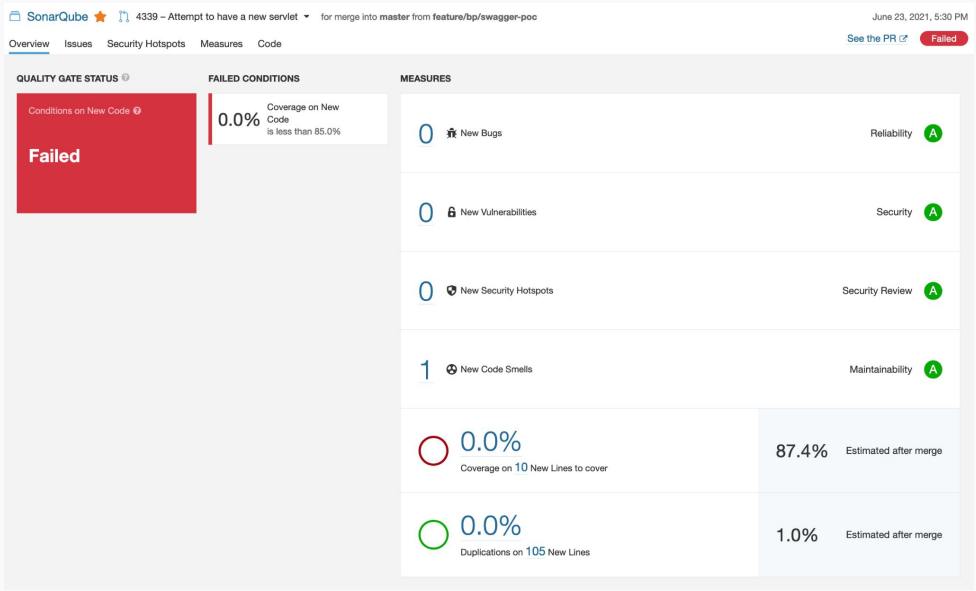
1. Register App on GitHub
2. Add information to SonarQube
   a. Private Key
   b. API URL
   c. App Name and ID
3. Install App on Organization

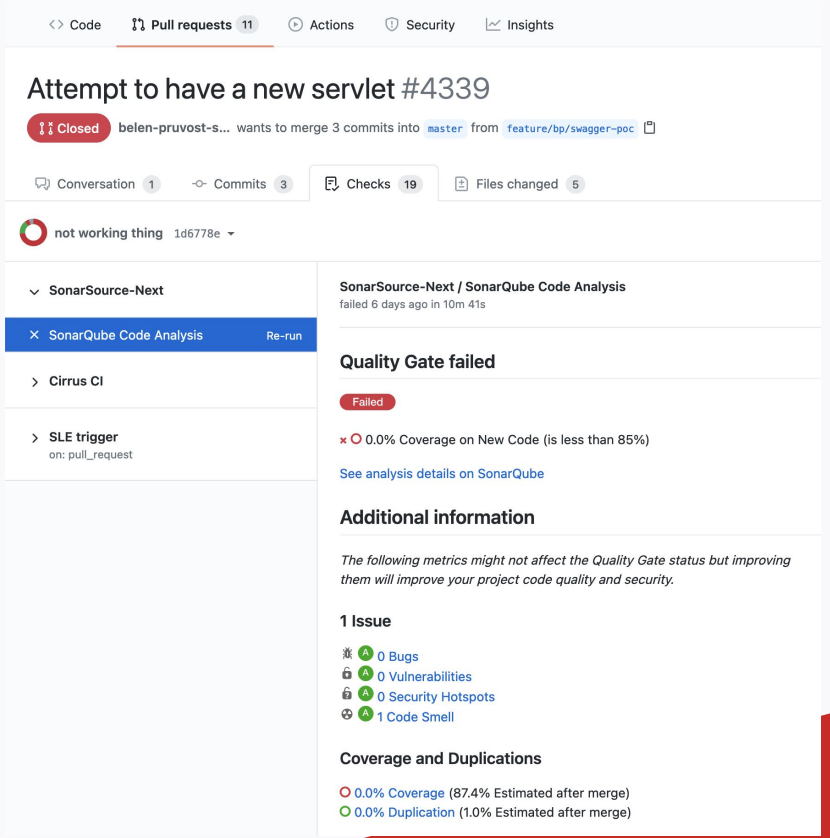
Project Settings

1. Add Repository identifier
2. Add pull request parameters to scan

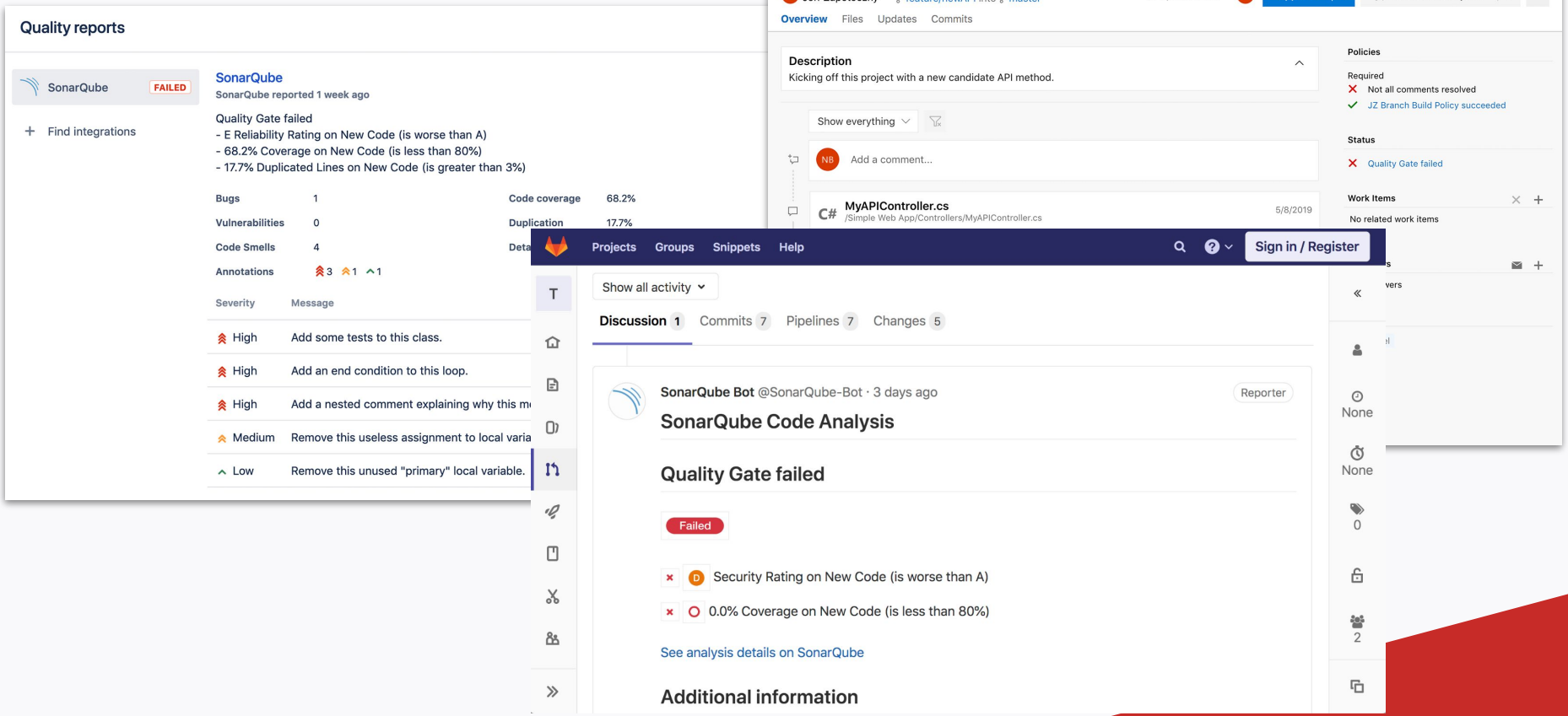
(Or use SonarQube Onboarding UI)

GitHub PR Decoration





...and other ALMs



2 ACTIVE **Adding a new API!**

Jeff Zapotoczny feature/newAPI into master

0/4 resolved NB **Approve** Set auto-complete ···

Overview Files Updates Commits

Description

Kicking off this project with a new candidate API method.

Show everything

NB Add a comment...

C# **MyAPIController.cs**

/Simple Web App/Controllers/MyAPIController.cs

5/8/2019

```
19 +                connection.ConnectionString = "db info";
20 +                connection.Open();
21 +                var selectSql = string.Format("select from MyStuff where id='{0}';", i
22 +                var selectCommand = new SqlCommand(selectSql, connection);
23 +
24 +                var dataReader = selectCommand.ExecuteReader();
25 +                return dataReader.GetString(0);
```

JZ **Jeff Zapotoczny** 5/8/2019 Active

Vulnerability: Refactor this code to not construct SQL queries directly from tainted user-controlled data. (roslyn.sonaranalyzer.security.cs:S3649)

See it in SonarCloud

NB Write a reply... **Resolve**

Policies

Required

- ✗ Not all comments resolved
- ✓ JZ Branch Build Policy succeeded

Status

- ✗ Quality Gate failed

Work Items

No related work items

Reviewers

No reviewers

Labels

Add label

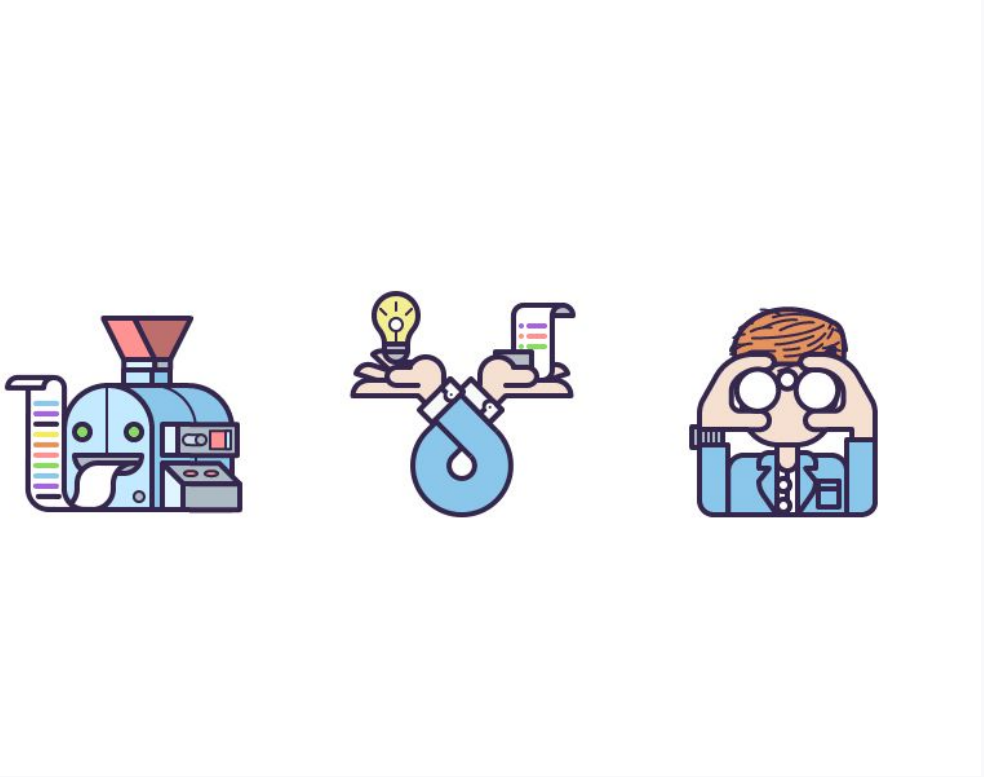
SonarLint

The ultimate shift left

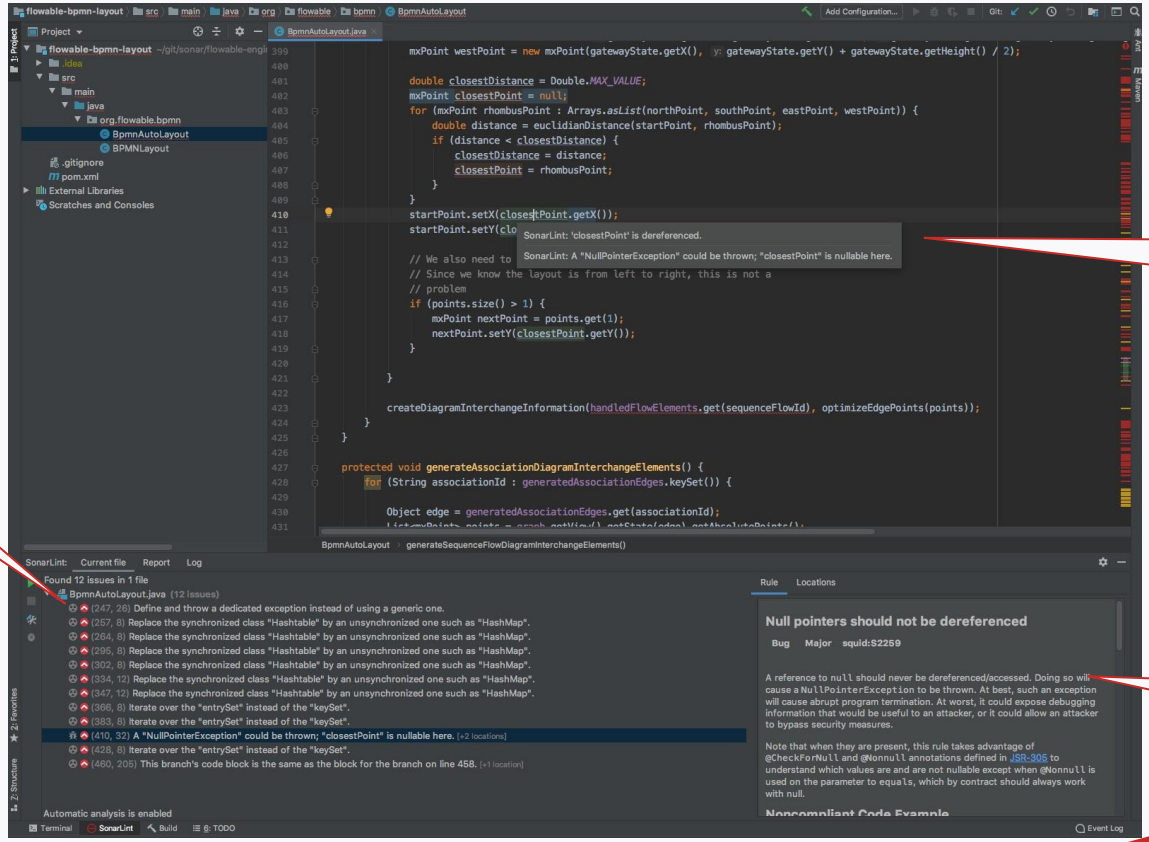
SonarLint key features

- Supports **five major IDEs**
  - Plus variants (e.g. Eclipse CDT, IDz/RDz, Topaz)
- **On-the-fly analysis** of code as you type
- Analysis of changed files or all files
- **Connected Mode**:
  - Analyze more languages
  - Align Quality Profiles with SonarQube
  - Suppress False Positives / Won't Fix
  - In-IDE notifications of key events in SonarQube
  - Hotspots and Vulnerabilities in the IDE

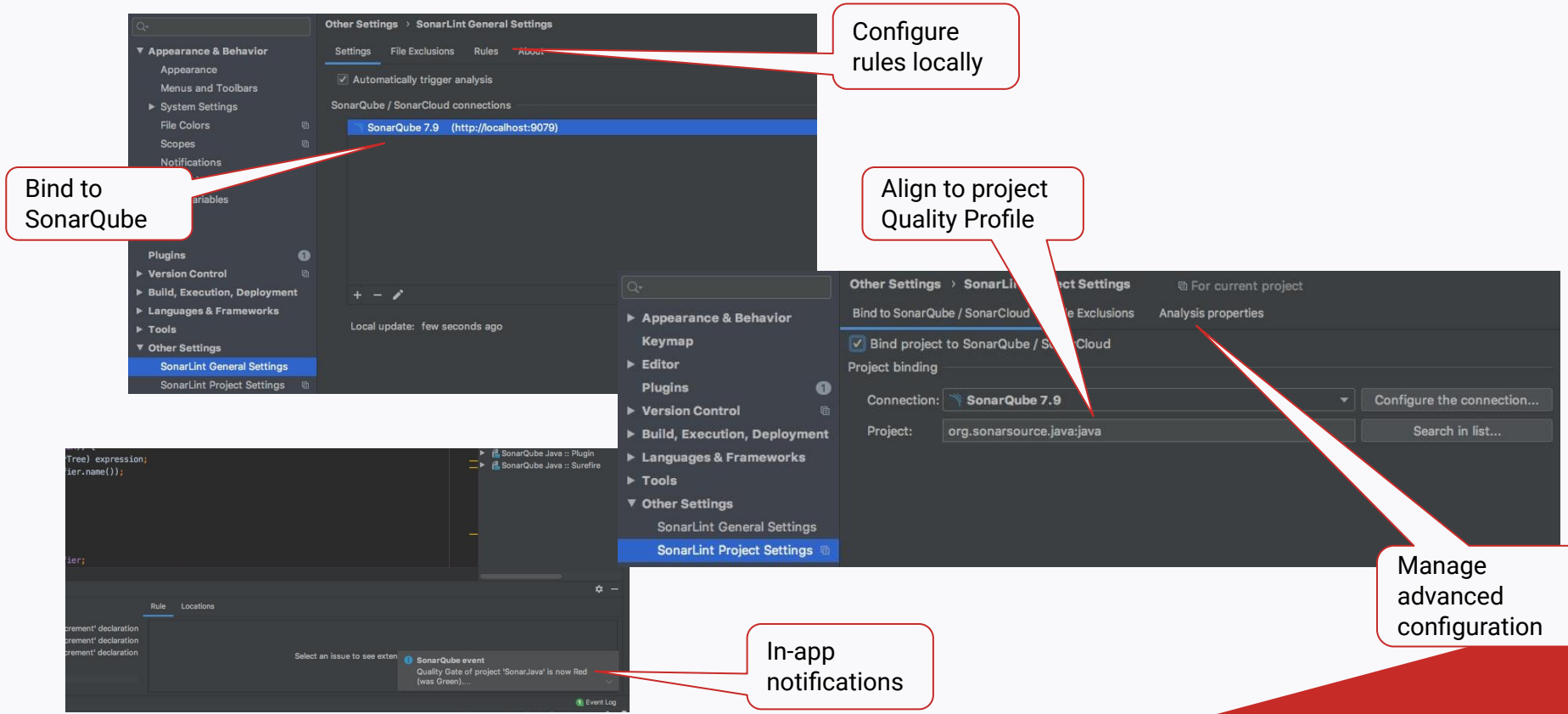
SonarLint Example



SonarLint key features



SonarLint connected mode

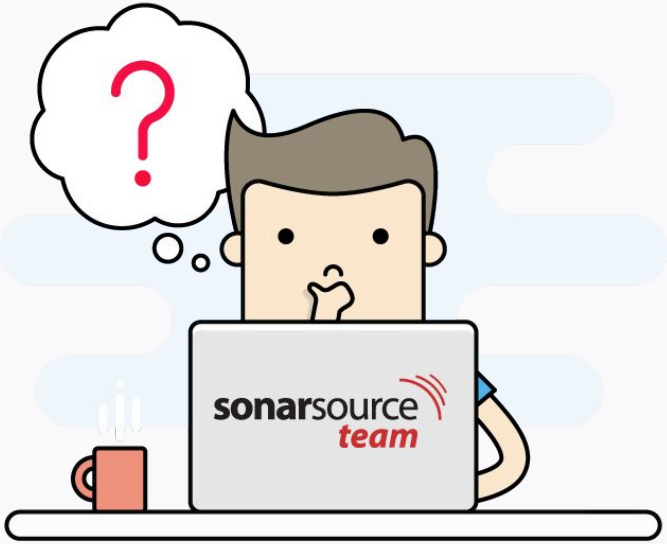


SonarLint language support

Which languages in which IDE?

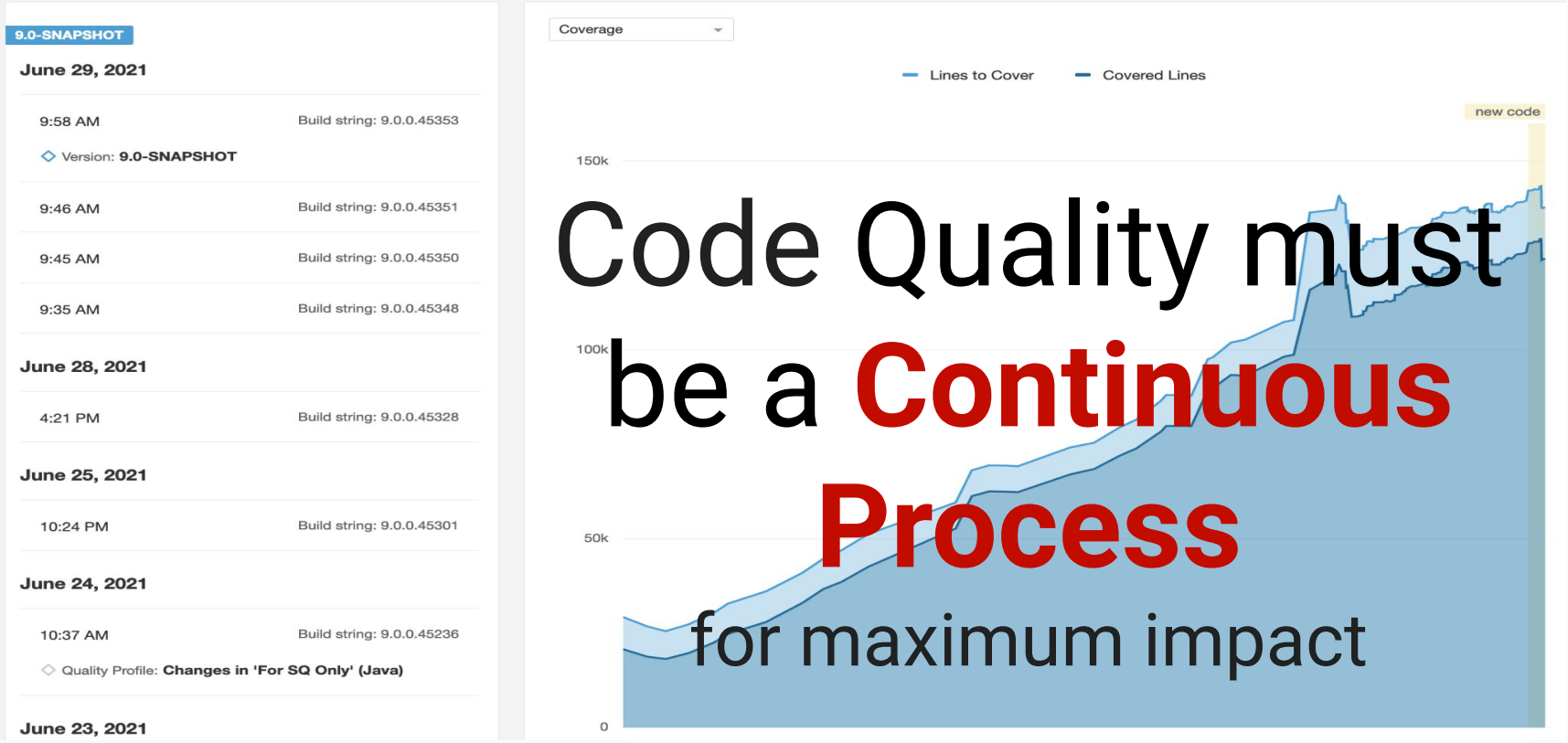
- Main languages for each IDE
  - E.g. Java for Eclipse, C# for Visual Studio
- Plus supplementary languages where possible
- Plus commercial languages in Connected Mode
  - E.g. COBOL in Eclipse
- Up-to-date list on https://www.sonarlint.org/
  - Choose your IDE to see the list

Questions ?



sonarsource

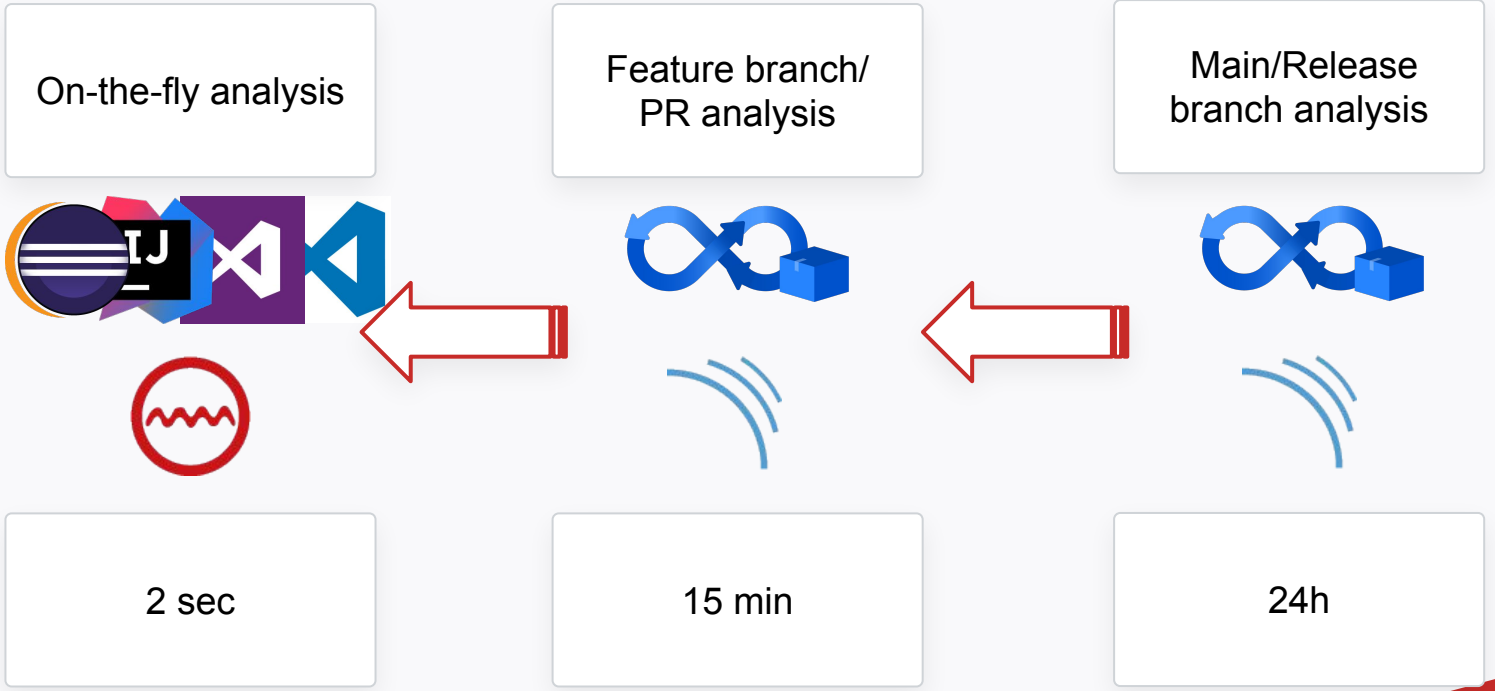
If you only remember a few things...


9.0-SNAPSHOT
June 29, 2021
9:58 AM
Build string: 9.0.0.45353
Version: 9.0-SNAPSHOT
9:46 AM
Build string: 9.0.0.45351
9:45 AM
Build string: 9.0.0.45350
9:35 AM
Build string: 9.0.0.45348
June 28, 2021
4:21 PM
Build string: 9.0.0.45328
June 25, 2021
10:24 PM
Build string: 9.0.0.45301
June 24, 2021
10:37 AM
Build string: 9.0.0.45236
Quality Profile: Changes in 'For SQ Only' (Java)
June 23, 2021
Coverage
Lines to Cover
Covered Lines
new code
150k
100k
50k
0
Code Quality must be a Continuous Process
for maximum impact


KEEP
CALM
AND
CLEAN AS YOU CODE

Shift Left

Shorten the feedback loop



Set and
Enforce
Quality Gates

Update Quality Gates
(and optionally Profiles)
on a regular basis

Feedback is a gift ! Thank you

http://tiny.cc/h21xlz

Questions ?