



ANALISI STATICA E QUALITA' DEL CODICE

43.22.28.159

CODE SMELLS

- **Code smells** are the results of bad programming practices
- They can cause many potential problems to quality attributes (e.g. performance, maintainability, testability, ...)
- They can cause failures, too
- Code smells are generally revealed by **static analysis**



CODE SMELLS

- There are many resources describing possible code smells and design anti-patterns
 - Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, Prentice-Hall
 - Robert C. Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design, Prentice-Hall
 - <https://blog.cleancoder.com/>





Good Programming Practice 2.2

Use white space to enhance program readability.





Good Programming Practice 2.3

By convention, every word in a class-name identifier begins with an uppercase letter. For example, the class-name identifier `DollarAmount` starts its first word, `Dollar`, with an uppercase D and its second word, `Amount`, with an uppercase A. This naming convention is known as **camel case**, because the uppercase letters stand out like a camel's humps.





Good Programming Practice 2.8

Choosing meaningful variable names helps a program to be self-documenting (i.e., one can understand the program simply by reading it rather than by reading associated documentation or creating and viewing an excessive number of comments).





Good Programming Practice 2.9

By convention, variable-name identifiers use the camel-case naming convention with a lowercase first letter—for example, `firstNumber`.





Good Programming Practice 2.4

Indent the entire body of each class declaration one “level” between the braces that delimit the class’s. This format emphasizes the class declaration’s structure and makes it easier to read. We use three spaces to form a level of indent—many programmers prefer two or four spaces. Whatever you choose, use it consistently.





Good Programming Practice 2.7

Place a space after each comma (,) in an argument list to make programs more readable.





Good Programming Practice 2.6

Indent the entire body of each method declaration one “level” between the braces that define the method’s body. This emphasizes the method’s structure and makes it easier to read.





Good Programming Practice 2.11

Indent the statement(s) in the body of an `if` statement to enhance readability. IDEs typically do this for you, allowing you to specify the indent size.





Error-Prevention Tip 2.4

You don't need to use braces, { }, around single-statement bodies, but you must include the braces around multiple-statement bodies. You'll see later that forgetting to enclose multiple-statement bodies in braces leads to errors. To avoid errors, as a rule, always enclose an `if` statement's body statement(s) in braces.





Common Programming Error 2.7

Placing a semicolon immediately after the right parenthesis after the condition in an `if` statement is often a logic error (although not a syntax error). The semicolon causes the body of the `if` statement to be empty, so the `if` statement performs no action, regardless of whether or not its condition is true. Worse yet, the original body statement of the `if` statement always executes, often causing the program to produce incorrect results.

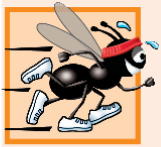




Error-Prevention Tip 2.5

A lengthy statement can be spread over several lines. If a single statement must be split across lines, choose natural breaking points, such as after a comma in a comma-separated list, or after an operator in a lengthy expression. If a statement is split across two or more lines, indent all subsequent lines until the end of the statement.





Performance Tip 7.1

Passing references to arrays, instead of the array objects themselves, makes sense for performance reasons. Because Java arguments are passed by value, if array objects were passed, a copy of each element would be passed. For large arrays, this would waste time and consume considerable storage for the copies of the elements.





Good Programming Practice 7.1

For readability, declare only one variable per declaration. Keep each declaration on a separate line, and include a comment describing the variable being declared.





Good Programming Practice 7.2

Constant variables also are called **named constants**. They often make programs more readable—a named constant such as `ARRAY_LENGTH` clearly indicates its purpose, whereas a literal value such as 10 could have different meanings based on its context.





Good Programming Practice 7.3

Constants use all uppercase letters by convention and multiword named constants should have each word separated from the next with an underscore (`_`) as in `ARRAY_LENGTH`.





Common Programming Error 7.4

Assigning a value to a previously initialized `final` variable is a compilation error. Similarly, attempting to access the value of a `final` variable before it's initialized results in a compilation error like, “`variable variableName` might not have been initialized.”





Good Programming Practice 8.1

By convention, predicate method names begin with `is` rather than `get`.





Good Programming Practice 10.1

When declaring a method in an interface, choose a method name that describes the method's purpose in a general manner, because the method may be implemented by many unrelated classes.





Good Programming Practice 10.2

Use `public` and `abstract` explicitly when declaring interface methods to make your intentions clear. As you'll see in Sections 10.10–10.11, Java SE 8 and Java SE 9 allow other kinds of methods in interfaces.





Good Programming Practice 11.1

Exception handling removes error-processing code from the main line of a program's code to improve program clarity. Do not place `try...catch...finally` around every statement that may throw an exception. This decreases readability. Rather, place one `try` block around a significant portion of your code, follow the `try` with `catch` blocks that handle each possible exception and follow the `catch` blocks with a single `finally` block (if one is required).





Good Programming Practice 11.2

Associating each type of serious execution-time malfunction with an appropriately named `Exception` class improves program clarity.





Good Programming Practice 11.3

By convention, all exception-class names should end with the word `Exception`.





Good Programming Practice 15.1

When building `Strings` that represent path information, use `File.separator` to obtain the local computer's proper separator character rather than explicitly using `/` or `\`. This constant is a `String` consisting of one character—the proper separator for the system.



FINDING CODE SMELLS

- Most of code smells can be found by static analysis
 - There are many active researches for the automatic finding of code smells from source code
 - Tools such as PMD, Checkstyle and SonarLint belong to this category

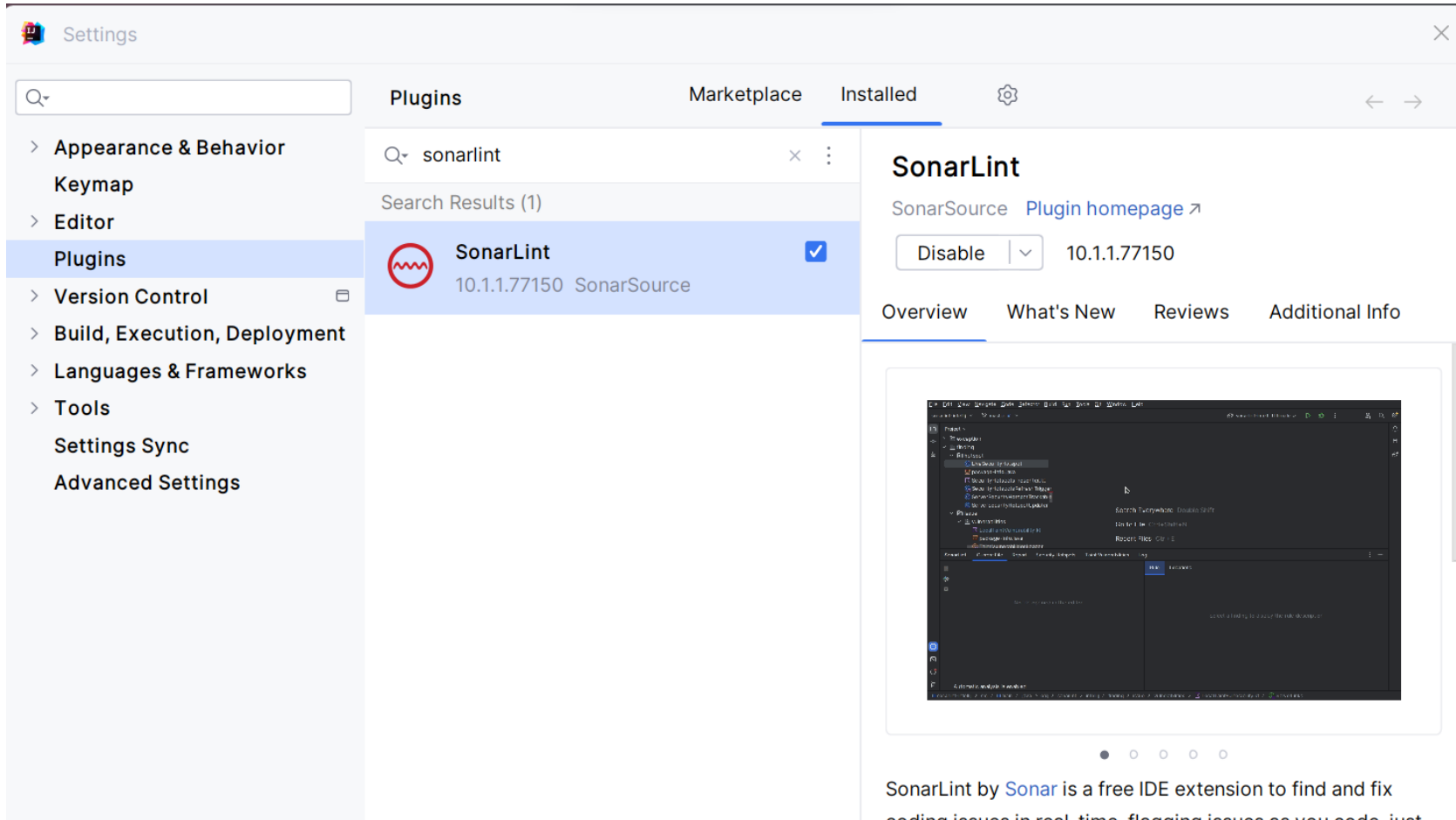


SONARLINT

- SonarLint represents one of the analyzers included in the SonarQube platform.
- It focuses on the analysis of the source code only and is freely distributed as a plug-in of all the most common IDEs
 - Differently from other parts of the SonarQube platform



INSTALLARE SONARLINT




The screenshot shows the IntelliJ IDEA Settings window with the 'Plugins' tab selected. The left sidebar lists various settings categories, with 'Plugins' highlighted. The main panel shows the 'Installed' tab for the 'SonarLint' plugin, version 10.1.1.77150 by SonarSource. The plugin is marked as 'Disable' and has a dropdown arrow next to the version number. Below the plugin information, there are tabs for 'Overview', 'What's New', 'Reviews', and 'Additional Info'. The 'Overview' tab is active, displaying a preview of the SonarLint IDE interface. The preview shows a code editor with a SonarLint sidebar on the right, displaying a list of issues. The issues are categorized by severity (Error, Warning, Info) and include details like the rule ID, description, and a link to the documentation. The SonarLint sidebar also shows a 'Run' button and a 'Refresh' button. The bottom of the preview shows a status bar with the text 'SonarLint by Sonar is a free IDE extension to find and fix coding issues in real-time, flagging issues as you code. Just'.

Settings

Plugins Marketplace Installed

Search sonarlint

Search Results (1)

 **SonarLint** 10.1.1.77150 SonarSource ☒

SonarLint

SonarSource [Plugin homepage](#)

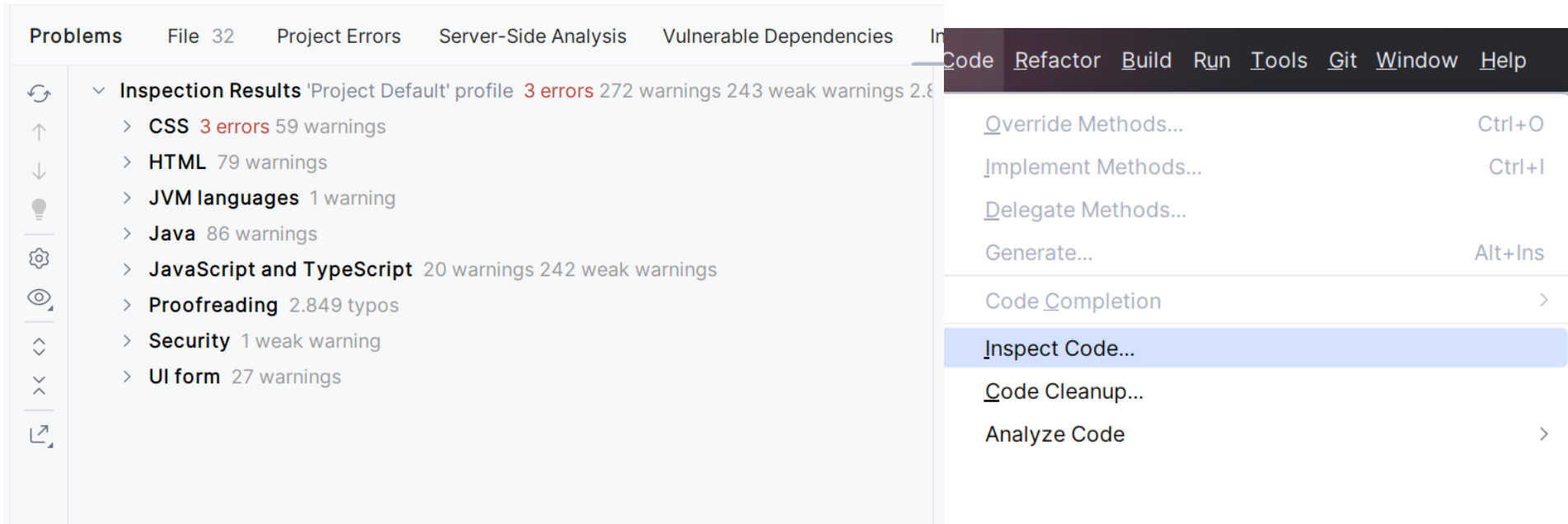
Disable 10.1.1.77150

Overview What's New Reviews Additional Info

SonarLint by Sonar is a free IDE extension to find and fix coding issues in real-time, flagging issues as you code. Just



UTILIZZARE SONARLINT



The screenshot displays an IDE interface with the SonarLint 'Problems' view on the left and a context menu on the right.

Problems View:

- File 32
- Project Errors
- Server-Side Analysis
- Vulnerable Dependencies
- Inspection Results 'Project Default' profile: 3 errors, 272 warnings, 243 weak warnings, 2.8
- > CSS: 3 errors, 59 warnings
- > HTML: 79 warnings
- > JVM languages: 1 warning
- > Java: 86 warnings
- > JavaScript and TypeScript: 20 warnings, 242 weak warnings
- > Proofreading: 2.849 typos
- > Security: 1 weak warning
- > UI form: 27 warnings

Context Menu:

- Code
- Refactor
- Build
- Run
- Tools
- Git
- Window
- Help
- Override Methods... (Ctrl+O)
- Implement Methods... (Ctrl+I)
- Delegate Methods...
- Generate... (Alt+Ins)
- Code Completion >
- Inspect Code...**
- Code Cleanup...
- Analyze Code >



ALCUNI PROBLEMI COMUNI

✓ JVM languages 1 warning

✓ Non-safe string is used as SQL 1 warning

✓ ListinoImplementazionePostgresDAO 1 warning

String, which is used in SQL, can be unsafe

✓ Code maturity 2 warnings

✓ Call to 'printStackTrace()' 2 warnings

✓ ConnessioneDatabase 1 warning

Call to 'printStackTrace()' should probably be replaced with more robust logging

✓ Performance 3 warnings

✓ 'String.equals()' can be replaced with 'String.isEmpty()' 3 warnings

> FinestraAcquistaAzione 1 warning

> finestraAggiungiSocieta 2 warnings

✓ No label for component 4 warnings

> FinestraAcquistaAzione.form 2 warnings

> finestraAggiungiSocieta.form 2 warnings

> Scrollable component not in JScrollPane 1 warning

✓ Declaration redundancy 30 warnings

✓ Declaration can have 'final' modifier 1 warning

✓ Home 1 warning

Declaration can have final modifier

> Empty method 1 warning

> Unused declaration 28 warnings

✓ UI form 27 warnings

✓ Hardcoded string literal in a UI form 13 warnings

> FinestraAcquistaAzione.form 3 warnings

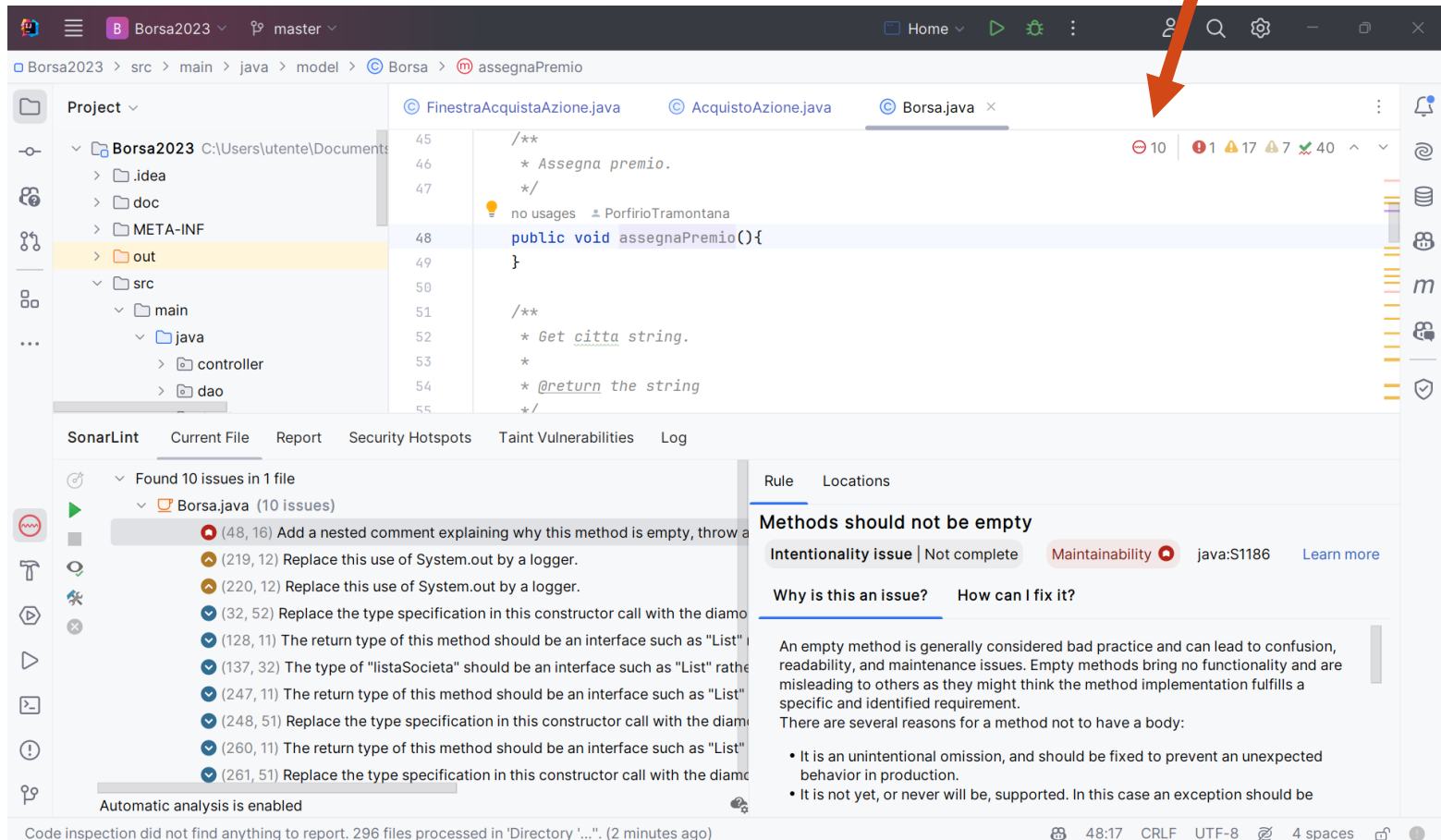
> finestraAggiungiSocieta.form 3 warnings

> Home.form 6 warnings

> ListaSocieta.form 1 warning



ANALISI DETTAGLIATA



The screenshot shows the IntelliJ IDEA IDE with a SonarLint analysis of a Java file named `Borsa.java`. The project is `Borsa2023` and the file is located in `src > main > java > model > Borsa > assegnaPremio`. The code editor shows the following code:

```
45  /**
46   * Assegna premio.
47   */
48  public void assegnaPremio(){
49  }
50
51  /**
52   * Get citta string.
53   *
54   * @return the string
55   */
```

The SonarLint panel at the bottom shows 10 issues found in 1 file. The issues are:

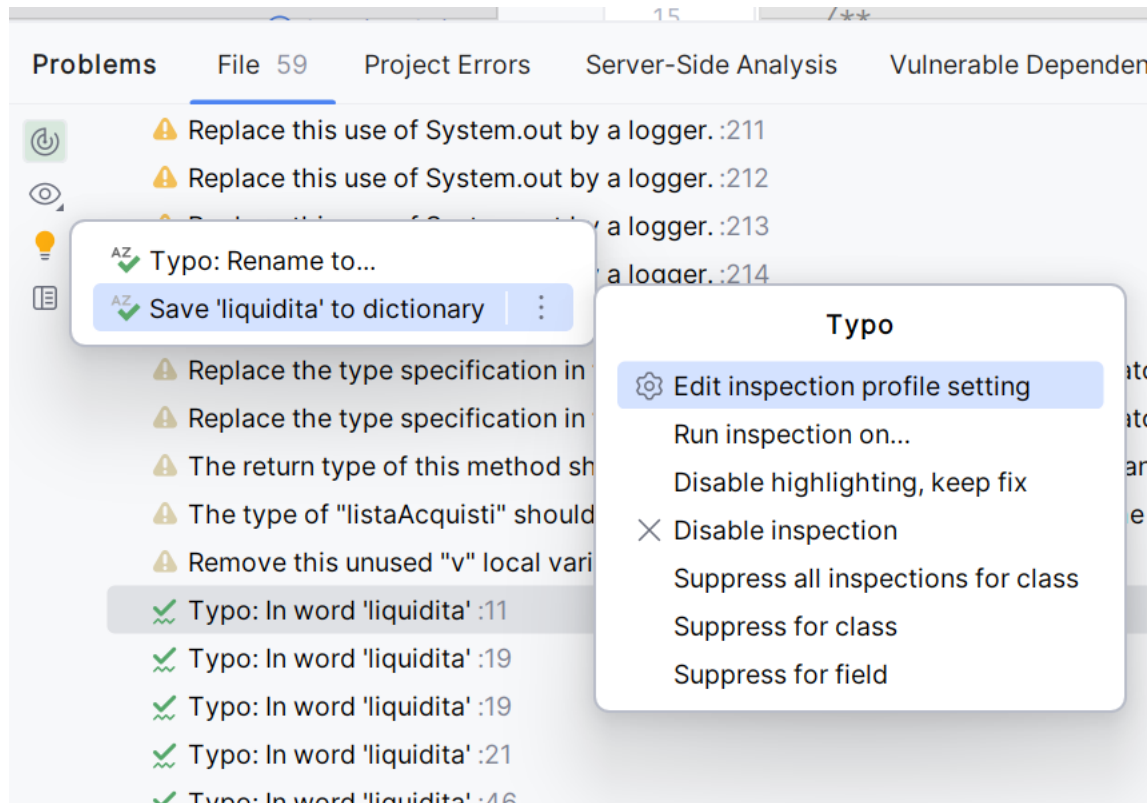
- (48, 16) Add a nested comment explaining why this method is empty, throw a
- (219, 12) Replace this use of System.out by a logger.
- (220, 12) Replace this use of System.out by a logger.
- (32, 52) Replace the type specification in this constructor call with the diamo
- (128, 11) The return type of this method should be an interface such as "List"
- (137, 32) The type of "listaSocieta" should be an interface such as "List" rather
- (247, 11) The return type of this method should be an interface such as "List"
- (248, 51) Replace the type specification in this constructor call with the diamo
- (260, 11) The return type of this method should be an interface such as "List"
- (261, 51) Replace the type specification in this constructor call with the diamo

The SonarLint panel also shows the rule `java:S1186` (Methods should not be empty) and provides a detailed explanation of why this is an issue and how to fix it.

Automatic analysis is enabled

Code inspection did not find anything to report. 296 files processed in 'Directory '...''. (2 minutes ago)

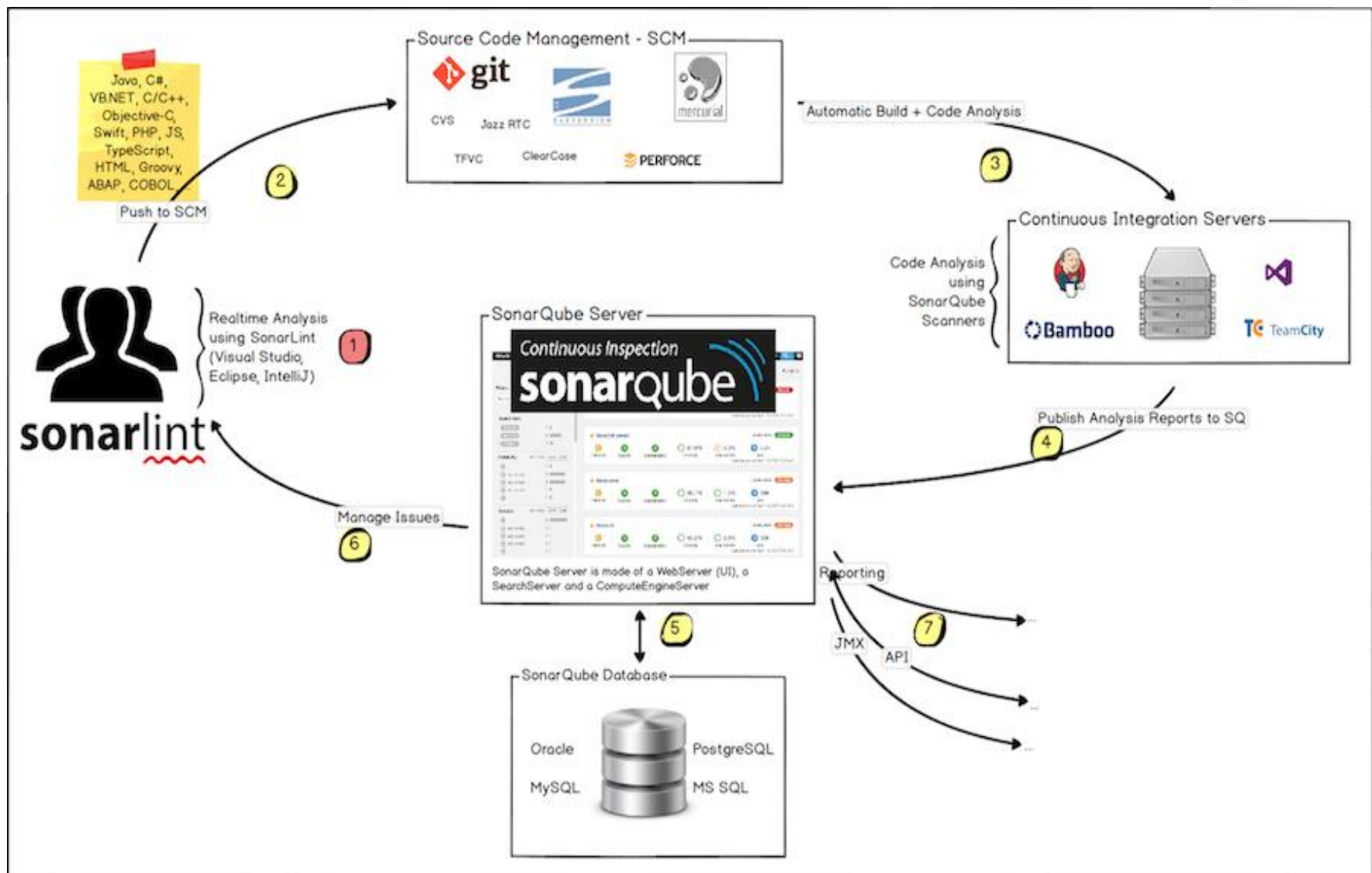
ACCETTARE SUGGERIMENTI



SONARQUBE

- SonarQube is an integrated open source tool that allows :
 - To perform many code quality testing and measurement tasks
 - To monitor code quality within a DevOps development cycle
- Available at <https://www.sonarqube.org/>





SONARQUBE PROCESS

1. Developers code in their IDEs and use SonarLint to perform local analysis during development
2. Developers submit their code in CVS (e.g. push to git)
3. The Continuous Integration Server (CIS) triggers an automatic build of the project and the execution of the SonarQube Scanner required to perform the analysis.
4. The analysis report is sent to the SonarQube server for processing.
5. SonarQube Server processes and stores the results of the analysis report in the SonarQube database and displays the results in the user interface.
6. Developers review, comment and address their issues to manage and reduce their technical debt through the SonarQube user interface.
7. Managers receive reports from the analysis.

