SourceMeter SonarQube plug-in

Rudolf Ferenc, László Langó, István Siket, Tibor Gyimóthy
University of Szeged, Department of Software Engineering
Árpád tér 2. H-6720 Szeged, Hungary
Email: {ferenc,lango,siket,gyimothy}@inf.u-szeged.hu

Tibor Bakota
FrontEndART Software Ltd.
Zászló u. 3 I./5. H-6722 Szeged, Hungary
Email: bakotat@frontendart.com

Abstract—The SourceMeter SonarQube plug-in is an extension of SonarQube, an open-source platform for managing code quality made by SonarSource S.A, Switzerland. The plug-in extends the built-in Java code analysis engine of SonarQube with FrontEndART's high-end Java code analysis engine. Most of SonarQubes original analysis results are replaced (including the detected source code duplications), while the range of available analyses is extended with a number of additional metrics and issue detectors. Additionally, the plug-in offers new GUI features on the SonarQube dashboard and drill-down views, making the SonarQube user experience more comfortable and the work with the tool more productive.

Keywords—SourceMeter, SonarQube, plug-in, software quality, metrics, clone detection, coding issues

I. INTRODUCTION

The source code of a program is usually its only up-to-date documentation. At the same time, the source code is the exquisite bearer of knowledge, business processes and methodology, accumulated over a long period of time. Source code quality decrease, which happens due to many quick fixes and time pressure, results in the increase of development and testing costs, and operational risks [1]. In spite of this, the source code usually receives hostile treatment and is merely considered as a tool.

SonarQube is an open source platform to manage code quality (http://www.sonarqube.org). It offers many features packaged in its basic installation, and it can be also extended with free and commercial plug-ins. Although the Java ecosystem of SonarQube has many useful features, its source code analysis engine provides only little information about the general quality of the code. For instance, it does not provide any metrics about individual Java classes or methods. Actually, the main focus and strength of SonarQube is the handling of coding issues. We developed our plug-in for improving the Java ecosystem of SonarQube.

FrontEndART has developed SourceMeter based on the Columbus technology [2] researched and developed at the Department of Software Engineering of the University of Szeged. Using the results of the analysis, the quality of the analyzed source code can be improved and developed both in the short- and long term in a directed way. By continuous static analysis, the software developers can:

- reduce the software erosion rate and this way decrease development costs;
- many coding problems can be identified before testing, so the number of test iterations and the testing costs can be reduced;
- the number of errors in delivered software can be reduced, so the operational risks can be decreased, increasing the company’s reputation.

All features of SourceMeter are free (can be downloaded from its homepage at http://www.sourcemeter.com), except the VulnerabilityHunter and FaultHunter modules. There is an online demonstration of the SourceMeter plug-in available at http://sonarqube.sourcemeter.com.

II. GUI FEATURES

SourceMeter SonarQube plug-in boosts the SonarQube GUI in various ways. In this section we list the added features.

A. SourceMeter Dashboard

The SonarQube Navigation menu on the left hand side is extended with two dedicated items: SourceMeter and...
the associated SourceMeter Help (see Figure 1) under the TOOLS section (the help is available also from the main page). The Help provides extensive documentation about the calculated metrics and provides a link to the User’s Guide.

The plug-in provides a new Dashboard to SonarQube, which includes a set of specifically crafted widgets as well as some built-in ones showing new metrics. This dashboard collects useful features provided by the plug-in and is accessible by clicking on the SourceMeter item on the main page.

B. Original SonarQube Dashboard

The original default Dashboard provided by SonarQube is not changed by the plug-in and can be used in the usual way. The main difference is that the measurement data shown on this page are actually the analysis results of the plug-in, which replaces the results provided by the default SonarQube analyzer. It can be observed that many metric results will be different than the results provided by the built-in analyzer. The reason is that the plug-in provides more accurate results.
C. Source view

The plug-in provides an enriched source view window (see Figure 2) that can be reached by selecting a source code element like class or method in different GUI windows (select SourceMeter: Source tab). It is accessible from various places including the Dashboard widgets (Hotspots, Size, Comments, etc.) and the well-known SonarQube drill-down view.

In addition to the built-in Source view, the SourceMeter: Source view provides the following features:

- The built-in drill-down mechanism is limited to folders and files. The plug-in extends the drill-down with classes and methods, which are actually new resource kinds added by the plug-in.
- Classes, including nested, local, and anonymous ones are listed and can be selected in the extended drill-down view.
- Methods of the selected class are listed and can be selected in the extended drill-down view.
- The selected class or method is highlighted in the source code text pane.
- The middle part of the window shows various metrics (more than 60) and coding problem summaries corresponding to the selected item (class or method). By default, only the most important metrics are shown; however, the others can also be made visible and hidden again by clicking on the Expand/Collapse link. The metrics can have one of the following three colors: red means that the metric is above the preset threshold, while green shows that the value is within this limit, finally those metrics that do not naturally have a threshold value are shown in black. The threshold values can be set on the Settings page under the SourceMeter categories (see Section II-G). Online help is available in a pop-up window for all metrics by clicking on their names.

D. Clones

The plug-in includes a sophisticated Type-2 clone detection engine (detects syntactically identical source code fragments except for variations in identifier names, literals, type references, whitespace, layout and comments)[3], which provides the following benefits over the built-in version of SonarQube, which checks only for text similarities:

- The plug-in takes into account the syntactic structure of the source code; hence no broken code fragments without syntactic structure will be reported.
- The plug-in calculates a number of very useful metrics related to clones.

Also, the plug-in provides an enriched view for the detected clones1 (see Figure 3). The clone view is available by selecting SourceMeter: Clones tab in different GUI windows (including the Clone Class Hotspots and Clone Instance Hotspots widgets - see Section II-F - and the drill-down view). Over the built-in version, the clone view provides the following features:

- All the related clone metrics are displayed. Similarly to the source view, the metrics are shown in red if the value is above the preset threshold, in green if the value is within the limit, or in black if there is no threshold value associated. Online help is available in a pop-up window for all metrics by clicking on their names.
- Various clone metrics can be used to sort the clone classes (select the desired metric from the Sort Clone Classes by drop-down list).
- Navigation from other views (for instance from the SourceMeter dashboard and widgets). In this case the selected clone class and clone instance will be highlighted.

1Two code segments correspond to each other if they are copies of each other. This relation is an equivalence relation and we use the notion of clone classes to the classes of the relation, and the members of the classes are referred to as clone instances. Owing to the nature of the relation, each clone class must contain at least two clone instances.
E. Issues Drilldown

This page is essentially not modified, but the plug-in includes powerful new coding issue detection not provided by SonarQube:

- OWASP (https://www.owasp.org) vulnerability issues will be reported as issues falling in the Blocker severity category (VulnerabilityHunter module; license key is required). Opposed to other coding rule violation where one source position is enough for showing the problem, vulnerability issues reveal a possible execution path from a source to a given sink, which means that the developer needs the trace for understanding the problem and fixing the issue. Therefore, the whole trace, which describes the possible execution path from the source to the sink, is also available at the warning (see Figure 4). Each element of the trace is presented in separate lines and consists of a file name, a line number in round brackets and a call stack depth in square brackets (counted starting from the source). To help the developer investigating the problem, each trace element is a link, and by clicking on it the given file will be opened in a new browser window and the source code will be positionned to the given line.

- Common programming mistakes detector, which reimplements many PMD (http://pmd.sourceforge.net) rule violation checks and provides less noise (false positives) and a number of real problems not detected by PMD (true negatives). It also provides a number of checks not available in PMD (FaultHunter module; license key is required).

- The PMD rules have been carefully selected and reprioritized by our software developer and QA experts. For these rules the plug-in uses the Minor, Major and Critical categories. If a valid FaultHunter license key is available, then the reimplemented PMD rules will be switched off automatically and FaultHunter will provide the results instead. If no license key is available (free version), then the original PMD rule violation checks will be executed.

- All metrics that do not fall within the preset threshold values will be reported as new issues (coding rule violations) falling into the Info severity category. This enables its use in various statistics, drilldown and source code view, for instance.

F. Hotspot Widgets

The plug-in provides new types of hotspot widgets, which can be accessed on the Configure widget page under the SourceMeter category. Added widgets are the following:

- Class Hotspots
- Method Hotspots
- Clone Class Hotspots
- Clone Instance Hotspots

These widgets can be used to list the corresponding types of entities in the order of decreasing values of a selected metric, which can be set by using Edit. Figure 5 shows example hotspot widgets. For instance, the upper left-hand
side widget lists the top 5 most complex methods in the analyzed project (the metric values are shown in red if they violate the preset threshold).

G. Settings

After logging in with administrator rights, various settings for the plug-in will be available under the SourceMeter, SourceMeter: Class Thresholds, SourceMeter: Clone Thresholds, and SourceMeter: Method Thresholds categories (see Figure 6).

![Figure 6. Metric thresholds settings](image)

The first category can be used to change general settings for the plug-in, while the other three are for setting the threshold values for the different metrics which will be used e.g. in the source code view, the new hotspot widgets, or the drilldown pages. The metric categories can be selected on the top part of the page (e.g. SM:Cohesion, SM:Complexity). Where appropriate, the default values are also shown.

III. ARCHITECTURE

Figure 7 shows the basic layout of the architecture of SonarQube with the SourceMeter plug-in. After installing the SourceMeter plug-in and setting the corresponding properties, the user does not have to do anything else, the analysis can be done in the regular way. This means that if e.g. SonarQube Runner is used and the necessary properties of the Java project are given, the user has to start SonarQube Runner as usual.

SonarQube Runner first establishes connections to the SonarQube Server and the database, then it downloads the required plug-ins from the server (thick white arrow in the figure). In case of the Java language, the SonarQube Java Ecosystem components do the analysis including metric calculation, code duplication and coding issue detection, building the hierarchy of the source code and storing the source file, etc. In the analysis phase, the original analyzer runs first but the SourceMeter plug-in disables the original code duplication and PMD coding issue detectors because the plug-in provides these results more accurately.

![Figure 7. Architecture of SourceMeter integration](image)

Next, the SourceMeter plug-in is activated and it invokes the SourceMeter command line toolchain with the appropriate parameters. The command line toolchain analyzes the Java source code, which means it builds the extended hierarchy of the system (including classes and methods), calculates metrics, detects code duplications and finds coding issues. The results are stored in binary and xml file formats for any further utilization, while the user can easily use the txt files that contain the coding issues in human readable form and the csv files where each csv file contains all metrics of a given source element kind (e.g. files, classes or methods). The temporary and log files inform the user about the technical details of the analysis and help to find eventual problems. (The SourceMeter command line toolchain can be of course used also separately, as a standalone tool, without using the SonarQube plug-in.)

After the command line toolchain finishes the analysis, the SourceMeter plug-in uses the results to update the original metric values calculated by the SonarQube analyzer, and to extend the results with class, method and code duplication level elements and their metrics. Fortunately, SonarQube provides a Java API for doing this, so no direct database operations are required.

To present the new element kinds and results, we improved the SonarQube graphical user interface as described in Section II. To achieve this, the SourceMeter plug-in enriches the appropriate SonarQube pages. This is mainly done by using the standard SonarQube Java API, but in certain rare cases we had to use JavaScript coding as well.

The SourceMeter plug-in extracts over 10 times more information from the source code than the original SonarQube analyzer does, which results in an increased memory usage.
and run time on the server side of SonarQube (on average, both increase typically by a factor of 3-5). Unfortunately, for very large projects (over 1MLOC) the run time of the data uploading on the server side of SonarQube increases exponentially and might become unacceptably slow. To speed up the analysis, the plug-in provides a simplified mode where the methods and the method level results are not uploaded but all other results remain the same.

IV. USE CASES

Although SonarQube provides different possibilities for the users to explore their project, the new features of the plug-in offer the user new dimensions in maintenance and quality assurance. Here we present some useful scenarios.

Use case 1: SonarQube (and the tools behind it) assigns each coding issue problem to one single source code line which is sufficient in most cases. On the other hand, the VulnerabilityHunter module of SourceMeter searches for such execution paths in the code on which the data can be transferred from a source method (e.g. reading input from web form) to a sink method (e.g. executing database query) without validation (utilizing control and dataflow analysis). This means that a simple warning message for a code line is not enough for the developer to validate or to correct the problem. Therefore, the plug-in improves the description of the issue with the corresponding trace information to help the developer to investigate the suspicious path (see Figure 4). All elements of the trace are hyperlinks which open the corresponding file positioned to the appropriate line in a new window, so the developer can easily navigate through the suspicious execution path.

Use case 2: SonarQube detects code duplications to a certain extent (using simple string matching) and calculates basic corresponding metrics, like the number of duplicated lines per file, so the user is able to search for those files that contain the most duplications, but it is not possible to find e.g. the longest or most complex code duplications (clone instances). To get around this weakness the plug-in introduces two new resource kinds: Clone Class and Clone Instance, which allows the user to define new hotspots based on them. For example, starting from the SourceMeter Dashboard (see Figure 5) the user can easily find those code parts that are copied to the most places (Clone Class Hotspots by Clone Instances), or the longest (Clone Instance Hotspots by Clone Lines of Code), or the most complex (Clone Instance Hotspots by Clone Complexity) copied parts. This information can be used e.g. to provide input for refactoring activities.

Use case 3: All source code elements must be tested before releasing a software system but due to the limited resources and strict deadlines this is in many cases impossible in practice. Therefore, to achieve the best efficiency in testing, it is important to focus on the most problematic parts of the system. Earlier research showed that more complex or coupled classes and methods are more fault-prone [4], therefore it is worth focusing on them. Although, in case of Java, there is a strong connection between a Java file and its public class, this association is not enough to select the most dangerous classes for testing (and says nothing about the methods). The plug-in introduces and handles Java classes and methods as well, which means that it is possible to define new hotspots based on their metrics. For example, the SourceMeter Dashboard highlights the most problematic classes and method from complexity and coupling point of view, but arbitrary other hotspots can be defined based on the numerous metrics. For instance, Figure 5 shows the top 5 most complex methods (Method Hotspots by McCabe's Cyclomatic Complexity).

V. CONCLUSION

Because of its easy usage and the support of many programming languages, SonarQube became a popular and widespread tool. But the “shallow” code analysis technique of its underlying analyzer engine and the low precision of the integrated tools (e.g. PMD) result in imprecise metrics and many false positive coding issues, which are degrading the practical usability of SonarQube.

In this paper we introduced the SourceMeter SonarQube plug-in, which eliminates many weaknesses of SonarQube, for free. The plug-in provides more precise and new metrics for the users, it provides more accurate duplicated code detection, and reprioritizes the PMD coding rules and switches off the useless (e.g. too noisy) ones. Additionally, the plug-in provides new resource kinds (class, method, clone class, clone instance) for the user to be able to examine their metrics, and it also provides enhancements to the graphical user interface of SonarQube.

REFERENCES


