Integrated Metrics Handling in Open Source Software Quality Management Platforms

Julio Escribano-Barreno, Javier García-Muñoz and Marisol García-Valls

Abstract Software quality is of vital importance in software development projects. It influences every aspect of the system such as the functionality, reliability, availability, maintainability, and safety. In critical software projects, quality assurance has to be considered at each level of the initial concept to the software engineering process: from specification to coding and integration. At the lowest coding level, there are several tools that enable the monitoring and control of software quality. One of them is SonarQube, an open source quality management platform, used to analyse and measure technical quality. I can be extended through plugins for customization and integration with other tools. The specific conception and development of these plugins is a significant design effort that ensures the correct handling of the different phases involved in the software quality process. We present an initial design and development of an integrated analyser component for extending the functionality of the open source framework for software quality management.

Keywords Software quality · Monitoring of verification · Software design

1 Introduction

Every software development process needs information about almost every aspect of the software development phase, like achievement of objectives, monitoring and control of activities, project costs and technical quality. Metrics are of key importance in all engineering disciplines and, in particular, for software development (see [13]),

J. Escribano-Barreno
Indra, Alcobendas, Spain
e-mail: jebarreno@indra.es

J. Escribano-Barreno · J. García-Muñoz · M. García-Valls(✉)
Universidad Carlos III de Madrid, Leganés, Spain
e-mail: {jgmunoz,mvalls}@it.uc3m.es

S. Latifi (ed.), *Information Technology New Generations*,
Advances in Intelligent Systems and Computing 448,
DOI: 10.1007/978-3-319-32467-8_45

providing a vital insight into the development process to assess maintainability, reliability and even development progress. Metrics provide reproducible indicators useful to estimate the quality, performance, management, and cost within a project. They bring in benefits like the possibility of analysing the data to understand, improve, and predict future behaviours for undertaking corrective actions on time.

Metrics has been studied and developed through years in order to improve software and systems development. They have increased their relevance due to their applied use and the contrasted benefits to define baseline quality indicators for serveral purposes. For example, the SEI Capability Mature Model Integration (CMMI) [9] for development, relies on the usage of metrics (see [17]), and is used for evaluating the maturity process of the organizations.

The collection and processing of the metrics can involve a significant human effort, that highlights the need of automating the specification of metrics and subsequent data collection that must later be processed. By using automatic analysis tools (such as [4] for static code analysis), the metrics collection effort is significantly reduced. One of the platofms that supports this process is Few open source platforms support the automatic metrics management for this process. Among them, the most popular quality management platform is SonarQube [10][43]. It enables continuous inspection and it supports a number of languages, including Java, C, C++, C#, PHP, and JavaScript. SonarQube provides some *basic* metrics like complexity, duplicated code detection, or lines of code counting, among others. However, this is a very generic functionality that requires to be enhanced for software development projects of a certain complexity. Highly complex projects such as critical software projects require that these basic metrics be enhanced to provide the information required by each particular project, as each project may have to adjust to specific normd. It is then useful and needed to take information from other sources, like other external tools, and stablish the adequate methodology in order to enhance the functional power of this framework in order to provide the required information with a suitable design that makes it easily customizable.

In this paper, we present an approach to enrich the metrics management and presentation to verification engineers based on the SonarQube framework. This has been performed by designing an *integrated metrics analyser* component that provides the enhanced functionality for the platform in order to integrate its own analysis results with the ones from external analysis tools in a single presentation space. The design is a modular one that allows to easily customize the integration of any external code analysis tool. The result is the achievement of a more complete set of metrics that can be managed in the projects to control and monitor the software development process according to the needs of each specific project. The information from the analysis of th project code is then centralized in the platform that is, at the same time, a collaborative environment that allows the remote work of teams of verification engineers. We validate the component by implementing the specific integration with external information sources that provide static analysis metrics (such as Understand [53]); this external tool provides metrics and rule-checks against both, custom and published standards. In our work, the rules and metrics can also be extended to

complement the provided ones. We exemplify technical metrics, related to software quality through actual static analyses on a real critical software project.

The paper is structured as follows. Section 1 includes a brief introduction and motivation for this work. Section 2 describes the related work in what concerns norms and practices related to technical quality and metrics. Section 3 describes the baseline framework offered by SonarQube. Section 4 presents our contribution in the form of a new functionality for the analysis of metrics from different tools, presenting the design details on the enabling plugin for this functionality. Section 5 validates the design through a concrete implementation for an external tool and we show its usage in the context of a real-world software project. Section 6 concludes this paper and presents the continuation work.

2 Background and Related Work

This section describes selected work most related to the objective of the present contribution, mainly concerning the existing norms and regulations for critical software development, the engineering processes that describe the steps to the objective software development, and current tools for software code analysis.

2.1 Norms and Software Quality Tools and Frameworks

Technical metrics are collected through static analysis techniques. Software static analysis is required in several norms related to critical systems, some of which are described here. The norms selected here require the use of static analysis techniques to comply with their objectives. For example, DO-178C [46] and DO-278A [47] introduces the use of metrics to be specified in the Software Quality Assurance Plan. The metrics collection and analysis, additionally supports the compliance of some norms, like North Atlantic Treaty Organization (NATO) AQAP-2210 [2], and its spanish version PECAL-2210 [39].

DO-178C is the norm for *Software Considerations in Airborne Systems and Equipment Certification*, [46]. It is one of the most accepted international standards. Includes additional objectives and it is complemented with the supplements [49], [50], [51] and [52]. The previous version of this norm is DO-178B [48]. DO-278A is the norm for *Guidelines for Communication, Navigation, Surveillance and Air Traffic Management* [47]. It provides guidelines for non-airborne Communication, Navigation, Surveillance and Air Traffic Management (CNS/ATM) systems, including the guidelines for the software assurance activities to be conducted with non-airborne CNS/ATM systems. IEC 61508 is the norm for *Functional safety of electrical/electronic/programmable electronic safety-related systems* [31]. It is the standard for industry automation, intended to be a safety standard applicable to all kinds of industry that includes the complete safety life cycle. Used as basis for other documents, as

railway (CENELEC 50128 [34]), automotive industries (ISO 26262 [31]) or nuclear power plants (IEC 61513 [32]).

The metrics and quality information with respect to a complex software project may easily require different analysis techniques that generate results and data from different sources, possibly also collecting data in different ways. In most projects, more than purely the source code is analyzed, e.g. dependencies among packages such as [14]. Information about the software is collected in different ways by means of different tools such as the ones presented as follows. *Understand* [53] and LDRA [37] are commercial tools for static code analysis, supporting multi-language. PC-Lint [41] is also a commercial solution for static code analysis supporting C and C++ languages. Splint [44] is a GNU licensed tool that supports static code analysis for C programs. PMD [42] supports static code analyser for Java, JavaScript, XML and XSL. SonarQube [10, 43] is an open source quality management platform, developed under LGPL v3 license.

As not a single tool is capable of providing all required data, it is typically required to set up a tool chain for code analysis setting also the procedures and principles for information collection and interpretation. in the form of a collaborative environment. As a result, the SonarQube platform has appeared as a platform to support advanced analysis; however, SonarQube appears in a similar way to a blank sheet of paper, so that the required techniques and methods to implement the needed functionality have to be designed and integrated in it.

The tools mentioned in section 2 are used to statically analyse the code quality. However, the metrics that they yield do not meet all the requirements across different projects. In each project, different metrics can be required, or different implementation languages can be used that vary from a project to another. This situation can make that one tool that is suitable for a project is not valid to other one.

For example, SonarQube is a quality management platform that can reflect and present the different quality aspects of several projects, coded in a number of different languages. This is carried out by the design and implementation of modules that can extend the functionality of the platform. Apart from the very basic set of metrics and data that SonarQube provides for any given project, a mechanism to customize it for particular projects that have different information requirements is needed. This characteristic can be complemented with the metrics available in other external code analysers (such as Understand) to enrich and extend the capability of SonarQube.

One other missing element in the current status of SonarQube is that each user needs to have the tool installed locally; as such, the run-time environment has little orientation to be a collaborative environment that can simultaneously support their access to several projects with a number of users.

We overcome these missing characteristics with the design and implementation of enhancements to SonarQube to support the customizable integration with external tools for code analysis. We design the required plugins to enrich the basic metrics presentation functionality of the platform, combining them in an unique repository that provides collaborative access to different (though shared) software projects.

2.2 *Software Architectures in Critical Systems in Emerging Domains*

An essential element of critical systems is the software architecture as it directly impacts the source code quality and the complexity of the final development. Critical software systems validation focuses, among other aspects, on temporal behavior applying real-time techniques [1, 6–8, 15]. These differ to some extent in the thorough application and verification of the temporal properties, rather providing quality of service mechanisms that are embedded in the software logic. These mechanisms allow dynamic execution preserving timely properties [11, 21–23, 25, 45]. Software quality should account for the verification of the properties of distributed software also related to newer domains as cloud [18], the characteristics of the middleware are integrated in the model [19, 20]. Critical software systems only tolerate off-line and design time verification; however newer domains such as cyber-physical systems require verification techniques to be applied on-line [5, 24]. For distributed environments based on middleware such as [29, 30], on-line decision on correctness of the system composition is applied (e.g. [26–28]). In such emerging domains, the software quality frameworks will have to device new ways of considering properties that will only emerge at execution time.

3 SonarQube Overview

SonarQube is a software quality management platform, multi-language, capable of performing simple analysis over the source code. Basically, it provides information about duplicated code, unit tests, coding standards, code coverage, code complexity, comments, and software design and architecture. The functionality and capabilities of SonarQube can be extended with new modules, namely *plugins*. This allows to integrate support for additional programming languages, additional metrics, or the integration with other tools that bring in new functionalities.

The underlying logic of SonarQube is based on a *source code analyzer* component that performs basic analysis activities (such as counting lines of code) and an *application server* that graphically displays the data that results from the analysis in a browser front-end. The following elements are key to the internal function of the SonarQube framework:

- *Widgets* are the components that configure the graphical display of data resulting from the source code analysis, i.e., it enables the customization of the analysis results presentation to the user. A widget supports the specification of the visual format and display locations of the presented data. Each widget yields one of the square boxes that are shown. Each box contains a number of data items whose display location and characteristics is indicated in the widget code. For each new analysed project, SonarQube creates a *project dashboard* for selecting, adding, or removing the available widgets.

- *Sensors* are components that access the specific source code to be checked and that support the implementation of analysis functions over the source code to extract the metrics.
- *Decorators* are the components that support the programming of additional processing over the initial metrics provided by sensors in order to derive more complex metrics.

The framework comprises a key component, Sonar Runner, that controls the sequence of steps to launch the source code analysis. In order to execute a SonarQube analysis, it is needed to initially launch *Sonar Runner* that is determines the sequence of invocations of the functions provided by the *Sensors* and *Decorators*. Once the process is completed, the *runner* stores in a database all the collected data.

4 Metrics Integration in the Quality Framework

This section presents the design and implementation of the integrated metrics analyser that provides the enhanced functionality for SonarQube to integrate its own analysis results with the ones from external analysis tools in a single presentation space. The enhancement has been done via a plugin that integrates the external data and displays the integrated metrics in the project dashboard. The design is flexible and modular in order to support the integration of any external tool with minor modifications.

4.1 Addition of Metrics

SonarQube provides basic analysis facilities, yielding very basic metrics over the code. In software projects for critical systems, the specific standards that must be applied require more complex metrics over the code. A few examples of these are: complexity, nesting levels, function parameters, and other values derived from the previous ones such as maximum, minimum and mean values for each of the previous metrics. These are not provided by SonarQube. However, there are other specific external tools that do provide a broad range of complex metrics.

With the enhancement of SonarQube quality management platform, users view richer information over a software project code by using SonarQube as the single front-end, presenting a number of metrics in the project dashboard, as an additional widget.

Following, the structure of a sensor is provided. The `Sensor` interface is a tagging components, i.e., a class extending this interface is automatically a sensor as it is obligued to implement the `analyse` method to provide a customized functionality for the sensor.

```
public interface Sensor extends BatchExtension, CheckProject {
    void analyse(Project module, SensorContext context);
}
```

The design of the integration plugin has to allocate modules to provide the logic for: (*i*) storage of the analysis results from the external tool (the external analysis data); (*ii*) use and extend a sensor template to locate and acces the external analysis data; (*iii*) overwrite the `analyse` method to scan and collect the external analysis data; and optionally (*iv*) design and implement a *decorator* that computes additional metrics from the data collected by the sensor.

4.2 Software Design

The architecture of the analyzer software module is explained below, containing the following clases:

- *ExternalToolPlugin* is the class containing the specification of the properties of the analyser module. An arbitrary number of properties can be specified. For the integrated metrics analyser, `sonar.externaltool.metrics` is the basic property to specify the path to the external analysis results. Other possible properties are programming language and language.
- *ListMetrics* is the class that specifies all metrics to be used (displayed) by the plugin. Metrics should be specified by name, type, description, qualitative or quantitative and domain. Sensors later assign values to each metric of the list as a result of the source code analysis done by SonarQube or by some other external tool. Here, this class should contain all metrics provided by the external tool, that are precisely the external analysis results.
- *ExternalToolMetricSensor* class contains the functionality to scan the analysis data produced by the external analysis tool in order to collect the metrics that it provides. This class is invoked when the *Runner* component is executed.
- *MetricsRubyWidget* class contains the definition of the properties of the widget, the title of the widget, and the the design and display characteristics of the data to be included in the project dashboard. Precisely, the file containing the information about the design and display characteristics (`html.erb`) that contains the template for such a design and the positioning of the data.

5 Implementation Details for Validation

For the validation, a real project developed under norms [46] and [2] was analysed with our software module. Understand was selected as the external tool to validate the analyser module that integrates different sources of analysis results.

The first step to integrate Understand analysis into the SonarQube framework was to develop an extension of the Understand metrics to provide the required files for SonarQube.

Input files were provided by the external tool and contained all the metrics that the integrated analyser module is able to detect. The text marked as *Free text to include comments* will not be analysed by the module.

The class *ExternalToolMetricSensor* of the integrated analyser module is extended to derive the class *UnderstandMetricSensor* that supports the specific characteristics of this specific external analysis tool. Consequently, when the *Runner* component is executed, this class reads the required output file from Understand to derive its analysis metrics.

For the class *MetricsRubyWidget* that defines the properties to customize the widget and the data display, the specific data for Understand is given:

- *getId()* returns an identifier to the external tool Understand that is *UnderstandMetrics*
- *getTitle()* returns *Understand Metrics*.
- *getTemplatePath()* that definesto the file *html.erb*.

6 Conclusions

The paper has presented the design and implementation of a modular integrated metrics analyser for the SonarQube framework. Its execution inside the SonarQube framework results in the integration and connection of both tools that improves their capabilities, yielding a single colloborative remote working space that supports the interaction of verification teams working over specific projects.

The module is applicable to all projects with the need of technical metrics collection where external tools are mandatory to collect some metrics not provided initially by the SonarQube platform, used as quality management plattform; and it has been tried in a real project that requires compliance with norms related to the development of critical software such as DO-178C [46] and software quality such as AQAP-2210 [2].

Acknowledgment This work has been partly supported by the Spanish national project REM4VSS (TIN 2011-28339) and the Technology and Product Management department of Indra (Spain) under contract no. 2004/00476/001.

References

1. Alonso, A., García-Valls, M., de la Puente, J.A.: Assessment of timing properties of family products. In: ARES Workshop – Development and Evolution of Software Architectures for Product Families. LNCS, vol. 1429, pp. 161–169. Springer (1998)
2. AQAP 2210. NATO Supplementary Software Quality Assurance Requirements to AQAP 2110, 1st edn., November 2006

3. AQAP 2110. NATO Quality Assurance Requirements for Design, Development and Production, 2nd edn., November 2006
4. Balachandran, V.: Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In: Proc. of International Conference on Software Engineering (ICSE) (2013)
5. Bersani, M.M., García-Valls, M.: The cost of formal verification in adaptive CPS. an example of a virtualized server node. In: Proc. of 17th IEEE High Assurance Systems Engineering Symposium (HASE), January 2016
6. Bouyssounouse, B., et al.: Programming languages and real-time systems. In: Embedded Systems Design: The ARTIST Roadmap for Research and Development. Springer (2005)
7. Bouyssounouse, B., et al.: QoS management. In: Embedded Systems Design: The ARTIST Roadmap for Research and Development. Springer (2005)
8. Bouyssounouse, B., et al.: Adaptive real-time systems development. In: Embedded Systems Design: The ARTIST Roadmap for Research and Development. Springer (2005)
9. CMMI Product Team. CMMI for Development, version 1.3. Improving processes for developing better products and services. CMU/SEI-2010-TR-033 (2010)
10. Campbell, G.A., Papapetrou, P.P.: SonarQube in Action. Manning Publications (2013). ISBN-9781617290954
11. Cano Romero, J., García-Valls, M.: Scheduling component replacement for timely execution in dynamic systems. Software: Practice and Experience **44**(8), 889–910 (2014)
12. CENELEC. Railway applications - Communications, signalling and processing systems. CENELEC (2001)
13. Coleman, D., Ash, D., Lowther, B., Oman, P.: Using metrics to evaluate software system maintainability. IEEE Computer **27**(8), 44–49 (2002)
14. di Ruscio, D., Pelliccione, P.: A model-driven approach to detect faults in FOSS systems. Journal of Software: Evolution and Process **27**(4), April 2015
15. Duenas, J., Alonso, A., Lopes Oliveira, W., Garcia, M., Leon, G.: Software architecture assessment. In: Software Architecture for Product Families: Principles and Practice. Addison-Wesley (2000)
16. Duvall, P.M., Matyas, S., Glover, A.: Continuous integration: improving software quality and reducing risk. Pearson Education (2007)
17. Fenton, N., Bieman, J.: Software metrics: a rigorous and practical approach. CRC Press (2014)
18. García Valls, M., Cucinotta, T., Lu, C.: Challenges in real-time virtualization and predictable cloud computing. Journal of Systems Architecture **60**(9), 736–740 (2014)
19. García Valls, M., Baldoni, R.: Adaptive middleware design for CPS: considerations on the OS, resource managers, and the network run-time. In: Proc. 14th Workshop on Adaptive and Reflective Middleware (ARM) (2015)
20. García-Valls, M., Fernández Villar, L., Rodríguez López, I.: iLAND: An enhanced middleware for real-time reconfiguration of service oriented distributed real-time systems. IEEE Transactions on Industrial Informatics **9**(1), February 2013
21. García-Valls, M., Alonso, A., de la Puente, J.A.: A Dual-Band Priority Assignment Algorithm for QoS Resource Management. Future Generation Computer Systems **28**(6), 902–912 (2012)
22. García-Valls, M., Alonso, A., de la Puente, J.A.: Mode change protocols for predictable contract-based resource management in embedded multimedia systems. In: Proc. of IEEE Int'l Conference on Embedded Software and Systems (ICESS), May 2009
23. García-Valls, M., Alonso Munoz, A., Ruíz, J., Groba, A.: An architecture of a quality of service resource manager middleware for flexible multimedia embedded systems. In: Proc. of 3rd Intern'l Workshop on Software Engineering and Middleware. LNCS, vol. 2596 (2003)
24. García-Valls, M., Perez-Palacin, D., Mirandola, R.: Time sensitive adaptation in CPS through run-time configuration generation and verification. In: Proc. of 38th IEEE Annual Computer Software and Applications Conference (COMPSAC), pp. 332–337, July 2014
25. García-Valls, M., Basanta-Val, P., Estévez-Ayres, I.: Real-time reconfiguration in multimedia embedded systems. IEEE Transactions on Embedded Consumer Electronics **57**(3), 1280–1287 (2011)

26. García-Valls, M., Basanta-Val, P.: A real-time perspective of service composition: key concepts and some contributions. Journal of Systems Architecture **59**(10), 1414–1423 (2013)
27. García-Valls, M., Basanta-Val, P.: Comparative analysis of two different middleware approaches for reconfiguration of distributed real-time systems. Journal of Systems Architecture **60**(2), 221–233 (2014)
28. García-Valls, M., Uriol-Resuela, P., Ibánez-Vázquez, F., Basanta-Val, P.: Low complexity reconfiguration for real-time data-intensive service-oriented applications. Future Generation Computer Systems **37**, 191–200 (2014)
29. García-Valls, M., Basanta-Val, P.: Usage of DDS Data-Centric Middleware for Remote Monitoring and Control Laboratories. IEEE Transactions on Industrial Informatics **9**(1), 567–574 (2013)
30. García-Valls, M., Basanta-Val, P., Estévez-Ayres, I.: Adaptive real-time video transmission over DDS. In: Proc. of 8th IEEE International Conference on Industrial Informatics (INDIN), July 2010
31. IEC 61508. Functional safety of electrical/electronic/programmable electronic safety-related systems, April 2010
32. IEC. Nuclear power plants. Instrumentation and control important to safety. General requirements for systems. IEC 61513 Ed.2.0., August 25, 2011
33. IEC. Medical Device Software–IEC, May 2006
34. ISO. Road Vehicles - Functional Safety. ISO-26262, November 11, 2011
35. Jenkins. Information. http://jenkins-ci.org/ (last retrieved, February 19, 2015)
36. Krutchen, P.: Contextualizing agile software development. Journal of Software: Evolution and Process **25**, 351–361 (2013)
37. LDRA. Information. http://www.ldra.com/ (last retrieved, February 19, 2015)
38. Maven. Information. http://maven.apache.org/ (last retrieved, February 19, 2015)
39. PECAL-2210. Requisitos OTAN de aseguramiento de la Calidad del software, suplementarios a la PECAL 2110, 1st edn., November 2007
40. PECAL-2110. Requisitos OTAN de aseguramiento de la Calidad para el diseño, el desarrollo y la producción, 2nd edn., November 2006
41. PC-Lint. Information. http://www.gimpel.com/html/index.htm (last retrieved, February 19, 2015)
42. PMD. Information. http://pmd.sourceforge.net/ (last retrieved, February 19, 2015)
43. SonarQube. Information. http://www.sonarqube.org/ (last retrieved, May 04, 2015)
44. Splint. Information. http://www.splint.org/ (last retrieved, February 19, 2015)
45. Otero Pérez, C.M., Steffens, L., van der Stok, P., van Loo, S., Alonso, A., Ruíz, J., Bril, R.J., García Valls, M.: QoS-based resource management for ambient intelligence. In: Ambient Intelligence: Impact on Embedded Sytem Design, pp. 159–182. Kluwer Academic Publishers (2003)
46. RTCA Inc. Software Considerations in Airborne Systems and Equipment Certification. RTCA Inc. DO-178C, December 13, 2011
47. RTCA Inc. / EUROCAE. Software Integrity Assurance Considerations for Communication, Navigation, Surveillance and Air Traffic Management (CNS/ATM) Systems. DO-278A, December 13, 2011
48. RTCA Inc. DO-178B. Software Considerations in Airborne Systems and Equipment Certification. RTCA Inc. DO-178B (1992)
49. RTCA Inc. Software Tool Qualification Considerations. DO-330, December 13, 2011
50. RTCA Inc. Model-Based Development and Verification Supplement to DO-178C and DO-278A. DO-331, December 13, 2011
51. RTCA Inc. Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A. DO-332, December 13, 2011
52. RTCA Inc. Formal Methods Supplement to DO-178C and DO-278A
53. Scitools. Scitools Understand. Information. https://scitools.com/ (last retrieved, February 19, 2015)