Chapter 3
White-Box Testing Automation With SonarQube: Continuous Integration, Code Review, Security, and Vendor Branches

Miguel Jorge Andrade
Polytechnic Institute of Porto, Portugal

ABSTRACT

Modern work patterns like continuous integration (CI) have an implicit need for testing automation. In current CI solutions, white-box testing is left to the work methodology, typically addressed after code reviews. Code security inspection is often done in specific code reviews focusing on security. SonarQube is a tool that, to a certain extent, can automate white-box design and testing and serve as a guide for formal code reviews. Moreover, this tool can help audit the code for potential security issues. Most web programming today uses components readily available and transparently managed by package managers, like npm for Node.js or Composer for PHP. This use must also be audited at least for potential security problems; yet traditional white-box test design would require a good understanding of the vendor code, which can be difficult/impractical to achieve. This chapter will address SonarQube as a valuable tool to automate white-box and security testing and also provide suggestions on how to manage your vendor branches when there is a need to audit/change the vendor source code.

DOI: 10.4018/978-1-5225-7455-2.ch003
INTRODUCTION

Testing code has been one of the most recurring practices to assure software quality and is a fundamental pillar of any software development methodology. Today this can be regarded as a widespread and adopted practice (Kochhar, Bissyande, Lo, Jiang 2013). It is followed in about 50% of Open Source projects available in GitHub and is part of any syllabus on software engineering. Testing is mostly addressed by unit testing, and other Quality Assurance (QA) methodologies are usually delegated to the software development methodology. One of the most recognized and effective (McConnell, 2016) QA methods is code reviews, or “peer code reviews”. Code reviews take time, and even if it’s recognized that they will save time in the project lifetime, they are often postponed or even ignored in small projects.

Historically, white-box testing or, to a certain extent, static code analyses, require a team to be available at some point in time, to execute the code review task, make corrections and design new tests, based on the code knowledge and project propose (Ostrand, 2002). On a Continuous Integration (CI) project and specifically in a Continuous Delivery project this action should be done more often, but this would be frequently infeasible, either to have a team reviewing each commit, or even if just reviewing before any production update.

This is where automated static code analysis is valuable (Zampetti, Scalabrino, Oliveto, Canfora, Penta, 2017).

There are several tools available that can help developers, language specific tools, multi-language tools and also on-line services. This chapter will address one specific tool, SonarQube. Other tools could be used to obtain the same results: hopefully the insights gained on analyzing a specific tool can also be helpful on addressing the same workflows with other tools, or assist in designing custom tools.

SonarQube provides preconfigured rules that check committed code to look for “code smells” and other patterns like potential bugs or security issues. These are just to detect potential problems, they do not assure a real problem exist, yet these rules can be combined to “score” a potential problematic commit, and force the build to fail. These rules can be independently written, but often they are sourced from a rules repository. These repositories are categorized by programming language and function. As an example, a project manager can enforce a certain indentation or code style for a project. In SonarQube These rules have associated an expected problem resolution time, which can be added to achieve a “technical debt” measure for the project. These analyses can be automated and triggered by commits in a CI environment. Different sets of rules, called “Quality Profiles”, can be applied depending on the nature of the project. A second SonarQube concept is the “Quality Gate”. This is an aggregation of several computed measures, like the number of
Tools for the Learning of Programming Languages and Paradigms: Integration of a Code Validator and Exercises Module Into the Moodle eLearning Platform
www.igi-global.com/chapter/tools-for-the-learning-of-programming-languages-and-paradigms/219979?camid=4v1a

Dynamic Generation of Documentation, Code, and Tests for a Digital Marketing Platform's API
www.igi-global.com/chapter/dynamic-generation-of-documentation-code-and-tests-for-a-digital-marketing-platforms-api/219974?camid=4v1a