# Preface

## INTRODUCTION

Programming is around for some decades. Nevertheless, quality control is not yet a reality. There are different approaches to this problem: code IDEs are able to detect some code smells, and suggest editions to the programmer; code generation, from abstract specifications, can allow the quicker development of code and, if correctly abstracting the generated code, would allow easier implementation, and better generated code quality; code documentation both allows programmers to understand the behavior of the code they use, as it allows other programmers to edit existing code with more confidence, but there is no such thing as good documentation. Tools to allow the generation of documentation allow to assess their completeness, but their quality is still a problem; While code auditing tools are able to analyze code flows, and detect common bad coding practices and code standards try to force programmers to follow a specific set of rules, in order to reduce these bad programming practices, few tools exist that really deal with automatic refactoring of this code; finally, Unit Testing and Feature Testing allows programmers to guarantee that a desired behavior for some code is kept intact during software development, but how to evaluate the coverage of these tests on all possible corner cases is still a main challenge. These are all aspects to be covered on this book, sharing what is being done in the actuality to reduce all these problems.

## THE CHALLENGES

In recent years we have witnessed a gigantic process of automating all our day-to-day tasks whether domestic or professional. This complex process requires expert people, machines, and frameworks to support all the code architecture behind this automation. In this sense, coding, once seen as another phase of the software development lifecycle, is now seen as the crucial phase for the quality of the automation process. Being an important and complex phase, it is necessary to provide the programmers

with techniques that allow to generate, test and evaluate the code in a simple and practical way. However often these techniques are difficult to apply due to many variables: much disorganized code made by several people with different styles, thus making refactoring difficult and requiring more robust tests, increasing and advanced level of malicious code, among other threats.

## DESCRIPTION AND ORGANIZATION OF THE BOOK

This book presents a comprehensive and recent view of the emerging trends, techniques, and tools for code generation, analysis tools, and testing for quality. At the same time, it identifies new trends on this topic from pedagogical strategies to technological approaches. The book has nine chapters organized in three sections. A brief description of each of the sections follows:

Section 1 describes techniques for code refactoring and tests. These novel approaches aim to increase code quality through code generation at project startup thus eliminating delayed and error prone boilerplate code creation. At the same time the refactoring allows developers to improve the internal structure of the code without changing its external behavior. At the same time, it improves code understanding, which facilitates maintenance and avoids the inclusion of defects.

Chapter 1 describes a digital marketing platform API with more than three hundred thousand clients in more than fifty countries. Due to the swift growth of the platform's functionality, its web API documentation and client code libraries became quickly outdated and with unexpected bugs and failures, since there is lack of software tests. This was the main reason for authors to propose a solution capable of successfully generating acceptance tests for their public API based on its specification.

Chapter 2 describes a procedure which ensures that all the covered lines of test suit are given by statement coverage analysis and branches using branch coverage analysis and generate a coverage report. All the specified test cases described in the testing section are tested properly and get appropriate outcomes for statement and branch coverage. The structural coverage analysis provides a sensible approach that balances the DO-178B requirements for structural coverage.

Chapter 3 details how to use SonarQube in an automated manner, more precisely, authors address analysis on security, and how to deal with code from vendor branches. SonarQube is an Open Source quality management platform, dedicated to continuously analyze and measure technical quality, from project portfolio to method. It incorporates a plugin system for extension purposes.

Section 2 enumerates several approaches to foster the teaching-learning process in the domain of computer programming. Since we are dealing with code learning,

several techniques are shared to improve learning engagement and integration in learning management systems.

Chapter 4 distinguishes MOOCs (Massive Open Online Courses) and Online Coding Bootcamps as two increasingly popular options for learners to improve their code development skills and find work within a relatively short amount of time. Among all the features available on these environments, one stands out, which is the code generation. This paper aims to detail and compare the most popular solutions for both learning contexts based on several criteria such as impact and maturity, user groups and tools and features.

Chapter 5 presents two educational tools focused on the client-side scripting language JavaScript to support the web development applications learning process for engineering students. The two tools are a Moodle-based JavaScript Code Validator and an exercises module. The experiences were carried out during an academic course with quite positive results.

Chapter 6 presents a three-year experience in teaching MDE in a course of a master program in Informatics Engineering. The chapter provides details on how a project-based learning approach was adopted and evolved along three editions of the course. Results of a student survey are discussed and compared to those from another course. In addition, several other similar teaching experiences are analyzed.

Section 3 promotes the use of Domain Specific Languages (DSL). The DSL are computer languages specialized to a particular application domain. In this case, DSLs are used to as a way to foster code generation and refactoring.

Chapter 7 presents new metrics for refactoring index to prioritize the code smell and find the software that needs maximum refactoring. In this study, refactoring index is calculated for an entire project whereas entropy method can be used for component based system where code smells can be detected from an individual components and components are prioritize according to refactoring need.

Chapter 8 presents a novel technique for the development of embedded DSL through the use of operator overloading. While operator overloading is a common functionality on recent object-oriented languages, the way these operators are used is, in most situations, the simple replacement of the default operator behavior.

Chapter 9 describes the OpenAPI specification (OAS) and presents a custom metamodel, from which a domain specific modeling language was developed. This model is a simplification of the main concepts that integrate the OAS, giving a high-level interpretation of the relationships between them, while supporting the DSL development. This metamodel answers one of the three questions raised by the authors, on what were the main compromises in the specification of a language agnostic model to represent the OAS, since it deliberately leaves out some OAS concepts.

## CONCLUSION

This book aims to share new approaches and methodologies for code generation, edition, analysis and testing. At the same time, it identifies new trends on these topics, from pedagogical strategies to technological approaches.

The proposed book could be used as a valuable resource for practitioners and as a reference for research scholars, computer science teachers and students pursuing computer science related subjects and enterprise developers.

*Alberto Simões*
*Polytechnic Institute of Cávado and Ave, Portugal*

*Mário Teixeira Pinto*
*Polytechnic Institute of Porto, Portugal*

*Ricardo Alexandre Peixoto de Queirós*
*Polytechnic Institute of Porto, Portugal*