Chapter 7.4
Teaching Agile Software Development Quality Assurance

Orit Hazzan
Technion–Israel Institute of Technology, Israel

Yael Dubinsky
IBM Haifa Research Lab, Israel
Technion–Israel Institute of Technology, Israel

ABSTRACT

This chapter presents a teaching framework for agile quality—that is, the way quality issues are perceived in agile software development environments. The teaching framework consists of nine principles, the actual implementation of which is varied and should be adjusted for different specific teaching environments. This chapter outlines the principles and addresses their contribution to learners' understanding of agile quality. In addition, we highlight some of the differences between agile software development and plan-driven software development in general, and with respect to software quality in particular. This chapter provides a framework to be used by software engineering instructors who wish to base students learning on students' experiences of the different aspects involved in software development environments.

INTRODUCTION

Quality assurance (QA) is an integral and essential ingredient of any engineering process. Though there is a consensus among software practitioners about its importance, in traditional software development environments conflicts may still arise between software QA people and developers (Van Vliet, 2000, p. 125).

Agile software development methods emerged during the past decade as a response to the characteristics problems of software development processes. Since the agile methods introduced a different perspective on QA, we will call the agile approach toward quality issues *agile quality*—AQ, and will focus, in this chapter, on the teaching of AQ. By the term AQ, we refer to all the activities (e.g., testing, refactoring, requirement gathering) that deal with quality as they are manifested and applied in agile software development environments. It is important to emphasize that the term

AQ does not imply that quality changes. To the contrary, the term AQ reflects the high standards that agile software methods set with respect to software quality.

Based on our extensive experience of teaching agile software development methods both in academia and in the software industry[1], we present a teaching framework for AQ. The teaching framework consists of nine principles, the actual implementation of which is varied and should be adjusted for different specific teaching environments (e.g., academia and industry to different sizes of groups). This chapter outlines the principles and addresses their contribution to learners' understanding of AQ.

In the next section, we highlight some of the differences between agile software development and plan-driven[2] software development in general, and with respect to software quality in particular. Then, we focus on the teaching of AQ. We start by explaining why quality should be taught and, based on this understanding, we present the teaching framework for AQ, which suggests an alternative approach for the teaching of AQ. Finally, we conclude.

Agile vs. Plan-Driven Software Development

In this section, we highlight some of the main differences between agile software development and traditional, plan-driven software development. Before we elaborate on these differences, we present our perspective within which we wish to analyze these differences.

Traditional software development processes mimic traditional industries by employing some kind of production chain. However, the failure of software projects teaches us that such models do not always work well for software development processes. In order to cope with problems that result from such practices, the notion of a production chain is eliminated in agile software development environments and is replaced by a more network-oriented development process (Beck, 2000). In practice, this means that in agile teams, the task at hand is *not* divided and allocated to several different teams according to their functional description (for example, designers, developers, and testers), each of which executes its part of the task. Rather, all software development activities are intertwined and there is no passing on of responsibility to the next stage in the production chain. Thus, all team members are equally responsible for the software quality. We suggest that this different concept of the development process results, among other factors, from the fact that software is an intangible product, and therefore it requires a different development process, as well as a different approach toward the concept of software quality, than do tangible products.

Agile Development Methods vs. Plan-Driven Development Methods

During the 1990s, the agile approach toward software development started emerging in response to the typical problems of the software industry. The approach is composed of several methods and it formalizes software development frameworks that aim to systematically overcome characteristic problems of software projects (Highsmith, 2002). Generally speaking, the agile approach reflects the notion that software development environments should support communication and information sharing, in addition to heavy testing, short releases, customer satisfaction, and sustainable work-pace for all individuals involved in the process. Table 1 presents the manifesto for agile software development (http://agilemanifesto.org/).

Several differences exist between agile software development methods and plan-driven methods. Table 2 summarizes some of these differences.