Chapter 53
Software Testing Under Agile, Scrum, and DevOps

Kamalendu Pal
https://orcid.org/0000-0001-7158-6481
City, University of London, UK

Bill Karakostas
Independent Researcher, UK

ABSTRACT

The adoption of agility at a large scale often requires the integration of agile and non-agile development practices into hybrid software development and delivery environment. This chapter addresses software testing related issues for Agile software application development. Currently, the umbrella of Agile methodologies (e.g. Scrum, Extreme Programming, Development and Operations – i.e., DevOps) have become the preferred tools for modern software development. These methodologies emphasize iterative and incremental development, where both the requirements and solutions evolve through the collaboration between cross-functional teams. The success of such practices relies on the quality result of each stage of development, obtained through rigorous testing. This chapter introduces the principles of software testing within the context of Scrum/DevOps based software development lifecycle.

INTRODUCTION

The world is witnessing a tremendous influence of software systems in all aspects of personal and business areas. Software systems are also heavily incorporated in safety-critical applications including manufacturing machinery, automobiles operation, and industrial process controls. In these applications, software failure can cause injury or loss of life. The correct behaviour of software is crucial to the safety and wellbeing of people and business. Consequently, there is an increasing requirement for the application of strict engineering discipline to the development of software systems.

DOI: 10.4018/978-1-6684-3702-5.ch053

The human being, however, is fallible. Even if they adopt the most sophisticated and thoughtful design techniques, erroneous results can never be avoided *a priori*. Consequently, software products, like the products of any engineering activity, must be verified against its requirements throughout its development.

One fundamental approach to verification is experimenting with the behaviour of a product to see whether the product performs as expected. It is common practice to input a few sample cases *(test cases*), which are usually randomly generated or empirically selected, and then verify that the output is correct. However, it cannot provide enough evidence that the desired behaviour will be exhibited in all remaining cases. The only testing of any system that can provide absolute certainty about the correctness of system behaviour is exhaustive testing, i.e., testing the system under all possible circumstances.

In addition, new improved methods and tools for software development are the goals of researchers and practitioners. The procedure of software development has evolved over the decades to accommodate changes in software development practice. Many methods and modelling techniques have been proposed to improve software development productivity. Also, software engineering has gone through an evolution in its conception by the business world in the 1960s to recent day application development methodologies (Pal, 2019).

Software practitioners employ software development methodologies for producing high-quality software, satisfying user requirements, effectively managing the software development cost, and ensuring timely delivery. In this way, software development methodologies play an important role to provide a systematic and organized approach to software development (Sommerville, 2019). According to Kevin Roebuck (Roebuck, 2012), a traditional Software Development Life Cycle (SDLC) provides the framework for planning and controlling the development or modification of software products, along with the methodologies and process models are used for software development.

According to researchers such as (Beck et al., 2001), the Waterfall Modell (Royce, 1970) was proposed to the information processing industry, as a way in which to assess and build for the users' needs. It starts with an end-user requirements analysis that produces all required input for the next stage (software system design), where software engineers collaborate with others (e.g. database schema designers, user interface designers) to produce the optimal information system architecture. Next, coders implement the system with the help of specification documents, and finally, the deployed software system is tested and shipped to its customers (Beck, 1999).

This process model (work practice) although effective from a theoretical perspective, did not always work as expected in real life scenarios. Firstly, software customers often change their minds. After weeks, or even months, of gathering requirements and creating prototypes, users can still be unsure of what they want – all they know is that what they saw in the produced software was *not* quite "it". Secondly, requirements tend to change mid-development, however, it is difficult to stop the momentum of the project to accommodate the change. The traditional process models (e.g. Waterfall, Spiral) start to pose problems when change rates of requirements are relatively high (Boehm, 2002) because coders, system architects, and managers need to introduce and keep up to date a huge amount of documentation for the proposed system, even for small changes (Boehm, 1988). The Waterfall software process model was supposed to fix the problem of changing requirements once and for all by freezing requirements and not permitting any change once the project starts. However, it is a common experience that software requirements cannot be pinned down in one fell swoop (Beck, 1999).

In recent decades, the software industry has moved its production mechanism from traditional software development practice to agile methodologies, to mitigate the ever-increasing software complexity and globalization of software design and development business. Many industries have started to adopt new