Chapter 4
Adapting Test-Driven Development to Build Robust Web Services

Nuno Laranjeiro
Universidade de Coimbra, Portugal

Marco Vieira
Universidade de Coimbra, Portugal

ABSTRACT

Web services are increasingly being used in business critical environments as a mean to provide a service or integrate distinct software services. Research indicates that, in many cases, services are deployed with robustness issues (i.e., displaying unexpected behaviors when in presence of invalid input conditions). Recently, Test-Driven Development (TDD) emerged as software development technique based on test cases that are defined before development, as a way to validate functionalities. However, programmers typically disregard the verification of limit conditions, such as the ones targeted by robustness testing. Moreover, in TDD, tests are created before developing the functionality, conflicting with the typical robustness testing approach. This chapter discusses the integration of robustness testing in TDD for improving the robustness of web services during development. The authors requested three programmers to create a set of services based on open-source code and to implement different versions of the services specified by TPC-App, using both TDD and the approach presented in this chapter. Results indicate that TDD with robustness testing is an effective way to create more robust services.

INTRODUCTION

Web services are increasingly being used in Service Oriented Environments as a strategic vehicle for data exchange and software component interoperability, providing a simple interface between a service provider and a consumer. Interaction between service consumers and providers is achieved by exchanging messages that comply with the SOAP protocol, which, along with WSDL and UDDI, constitute the core of the web services technology (Curbera et al., 2002).

Web services are frequently complex software components that can implement a composite ser-

DOI: 10.4018/978-1-4666-4301-7.ch004

vice, in some cases using a set of external web services. Software faults (i.e., program defects or bugs) (Kalyanakrishnam, Kalbarczyk, & Iyer, 1999; Lee & Iyer, 1995) are a relevant cause of computer failures and, research indicates that web services are not different from other types of software, in this matter (Vieira, Laranjeiro, & Madeira, 2007a). With the increase of the software complexity, the weight of software faults also tends to increase.

Interface faults are related to problems in the interaction among software components or modules (Weyuker, 1998) and are of utmost importance in web services environments. Web services must provide a robust interface to client applications even when clients misuse the service by providing invalid input calls. Such invalid inputs may result from bugs in the client applications, data corruption caused by silent network failures, or even security attacks. Obviously, in web services compositions (a set of web services that work together to achieve a goal), when a component fails (by, for instance, throwing an unexpected exception), the entire composition may be affected. In fact, the execution results of subcomponents (i.e., external services) can be seen as inputs for the main service and are, in fact, a potential source of robustness issues. Additionally, a particular web service composition may use services provided by external entities, which emphasizes the importance of mitigating unexpected inputs to improve the robustness of the overall composition.

Creating robust web services is a challenging task. In fact, research and practice show that many web services are being deployed on the web with robustness problems (Vieira, Laranjeiro, & Madeira, 2007a), i.e., displaying unforeseen behaviors when handling invalid inputs. Among other effects, these robustness issues can result in security vulnerabilities due to the lack (or incorrect use) of input validation. A frequently observed case is the presence of SQL Injection vulnerabilities, where unchecked inputs are exploited by hackers with the goal of modifying the structure of a SQL command (Stuttard & Pinto, 2007).

Test-Driven Development (TDD) (Beck, 2003) is an agile software development technique based on test cases that define new software functionalities or improvements (i.e., unit tests specify the requirements and are created before writing the functionality code itself). Development then follows in short iterations, where the developer creates the code that is required for the tests to pass. The process explicitly incorporates changes (via refactoring) as a means to improve code quality. Despite this, the definition of test cases that assure high coverage is quite demanding and developers tend to focus on the creation of tests that satisfy the requirements in normal situations, while often disregarding the verification of limit condition, such as the ones targeted by robustness testing.

Robustness testing can characterize the behavior of a particular system in presence of invalid input conditions (Mukherjee & Siewiorek, 1997). Web services robustness testing is an after-development technique that has its origin in traditional robustness testing approaches (Koopman & DeVale, 1999; Rodríguez, Salles, Fabre, & Arlat, 1999), typically used to assess of robustness of operating systems and microkernels. The fact that this testing technique was designed to be executed after development conflicts with the Test-Driven Development approach, which requires the tests to be created before developing the software functionalities.

In previous work we presented a preliminary approach that illustrates how to use robustness testing in a Test-Driven Development environment (Laranjeiro & Vieira, 2009) and focuses on basic required adaptations on both techniques. In this paper we present a thorough methodological view on how Test-Driven Development can be extended to include robustness testing. The approach is discussed in detail from a software development process point-of-view, while still including the theoretical and technical aspects