Chapter 8.7
Enhancing the Testability of Web Services

Daniel Brenner
University of Mannheim, Germany

Barbara Paech
University of Heidelberg, Germany

Matthias Merdes
Heidelberg Mobil International GmbH, Germany

Rainer Malaka
University of Bremen, Germany

ABSTRACT

For the foreseeable future, testing will remain the mainstay of software quality assurance and measurement in all areas of software development, including Web services and service-oriented systems. In general, however, testing Web services is much more challenging than testing normal software applications, not because they are inherently more complex, but because of the limited control and access that users of Web services have over their development and deployment. Whereas the developers of normal applications, by definition, have full control over their application until release time, and thus, can subject them to all kinds of tests in various combinations (e.g., integration testing, system testing, regression testing, acceptance testing, etc.), users of Web services can often only test them at run-time after they have already been deployed and put into service. Moreover, users of Web services often have to share access to them with other concurrent users. In order to effectively test Web services under these conditions special measures and approaches need to be taken to enhance their testability. Right from the early phases of development, the testability of services needs to be taken into account and "designed into" services. In this chapter we consider these issues and with the aid of a case study we present a methodology that can be used to enhance the testability of Web services.

DOI: 10.4018/978-1-60566-042-4.ch010

INTRODUCTION

Service-oriented development is based on the idea of building new software applications using software "services," often built and deployed by third party organizations. It therefore assumes a fundamental separation of concerns between *service developers*, who create and offer services with no knowledge of specific applications to which they may be put, and *service users*, who assemble new applications which use the services available on the Internet or in a company's Intranet.

Since they are themselves software applications, services are typically developed and tested using the same basic practices and techniques used to develop normal software applications, and as a result, they can be expected to exhibit the same levels and variations in quality found in the general population of software applications. However, in service-oriented development, the quality of a service-based application is not just based on the inherent quality of the services it uses, it is also dependent on whether they are used or assembled in the correct way. A system assembled from perfectly correct services may still function incorrectly if it uses the services in a different way to that intended; in other words, if the system's understanding of its contract with a service is different to the service provider's.

This problem exists in all component-based approaches whenever a new application is created from prefabricated parts. However, the situation is more acute in service-based development because service users have much less access to, and control over, their components than in regular component-oriented development approaches using such technologies as EJB, SPRING, or .NET. If the developers are creating all of their own components in-house, they can test larger assemblies of components while the development process is underway (integration testing). And even when some of the components are purchased from a third party, once the first version of the system has been completed they can still often test the full system under controlled conditions in the development/test environment (system testing). Finally, once a traditional component-based system has been delivered to the customer, it can be further tested in the customer's target environment to determine whether it fulfils the customer's needs (acceptance testing) before being made accessible to end users.

In general, none of these traditional testing activities can be carried out in the normal way when developing a service-oriented system, however. First, the final collection of "components" that make up an application is typically not known until deployment-time, making full integration and system testing in the traditional form impossible. Second, many of the "components" may already provide services to other users, and cannot be shut down even temporarily to participate in traditional testing activities in a controlled environment. A new application will often have to share its components with other applications and cannot assume that these will cooperate while it is in testing mode.

Although subtle, these differences have a fundamental impact on the role and goals of testing in the system development process. Because services are in effect independent, multiuser systems in their own right, new applications have to test them at run-time in a way that combines acceptance testing with integration and system testing. In other words, tests of "components" by applications can no longer be regarded as a pure verification exercise, as has traditionally been the case, but must also include an element of "validation" (Boehm, 1984). An application that is connected to a new service at run-time needs to determine whether the service does the "right thing," just as much as it needs to determine whether it does that thing "right."

Building services so that they can be tested in this way requires changes to the way they are traditionally designed and the way that tests are carried out. The purpose of this chapter is to discuss these changes and to present a methodology