Chapter 2
Semi-Automated Lifecycles for Eliciting Requirements for Service-Oriented Environments

M. Brian Blake
University of Notre Dame, USA

ABSTRACT

Service-based tools are beginning to mature, but there is a cognitive gap between the understanding of what currently exists within an organization and how to use that knowledge in planning an overall enterprise modernization effort that realizes a service-oriented architecture. Traditional and contemporary software engineering lifecycles use incremental approaches to extract business information from stakeholders in developing features and constraints in a future application. In traditional environments, this information is captured as requirements specifications, use cases, or storyboards. Here, we address the evolution of traditional software engineering approaches to support the conceptualization of abstract services that overlap multiple organizations. Traditional software engineering lifecycles must be enhanced with emerging processes related to the development applications for service-oriented environments. The chapter discusses state-of-the-art approaches that elicit information about the requirements for service-oriented architectures. These approaches tend to leverage existing requirements engineering approaches to suggest aggregate service-based capabilities that might be most effective for a particular environment.

DOI: 10.4018/978-1-61350-159-7.ch002

INTRODUCTION: THE BACKGROUND OF SERVICE LIFECYCLES THAT SUPPORT NEW REQUIREMENTS ENGINEERING PRACTICES

Since the term software engineering was coined in the 1960s, one of the primary thrusts has been toward software modularity. A goal of software modularity is the development of packaged software mechanisms that perform concise domain-specific tasks. Such an application can be clearly understood by adjacent stakeholders and seamlessly integrated and consumed into new systems. A significant thrust towards software modularity was the introduction of object-oriented analysis and development. *Object-oriented programming* was first introduced in the early 1960's as a part of the development of SIMULA 67 (Nygard, 1986). Interestingly enough, the impact of object-oriented systems was not fully realized until the mid 1980's with the establishment and acceptance of methodologies introduced by Ivar Jacobsen, Grady Booch, and James Rumbaugh. This represented a gap of approximately 20 years before object-oriented programming experienced large-scale acceptance in application. Similarly, while *component-based programming* (i.e. a higher-level abstraction of object-oriented design) was introduced in late 1960's as a part of the NATO conference on the software *crisis* (Naur & Randell, 1969), it was not until the mid 1990's that large-scale component –based applications became apparent.

The combination of components and the World-Wide Web has led to the emergence of Web services: components with URIs that can be invoked according to standard protocols. Web services underlie the general notion of a *System of Systems* where the underlying computations and interactions follow the principles of *service-oriented computing* (Bichler & Lin, 2006)(Rao & Su, 2004). Unlike the above two approaches for building systems, service-oriented computing has experienced a relatively short horizon from conceptualization to application. While Gartner has been credited for the first mention of service-oriented architecture in 1996 (SSA, 1996), just 5 years later research labs, industrial organizations, and federal government facilities have all adopted a paradigm shift to create these types of architectures within their domains (see Figure 1).

Service-oriented architecture (SOA) is made possible by the existence of services both inside and outside of an enterprise. Within such an environment, independent tasks can be referred to as organizational *capabilities*. *Web services* provide the implementations of the capabilities and represent the building blocks. Properly-developed web services should execute well-defined tasks as supported by concise, openly accessible interfaces. As such, it is apparent that the first step in the development of any SOA is the acceptance of a service-based development approach (Blake, 2007) and the population of an array of services that represent capabilities across a wide variety of domains.

Figure 1. The evolution of software modularity