Chapter V

Evolving Legacy System Features into Fine-Grained Components

Alok Mehta, American Financial Systems, Inc., USA
George T. Heineman, WPI Computer Science Department, USA

ABSTRACT

There is a constant need for practical and cost-effective software evolution techniques. We have developed a novel evolution methodology that integrates the concepts of features, regression testing, and component-based software engineering (CBSE). Regression test cases are untapped resources, full of information about system features. By exercising each feature with its associated test cases using code profilers, code can be located and refactored to create components. These components are then inserted back into the legacy system, ensuring a working system structure. This methodology is divided into three parts. Part One identifies the source code associated with features that need evolution. Part Two deals with creating components and Part Three measures results. We have validated our approach on the evolution of a real-world legacy system. By applying this methodology, American Financial Systems, Inc. (AFS) has successfully restructured its enterprise legacy system and reduced the costs of future maintenance. In this chapter we show a simple example of this methodology in action.
INTRODUCTION

Execom (2003) estimates that legacy software systems capture and manage 75% of the world’s data and that they consume at least 80% of available Information Technology resources. These legacy systems provide a competitive advantage to many organizations but are expensive to maintain. The constant pressure to evolve these systems is driven by escalating expectations of the customer for new enterprise standards, new products and system features, and improved performance. Thus, these organizations face a dilemma — they cannot afford to lose their competitive advantage by rewriting or redesigning the software system, nor can they continue to accept the high maintenance costs. To effectively evolve legacy systems in such a rapidly changing environment, organizations must answer two questions (Smith, Müller & Tilley, 1997): What are the critical success factors of system evolution? How can a system be evolved without adversely affecting operations and revenue? These pressing business concerns motivate this chapter.

Incremental Evolution of Legacy Systems

Of the many strategies for managing legacy software systems, such as doing nothing, rewriting, or replacing with other systems (Bergey, Northrop & Smith, 1997; Brodie & Stonebraker, 1995; Weideman et al., 1997; Tilley & Smith, 1996), we recommend incremental evolution. Incremental evolution focuses on problems that are most visible to end-users. Rather than replacing or rewriting the entire legacy system, incremental evolution directly “fixes” end-user problems “one at a time”. This is the only choice left to many organizations that must continue to receive revenue from software systems to stay competitive. However, many incremental evolution initiatives do not sufficiently incorporate the end user’s point of reference (Dorda et al., 2000); such lack of consideration can leave end users unsatisfied because they may not see the benefit of the system evolution.

Problem and Solution Domains

Researchers have identified the two domains around which the entire field of software engineering revolves: the problem domain and the solution domain (Davis & Rauscher, 1979; Kaindl et al., 1998; Raccoon, 1995; Tsang & Magill, 1998). End users interact with the system through input files or a user interface. Because these users are directly concerned with system features, their perspective is always in the problem domain. Developers (and the software process team) are primarily concerned with creating and maintaining software development life cycle artifacts such as components; their perspective is therefore firmly rooted in the solution domain.

A major source of difficulty in developing, delivering, and evolving successful software is the complexity gap, as termed by Raccoon (1995), that exists between the problem and the solution domains, as shown in Figure 1. To view evolution from a single domain upsets the delicate balance between the two domains. Evolution focused solely on the problem domain may lead to changes that degrade the structure of the original code; similarly, evolution based solely on technical merits could create changes unacceptable to end users. External evolutionary pressures drive the implementation of new enhancements and functionality by causing developers to focus on implementing the
On Selection, Implementation, and Operations of ERP Systems
Sven Völker and Torsten Munkelt (2013). Sociotechnical Enterprise Information Systems Design and Integration (pp. 56-71).

www.igi-global.com/chapter/selection-implementation-operations-erp-systems/75874?camid=4v1a