Chapter IV
Towards Code Reuse and Refactoring as a Practice within Extreme Programming

Vijayan Sugumaran
Oakland University, USA

Gerald DeHondt
Grand Valley State University, USA

ABSTRACT

Software reuse has been discussed in the literature for the past three decades and is widely seen as one of the major areas for improving productivity. Agile development techniques were first developed in the mid-1990s as a code-oriented method of software development that seeks to improve upon the traditional plan-based methodologies. Both approaches bring value to the software development process. The purpose of this chapter is to propose a framework that will integrate the strengths of code reuse into the Extreme Programming methodology. It is believed that this approach will lead to a more effective method of software development.

INTRODUCTION

Agile software development first began in the mid-1990s as an alternative to the traditional Systems Development Life Cycle or plan-based methodologies widely implemented at the time. These lifecycle methodologies take a phased approach to systems development, requiring that one phase be completed prior to beginning the next phase (Hoffer et al. 1998). Agile methods, on the

other hand, focus on iterative software development, the continuous implementation of working code. From the beginning of the agile "revolution", specific methodologies have continued to refine these techniques, the most popular being Extreme Programming (XP). This approach was first implemented at Chrysler in 1996 (C3 Team 1998) as a way to accelerate development efforts while producing better software. XP is an implementation of Agile development techniques based upon twelve practices, one of which is refactoring. Specifically, refactoring involves modifying software to improve its internal structure in a way that does not alter the external behavior of the code (Fowler 1999).

At its core, Extreme Programming emphasizes rapid and frequent feedback to the customers and end users, unit testing, and continuous code reviews. By focusing on rapid iterations of simpler code, XP seeks to identify and resolve potential pitfalls in the development process early, leading to projects that remain focused on the ultimate goal – timely delivery of a well-designed and tested system that meets customer requirements. This methodology works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and tune the practices to their unique situation. It also seeks to implement the simplest design that will satisfy current user requirements (Lindstrom and Jeffries 2004) without attempting to anticipate future design or user requirements.

Nerur and Balijepally (2007) state that agile methods are people-centric, recognizing the value that competent people and their relationships bring to software development. In addition, it focuses on providing high customer satisfaction through three principles: quick delivery of quality software; active participation of concerned stakeholders; and creating and leveraging change (Highsmith 2002). Big upfront designs/plans and extensive documentation are of little value to practitioners of agile methods. Important features of this approach include evolutionary delivery through short iterative cycles — of planning, action, and reflection — intense collaboration, self-organizing teams, and a high degree of developer discretion (Nerur and Balijepally 2007). Organizations undertaking agile methodologies must invest in tools that support and facilitate rapid iterative development and versioning/configuration management (Nerur, Mahapatra, and Mangalaraj 2005). One way to achieve this is through investment in a suitable reuse strategy supporting agile development.

Reuse of existing software components has been an area of investigation since the early 1980's and is widely seen as one of the major areas for improving software productivity. By reusing previously tested and implemented code, it is hoped that developers will become more efficient by not having to solve the same problem twice. One of the key challenges with effectively implementing a program of software reuse is the identification of suitable components. If the identification process consumes more resources than saved in development time, these programs will not be undertaken by developers.

Additional challenges encountered in this approach include a lack of incentives, lack of available resources, suitable component identification, and necessary tools for customization and validation, among other items. In spite of this, appropriate component reuse can integrate previously implemented software into current development projects serving to propagate validated code within the application infrastructure (Ravichandran 2005).

Once the appropriate component has been detected, it is then up to the developers to refactor the software into a more suitable solution to the problem at hand. Both techniques, code reuse and refactoring, focus on efficiency in the systems development process. This research proposes an integrative framework that combines code reuse - for component identification - and refactoring - for improved software performance and maintainability - into an overriding process