Chapter I

Agile Development Methods and Component-Orientation: A Review and Analysis

Zoran Stojanovic, Delft University of Technology, The Netherlands

Ajantha Dahanayake, Delft University of Technology, The Netherlands

Henk Sol, Delft University of Technology, The Netherlands

ABSTRACT

Agile software development methods have been proposed as the way to address the problem of delivering high-quality software on time under constantly and rapidly changing requirements in business and IT environments. An agile development process is characterized by extensive coding practice, intensive communication between stakeholders, fast iterative cycles, small and flexible teams, and minimal efforts in system modeling and architectural design. This paper presents the state-of-the-art of agile methods and analyzes them along the selected criteria that highlight different aspects of their theory and practice. Certain limitations of agile methods are identified. The chapter presents the component paradigm as a way of balancing traditional (model-driven or plan-driven) and agile development, depending on the project settings. Service-based component concepts applied at the level of modeling, architectural design and development can ensure and strengthen agile development principles and practices, and at the same time introduce necessary agility to more traditional development. By using components, the software development process can easily scale in size, robustness, and the level of details. This provides an effective balance between the requirements for agility in software development and needs for a disciplined, design-driven way of building complex software.

This chapter appears in the book, Advanced Topics in Database Research, Volume 3, edited by Keng Siau. Copyright © 2004, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.
INTRODUCTION

EXTreme Programming (XP) and other Agile Methodologies (AMs) have started to gain considerable interest in the IT community during the last several years. They have been proposed as a way to build quality software systems fast and be able to easily adapt to rapidly and frequently changing requirements in the environment. Agile processes are focused on early, fast and frequent production of working code through the fast iterations and small increments. The processes are characterized by intensive communication between participants, rapid feedback, simple design and frequent testing. By their proponents, the software code is the main deliverable, while the role of system analysis, design and documentation in software development and maintenance is de-emphasized and to some extent ignored.

A number of processes claiming to be “agile” have been proposed so far. The best examples are eXtreme Programming (XP) (Beck, 2000), Scrum (Schwaber & Beedle, 2002), Feature-Driven Development (FDD) (Palmer & Felsing, 2002), Adaptive Software Development (ASD) (Highsmith, 2000), Crystal methods family (Cockburn, 2002) and DSDM (Stapleton, 2003). There have been attempts in applying agile values, principles and practices in earlier phases of the software life cycle, such as analysis and design, under the initiatives called Agile Modeling (Ambler, 2002) and eXtreme Modeling (Extreme, 2003). Efforts have been made to investigate how the Unified Modeling Language (UML) can be used in an agile process, as well as how to use the Rational Unified Process (RUP) (Jacobson, Booch & Rumbaugh, 1999) in an agile manner (Larman, 2001; Ambler, 2002). The authors of the listed agile approaches have formed the Agile Alliance and published the Agile Manifesto that represents a condensed definition of principles and goals of agile software development (Agile Alliance, 2001). These principles are:

- Individuals and interactions over processes and tools,
- Working software over comprehensive documentation,
- Customer collaboration over contract negotiation, and
- Responding to change over following a plan.

Agile Development (AD) paradigm challenges many of the common assumptions in software development. One of the most controversial is its rejection of significant efforts in up-front design in favor of a more evolutionary approach. According to its critics this is very similar to the so-called code-and-fix hacking strategy in software development. XP and other AMs minimize the role of common design techniques in traditional software development such as frameworks, design patterns, modeling tool support, modeling languages, model repositories and reusability. On the other hand, AD supporters claim that their methodologies include just enough design efforts for the project to be successful, and AD design is actually done in a different way than in traditional software processes. For example, in XP simple metaphor-like design, refactoring, architecture prototypes, and test-based design are used in an evolutionary way for software design purposes. These characteristics of XP and other AMs are opposite to the current initiatives and paradigms in software development, such as Model-Driven Development (MDD) (OMG, 2003). While both AD and MDD claim to address the challenges of high change rates, short time-to-market, increased return-on-investment and high quality software, their proposed solutions are actually very dissimilar. The question is whether principles and practices of both development paradigms can be combined in order to take the benefits of both approaches.
A Practical Perspective on Data Quality Issues
[www.igi-global.com/article/practical-perspective-data-quality-issues/51192?camid=4v1a](www.igi-global.com/article/practical-perspective-data-quality-issues/51192?camid=4v1a)

A Scalable Middleware for Web Databases
[www.igi-global.com/chapter/scalable-middleware-web-databases/7949?camid=4v1a](www.igi-global.com/chapter/scalable-middleware-web-databases/7949?camid=4v1a)