Creating a Dual-Agility Method: The Value of Method Engineering

B. Henderson-Sellers, University of Technology, Australia
M.K. Serour, University of Technology, Australia

ABSTRACT

Agile approaches to software development provide flexibility within the method, but provide little or no flexibility to support software process improvement (SPI). This second kind of flexibility is important to permit organizations to improve with time. To accomplish such improvement, a method-engineering approach is shown here to be useful. Examples of the application of method engineering to agile method construction are given based on the use of the OPEN process framework. A dual-agility method to support both intraproject and interproject flexibility can then be constructed by the selection of appropriate method fragments from this repository with the use of the accompanying guidelines. This is illustrated in an industry study of an organization transitioning to an agile software-development environment in which a minimalistic agile method was first constructed and then enhancements were made as the organization itself improved the process.

Keywords: action research; agile methods; dual-agility methods; metamodeling; method engineering; software process improvement; the OPEN Process Framework

INTRODUCTION

In the context of software (and indeed systems) development, it has long been recognized that it is not possible to design or select a single method for all conceivable situations (Cockburn, 2000; Turk, France, & Rumpe, in press), especially for all prevailing organizational cultures (Constantine & Lockwood, 1994). Despite this obvious truism, methodologists over the last two to three decades have created all-encompassing methods that purport to offer “the solution” (see, e.g., the discussion in Abrahamsson, Warsta, Siponen, & Ronkainen, 2003). Chau, Maurer, and Melnik (2003) argue that this resulted from the adoption of Tayloristic (1998) methods from the manufacturing domain into the software-development domain: methods that shift the focus from people to an overinflated importance of process for its own sake. Many of these large methods have been marketed by large consulting firms and others have been published in books for direct client adoption.
As a reaction to these large, so-called “heavyweight” methods, both in structured development and, more recently, in object-oriented development, the last few years have seen the rise of a group of lightweight or agile methods (such as Crystal, Scrum, XP) as offering more flexibility, particularly in situations of unstable requirements (e.g., Bettin, 2003; Newkirk, 2002) and in a reversion to a focus on the people involved in software development. These agile methods have, to date, been largely tried on small projects, and there is significant debate as to what extent they will or will not scale up (see, for example, Constantine, 2002; Reifer, Maurer, & Erdogmus, 2003). However, the research literature is scant (see, e.g., Abrahamsson et al., 2003) and many success stories are anecdotal (Turk et al., in press). Indeed, Lindvall et al. (2002) identify “an urgent need to empirically assess the applicability of these methods.” While these newer agile methodologies offer flexibility and responsiveness, they are static in that they offer little or no support for large-scale process improvement in the sense of CMM or SPICE.

Interestingly, although software process improvement (SPI) addresses issues of moving from one software maturity level to another, its focus is on assessment, not on the facilitation of that advancement. Consequently, the process needed to effect SPI is rarely discussed in the SPI literature (e.g., Jalote, 2002). Indeed, in a questionnaire survey by Rainer and Hall (2002), no questions were asked about these process maturation topics. As we shall show, one possible underpinning technology for SPI could be method engineering (ME): There has to date been no intersection of ME ideas and SPI ideas.

In this article, we evaluate these contradictory demands for a good software-development method: the freedom for software developers to be creative and not stifled by bureaucracy on a specific project vs. the need for a standard approach to developing software such that organizational learning and SPI might flourish. In the context of software engineering, a viable development method needs to be comprehensive enough to support the entire development cycle even if it is required to provide flexibility or agility. Agile methods are not designed nor expected to be incomplete or functionally disabled. Instead, they should add more features and functionality to existing methods in order to be able to handle the ever-changing world of requirements.

By examining the limitations of most of the existing lightweight and agile methods, we can clearly state that there is a real need for agile methods that support flexibility within the method’s application to a specific project (a.k.a. enactment), and also that support SPI, preferably underpinned by ME, that is, agility in two dimensions. We name these dual-agility methods, created by adding even more agility to the existing agile methods to be able to offer more flexibility, applicability, and functionality as organizations evolve and mature. Such a dual-agility method is engineered to be agile within a single project and also to offer agility across multiple projects so that the organization can also reap the benefits of software process improvement.

In this article, we propose a way of accomplishing this second kind of agility by the adoption of a method-engineering approach using the metamodel-based OPEN process framework or OPF (Firesmith & Henderson-Sellers, 2002). In the next section, we first review some variants on the notion of agility and then, in the subsequent section, identify the requirements for an agile methodology in an evolving organization. In the section titled “Method Engineering,” we begin to seek a solution in terms of method engineering, particularly by the use of the OPF. In “Empirical Evaluation,” we present a preliminary empirical evaluation before summarizing and concluding.

AGILITY VARIATIONS

Agile or Lightweight?

Many software methodologists and developers use the term agile and lightweight interchangeably. Here, we argue that the two words are not the same, and we should select the most appropriate word for our purpose to avoid any ambiguity.
Related Content

- **Novel Indexing Method of Relations Between Salient Objects**
  [www.igi-global.com/chapter/novel-indexing-method-relations-between/9211?camid=4v1a](www.igi-global.com/chapter/novel-indexing-method-relations-between/9211?camid=4v1a)

- **Semantic Enrichment in Knowledge Repositories: Annotating Semantic Relationships Between Discussion Documents**
  [www.igi-global.com/article/semantic-enrichment-knowledge-repositories/3347?camid=4v1a](www.igi-global.com/article/semantic-enrichment-knowledge-repositories/3347?camid=4v1a)

- **Similarity Learning in GIS: An Overview of Definitions, Prerequisites and Challenges**
  [www.igi-global.com/chapter/similarity-learning-gis/29669?camid=4v1a](www.igi-global.com/chapter/similarity-learning-gis/29669?camid=4v1a)