Structuring Complexity Issues for Efficient Realization of Agile Business Requirements in Distributed Environments

Richard Mordinyi, Eva Kühn, and Alexander Schatten

Space-based Computing Group
and
Christian Doppler Laboratory "Software Engineering Integration for Flexible Automation Systems"
Vienna University of Technology
1040 Vienna, Austria
{richard.mordinyi,eva.kuehn,alexander.schatten}@tuwien.ac.at

Abstract. One of the ideas of agile software development is to respond to changes rather than following a plan. Constantly changing businesses result in changing requirements, to be handled in the development process. Therefore, it is essential that the underlying software architecture is capable of managing agile business processes. However, criticism on agile software development states that it lacks paying attention to architectural and design issues and therefore is bound to engender suboptimal design-decisions. We propose an architectural framework, that by explicitly distinguishing computational, coordinational, organizational, distributional, and communicational models offers a high degree of flexibility regarding architectural and design changes. The framework strength is facilitated by a) combining the characteristics and properties of architectural styles captured in a simple API, and b) offering a predefined architectural structure to the developer of distributed applications to cope with complexities of distributed environments. The benefit of our approach is a clear architectural design with minimal mutual effects of the models with respect to changes, accompanied by an efficient realization of new business requirements.

Keywords: Agile Business Requirements, Agile Software Development.

1 Introduction

Business constantly changes. Therefore, software architectures should be able to manage agile business processes and need to have the ability to meet future changes and business needs. The field of agile software development [1] addresses exactly the challenges of an unpredictable, turbulent business and technology environment. Thus, the main question [2] in software development regarding software architecture is how to handle these changes better, while still achieving high quality. On the one hand, how many various numbers of eventualities have

A. Sillitti et al. (Eds.): XP 2010, LNBIP 48, pp. 202–207, 2010.

to be taken into consideration, and therefore how much time and effort should be invested into design and implementation of components or layers with respect to a good architectural design to cover all these circumstances, which eventually at the end may be not used at all. And on the other hand, performing no or hardly any planning ahead, and at the same time bearing the risk of redesigning the existing architectural design from the scratch, once it is not capable of handling the latest requirement [3]. The former case means, that software developers a) do not really focus on realizing the current requirement, but b) develop components which might not be needed in future. This results in higher development time and costs. Problems regarding architectural and design issues in ASD have been discussed in several papers, like [4], [5], [6]stating that ASD lacks paying attention to architectural and design issues and therefore is bound to engender suboptimal design-decisions.

In this paper we propose the Architecture Framework for Agile business requirements (AFA)[1] in ASD, in which it is explicitly distinguished between computational logic, coordinational, organizational, distributional, and communicational models. The five categories are independent of each other and therefore AFA offers a high degree of flexibility regarding architectural and design changes introduced by agile business processes. In the proposed framework the categories are not implemented by means of the software layer pattern [7] but rather in a component-oriented style [8], while making usage of the interceptor pattern. This allows to keep changes local or extend categories individually. AFA can be seen as an abstraction layer between applications and architectural styles, and as such it provides loosely coupling between the applications and their way of coordinating each other, between applications and architectural styles, between the applications and the way they process, distribute, and exchange information.

2 Related Work

Distributed middleware are mostly based on either dataflow style, such as pipes-and-filters, on data-centered style, i.e. a repository, or on implicit invocations, like publish-subscribe or event-based [9]. A detailed description regarding the advantages and limitations of the combination of architectural styles can be found in [10].

Concepts for agile software development (ASD) have been created by experienced practitioners and can be seen as a reaction to e.g., plan-based methods, which attach value to "a rationalized, engineering-based approach" [11]. By contrast, ASD has been proposed as a solution to problems resulting from an unpredictable world, and several agile methods have evolved over time, like Dynamic Systems Development Method [12], or XP [13].However, there is also skepticism [14] regarding ASD with respect to architecture design and implementation issues. One of them is that agile development is an excuse for developers to implement as they like, coding away without proper planning or design [4], [2] and consequently causing suboptimal design-decisions [5], [6].

[1] An implementation of the framework can be downloaded at `www.mozartspaces.org`