International Journal of Information Technologies and Systems Approach
Volume 15 • Issue 1

Using Logical Architecture Models for Inter-Team Management of Distributed Agile Teams

Nuno António Santos, CCG/ZGDV Institute, Portugal
https://orcid.org/0000-0002-8247-7253
Jaime Pereira, CCG/ZGDV Institute, Portugal
Nuno Ferreira, i2S Insurance Knowledge, S.A., Portugal
https://orcid.org/0000-0003-3561-331X
Ricardo J. Machado, CCG/ZGDV Institute, Portugal

ABSTRACT

Projects at large scale have been adopting agile practices in order to optimize how a group of teams deliver software. However, facing issues when dealing with work items that are dependent on each other, there is a need for well-defined interfaces and shared understanding of the existing knowledge. Systems thinking and modeling can have important contributions in managing a process with this nature. This article proposes a design framework called logical architecture-based distributed agile team management (LADATM) that uses logical architecture models to serve as the basis for the delivery of work items to distributed agile teams; support the identification of boundaries, dependencies, and coordination needs; and derive an agile team product backlog. By performing design science research (DSR) cycles, this article introduces the constructs of the LADATM framework. Afterwards, it demonstrates—using a research project—how it provided a systematized approach for process management decision support for agile teams based in requirements models and through shared understanding.

KEYWORDS

Agile, Agile Architecting, Coordination, Dependencies, Logical Architectures, Models, Product Backlog, Requirements, Requirements Communication

INTRODUCTION

The digital transformation of businesses increased the creation of new software ecosystems. Additionally, software solutions allow third-party integration (e.g., using Application Programming Interfaces – APIs) towards full support of the supply chain. Many times this means that software development teams are no longer developing software "alone", but rather cooperating with other teams and organizations. While agile software development (ASD) has been adopted to optimize how a team delivers software, its use in scaled and distributed contexts is still the object of research, with some emphasis on planning and inter-team coordination (Moe & Dingsøyr, 2017).

Software development processes in these contexts need to address not only how a software increment – delivered by a team – fits in the overall solution, but also how teams must define their

DOI: 10.4018/IJITSA.289996

boundaries, interfaces, dependencies, and priorities. Only after such context is clarified is it possible to apply ASD practices in a scaled context – i.e., the concept of "*large-scale agile*" (LSA) (Dingsøyr & Moe, 2014).

The process of delivering software using more than one development team – and often distributed – faces issues of dependencies, boundaries, coordination and/or synchronization. Challenges include making decisions, setting goals, communicating, building trust and managing the team (Owen, 2016). With the rise of ASD, these processes were rethought (Dingsøyr, Bjørnson, Moe, Rolland, & Seim, 2018).

In process management, architectures are an artifact from model-based systems thinking capable of supporting a set of coordination decisions. Additionally, architecture is a central artifact when scaling up agile methods, as it is explicitly present in LSA frameworks, like Scaled Agile Framework (SAFe) (Leffingwell, 2016), Large-Scale Scrum (LeSS) (Larman & Vodde, 2016), Disciplined Agile Delivery (DAD) (Scott Ambler & Lines, 2012), Scrum@Scale (Sutherland, 2018), Nexus (K Schwaber, 2015) or Enterprise Scrum (Greening, 2010). Communities such as Industrial XP (Kerievsky, 2005) include "Evolutionary Design" practices, and the "Spotify model" (Kniberg & Ivarsson, 2012) has specific architecting roles. Other LSA approaches include explicit architecture practices, namely the Agile Product Line Architecting (APLA) (Díaz, Pérez, & Garbajosa, 2014), a tailored XP model for large-scale projects (Cao, Mohan, Xu, & Ramesh, 2004), or a hybrid RUP+Scrum (Cho, 2009). Although acknowledging the importance of architecture in managing inter-team processes in an LSA context, these approaches lack a structured approach for using such information to manage the software delivery process. Models are about presenting an abstraction of reality towards a shared understanding of the problem, but a proper analysis requires depicting their input in assigning work, deriving dependencies, and managing inter-team communication and coordination.

This paper addresses the following research question: "**How to use logical architectures for process management within LSA projects?**". By following the Design Science Research (DSR) methodology (Hevner, March, Park, & Ram, 2004), this research describes a distributed agile team management framework, called *logical architecture-based distributed agile team management* (LADATM). The LADATM framework is used as basis for managing the process of setting delivery boundaries, communicating the requirements, coordinating, and synchronizing multiple teams. Each team's work and backlog are also derived from models. The approach, presented in further sections, has as a starting point a logical architecture. It contributes by proposing systems thinking and modeling with an important role in managing the process of LSA. Afterward, the research focuses on analyzing the teams, available artifacts and the processes. The framework was applied in a live research project called "Unified Hub for Smart Plants" (UH4SP).

RESEARCH METHOD

The research question is how to use logical architectures for process management within LSA projects. For that reason, this research focuses on developing a logical architecture-based distributed agile team management framework. The process is the central artifact under design, even though product artifacts (namely UML diagrams) are also designed throughout the process. This research is structured according to the DSR methodology (Hevner, March, Park, & Ram, 2004), mainly due to the reason that the design-science paradigm seeks to extend the boundaries of human and organizational capabilities by creating new and innovative artifacts. Within the case of Information Systems (IS) research, DSR focuses on the design of IT artifacts. These artifacts are broadly defined as *constructs* (vocabulary and symbols), *models* (abstractions and representations), *methods* (algorithms and practices), and *instantiations* (implemented and prototyped systems). DSR describes the world as acted upon (processes) and the world as sensed (artifacts) (Hevner et al., 2004) and its instantiation have a physical existence in the real world (Gregor & Jones, 2007).

Related Content