

# Balancing Agile and Disciplined Engineering and Management Approaches for IT Services and Software Products

Manuel Mora  
*Autonomous University of Aguascalientes, Mexico*

Jorge Marx Gómez  
*University of Oldenburg, Germany*

Rory V. O'Connor  
*Dublin City University, Ireland*

Alena Buchalcevo<sup>á</sup>  
*University of Economics, Prague, Czech Republic*

A volume in the Advances in Systems Analysis,  
Software Engineering, and High Performance  
Computing (ASASEHPC) Book Series



Published in the United States of America by

IGI Global  
Engineering Science Reference (an imprint of IGI Global)  
701 E. Chocolate Avenue  
Hershey PA, USA 17033  
Tel: 717-533-8845  
Fax: 717-533-8661  
E-mail: [cust@igi-global.com](mailto:cust@igi-global.com)  
Web site: <http://www.igi-global.com>

Copyright © 2021 by IGI Global. All rights reserved. No part of this publication may be reproduced, stored or distributed in any form or by any means, electronic or mechanical, including photocopying, without written permission from the publisher. Product or company names used in this set are for identification purposes only. Inclusion of the names of the products or companies does not indicate a claim of ownership by IGI Global of the trademark or registered trademark.

Library of Congress Cataloging-in-Publication Data

Names: Mora, Manuel, 1961- editor. | Gómez, Jorge Marx, 1959- editor. | O'Connor, Rory V., 1977-2019 editor. | Buchalcevoová, Alena, 1958- editor.  
Title: Balancing agile and disciplined engineering and management approaches for IT services and software products / Manuel Mora, Jorge Marx Gómez, Rory V. O'Connor, and Alena Buchalcevoová, editors.  
Description: Hershey, PA : Engineering Science Reference, an imprint of IGI Global, [2021] | Includes bibliographical references and index. | Summary: "This book explores the theoretical foundations of balanced design methods for software and IT service"-- Provided by publisher.  
Identifiers: LCCN 2020002824 (print) | LCCN 2020002825 (ebook) | ISBN 9781799841654 (hardcover) | ISBN 9781799853794 (paperback) | ISBN 9781799841661 (ebook)  
Subjects: LCSH: Information technology--Management. | Agile software development. | Project management.  
Classification: LCC HD30.2 .B349 2021 (print) | LCC HD30.2 (ebook) | DDC 004.068/4--dc23  
LC record available at <https://lccn.loc.gov/2020002824>  
LC ebook record available at <https://lccn.loc.gov/2020002825>

This book is published in the IGI Global book series Advances in Systems Analysis, Software Engineering, and High Performance Computing (ASASEHPC) (ISSN: 2327-3453; eISSN: 2327-3461)

British Cataloguing in Publication Data

A Cataloguing in Publication record for this book is available from the British Library.

All work contributed to this book is new, previously-unpublished material. The views expressed in this book are those of the authors, but not necessarily of the publisher.

For electronic access to this publication, please contact: [eresources@igi-global.com](mailto:eresources@igi-global.com).

# Chapter 11


## A Framework for Analyzing Structural Mechanisms Deployed to Support Traditional and Agile Methods: Making Sense of “Democratization” in the Software Development Workplace

**Michal Dolezel**

 <https://orcid.org/0000-0002-5963-5145>

*University of Economics, Prague, Czech Republic*

**Alena Buchalcevo**

 <https://orcid.org/0000-0002-8185-5208>

*University of Economics, Prague, Czech Republic*

### **ABSTRACT**

*People rely on structures to make their worlds orderly. This chapter conceptually probes into the problem of the differences between organizational structures deployed in traditional and agile environments. The authors develop an argument that all common forms of organizational entities can be classified by involving a two-dimensional classification scheme. Specifically, they constructed a typology to examine the issues of formal vs. informal authority, and disciplinarity vs. cross-functionality in terms of their significance for traditional and agile software development workplaces. Some examples of concrete organizational forms—including traditional project team, independent test team, self-organizing agile team and developers’ community of practice—are discussed. In sum, they argue that by employing this classification scheme, they can theorize the nature of the on-going structural shift observed in conjunction with deploying agile software development methods. They acknowledge that the structures have fundamentally changed, terming the move “democratization” in the software development workplace.*

DOI: 10.4018/978-1-7998-4165-4.ch011

## INTRODUCTION

Over the last two decades, Agile Software Development Methods (ASDMs) have rapidly grown in popularity. While ASDMs were originally coined by a group of “organizational anarchists” (Fowler & Highsmith, 2001) who wanted to “undo the damage that waterfall had done to ... [the software development] profession” (Schwaber, 2013), the methods seem to be nowadays widely accepted as *the* mainstream way of software development. The shift from the plan-driven methods to ASDMs can be characterized as a move from the rigorous and well-defined processes towards rapid, people-oriented and customer-centered software practices (Cohn, 2010). Indeed, the shift was supposed to also bring some fundamental changes in the overall philosophy and value-orientation with regard to software development activities. These changes were exemplified in the Agile manifesto and its twelve principles (Fowler & Highsmith, 2001). However, despite the huge attention which the Agile manifesto attracted in the past decade, the question whether it is yet making a real, on-going impact remains open. In our view, this may be due to the essential fluidity of the Agile manifesto, despite the fluidity being intentional (Hohl et al., 2018).

In contrast to being fluid and uncertain, in many aspects of their everyday lives people naturally tend to look for tangible guiding principles and structures. In that sense, social psychologists argue that “structure provides people with the means to achieve their desired ends”, as “[w]hen the world is orderly and structured, people can make sense of events” and predict the future (Laurin & Kay, 2017). Notably in the world of work, structures play an important role. In this context, a structure means a generic mechanism allowing workers to get their work done. In particular, organizational structures, which are typically over-simplistically represented by corporate organizational charts, are frequently used in practice and examined by management scholars (Daft et al., 2010; Pugh et al., 1969). It is reasonable to expect that software teams are no exception in their need for a structure.

Apparently, due to ASDMs in general, and scaled agile frameworks in particular, the commonly used approaches to organizational structuring within the software world have changed during the last decade or so. It appears that software practitioners hold that the structures which previously worked in plan-driven settings (e.g. silo-ed functional teams) are now partly or fully dysfunctional in agile settings (Noordeloos et al., 2012). Also, emerging structuring mechanisms such as cross-functional teams, communities of practice (Cohn, 2010), agile centers of excellence (Knaster & Leffingwell, 2019), agile chapters and guilds (Smite et al., 2019) etc. have appeared in the organizational repertoires of software development enterprises. One of key drivers for the adoption of these structures seem to be the growing popularity of scaled agile frameworks. These frameworks now play a dominant role by demonstrating how to conduct and structure software development activities in an agile manner at a large scale (Ebert & Paasivaara, 2017; Kalenda et al., 2018). In some of these frameworks, it is possible to find a prescription of the role delineation to be implemented (Smite et al., 2019), which is an important structural aspect of modern organizations (Daft et al., 2010).

In sum, we believe that the issues such as how practitioners structure the fluid world of ASDMs, and whether they do so at all, are of great practical and theoretical relevance. Moreover, the current “restructuring” trend, i.e. the move from certain type of structures to others, is an interesting phenomenon *per se*. In particular, a deeper understanding of the emerging agile organizational structures may help software leaders and managers to understand the nature of the shift from traditional to agile methods. Also, it may help them to choose an appropriate structure by finding a fit between their context and available organizational options. In addition, we propose that the present theory-founded analysis can stimulate further research on this currently under-researched topic. Therefore, the question we pose herein is:

*What is the impact of the shift from plan-driven to agile methods on the selected structural aspects of the software development organization?* In this chapter, we conceptualize this problem by introducing a typology of organizational arrangements through demonstrating the influence of two important dimensions – namely, functional orientation and authority.

The chapter is organized as follows. Section 2 reviews the theoretical foundations. Then, Section 3 introduces the typology. Next, Section 4 discusses the selected aspects of the typology and its application. After that, Section 5 demonstrates how the typology can be used. Finally, Section 6 provides concluding remarks.

## **THEORETICAL PRELIMINARIES**

Below we provide the starting point for our typology construction effort by presenting theoretical foundations. Specifically, Section 2.1 outlines three core concepts. Expanding on them, Sections 2.2 and 2.3 define the role of authority and functional orientation, respectively.

### **Examining the Context Beyond a Particular Software Team**

We theoretically frame the core of the organizational structuring activities outlined in Section 1 as the issue of designing effective inter- and intra-team Communication, Coordination and Integration mechanisms (Daft et al., 2010), further abbreviated as **CCI**. The definitions of these fundamental concepts are provided in Table 1.

*Table 1. Definitions of core concepts*

<b>Concept</b>	<b>Definition</b>
Communication	The process of “imparting or interchanging thoughts, opinions, or information by speech, writing, or sign” (Mishra & Mishra, 2008)
Coordination	The process of “managing dependencies between activities” (Malone & Crowston, 1994) The process of “harmonious adjustment or interaction of different people or things to achieve a goal or effect” (Mishra & Mishra, 2008)
Integration	The process of “achieving unity of effort among the various subsystems in the accomplishment of the organization’s task” (Lawrence & Lorsch, 1967)

In general, software development is a concord of activities occurring not only between software workers seen as individual players, but many times also between whole organizational entities (i.e. software teams) seen as uniform people aggregates. To elaborate on the latter abstract notion, not only individual people, but also whole teams, are expected to interact with their counterparts in order to get the work done (Begel et al., 2009). Stated more accurately, for an effective performance of their work duties, software workforce must implement and utilize effective inter and intra-team CCI mechanisms, supplemented with appropriate decision-making patterns and management responsibilities (Kalliamvakou et al., 2017).

In a pursuit of consistency, the actions of individual team members are expected to be aligned with a collective will of the aggregate they are members of. Among the CCI aspects, integration seems to be

a salient set of mechanisms for preventing conflicts among software professionals. For example, a “silo mentality” among various software professions is an issue claimed to be resolvable by introducing more integrated organizational models (Jesse, 2018). Similarly, some companies experiment with a software role re-definition by integrating previously separated roles together, naming such model “Combined Software Engineering” (Dolezel & Felderer, 2018).

It is important to note that, in principle, a number of possible CCI patterns and means are available for use alongside organizational structures. To briefly illustrate the richness and broadness of the repertoire, the work of Bick et al. (2018) stresses the importance of inter-team planning meetings. Next, Šmite et al. (2017) identified six distinct mechanisms for networking, one of them being associated with software workforce’s participation in forums and communities of practice. Finally, Bjørnson et al. (2018) propose that there are three crucial enabling mechanisms at the inter-team level: shared mental models, closed-loop communication and trust.

Aside from the above less-tangible, mostly informal aspects of CCI, research literature in the field of management studies and information systems commonly use the term “structural overlays” to denote a more formalized organizational arrangement (Balaji & Brown, 2014). Along this line of research, structural overlays are understood as a broad concept, encompassing, for example, also governance committees and various liaison roles. Similarly, in the software engineering field, Bannerman (2009) introduced the concept of Software Development Governance (SDG). In his understanding, governance means “managing the management function of the organization” (Bannerman, 2009). A similar view is particularly popular in research literature focusing on IT Governance (Brown & Grant, 2005). Accordingly, SDG’s main purpose is “to establish how the organization’s software development capability is sustained (in terms of structures, processes and relational mechanisms) to meet its engineering and business needs” (Bannerman, 2009).

In line with the cited research, we assume that in order to analyze organizational mechanisms and patterns holistically, it is necessary to look beyond the immediate horizon, i.e. beyond individual software teams. That means, an individual or single-team centered perspective seems to be insufficient when trying to gather a complete snapshot of the CCI vehicles implemented in a software enterprise. Therefore, our main aim is to contrast these entities against each other, and inductively derive certain conclusions about their essential features. We expect that those conclusions should be generalizable to a broader population of CCI entities encountered in the software industry. Such analysis is to provide important insights about the nature of the shift from traditional methods to ASDMs. In this case, we mostly rely on two important aspects: (i) authority, a concept adopted from sociology; (ii) functional orientation, a concept adopted from management and organizational studies.

## **Authority**

The on-line Cambridge dictionary<sup>1</sup> defines authority as “the moral or legal right or ability to control”. In our everyday lives, we have been greatly influenced by numerous authorities. For example, obeying parental authority was a natural act during certain parts of our lives. As adults, we respect both formal and informal authorities, attributable to concrete people and institutions. Here we draw upon the concept of authority as defined by the European sociologist Max Weber (Shepard, 2013). Weber distinguished among three types of authority: (i) traditional, which is based on long-established norms, customs and patterns in society, (ii) legal-rational, which “derives from legal norms” (Spencer & Spencer, 1970), and

(iii) charismatic, which can be associated with a concrete person and his/her behavior strongly appealing to others. See Table 2 for a summary with examples.

*Table 2. Forms of authority in the societal and organizational context (adapted from Shepard, 2013)*

Type of authority	Essential mechanism	Example
Charismatic	Based on a leader’s personal qualities	<i>Society:</i> Power of charismatic and influential figures of the public life, e.g. Dalai Lama or Nelson Mandela
		<i>Modern organizations:</i> Power of informal influencers (i.e. people with trust and respect) within an organization
Traditional	Rooted in customs, habits and rituals	<i>Society:</i> Power of early kings based on divine right (i.e. a belief that power is given to them by God)
		<i>Modern organizations:</i> Significant decision-making authority in an area, which is exercised by a person or department for a number of years or even decades, while not being formally mandated
Rational-legal	Associated with a formally defined standing or position	<i>Society:</i> Power of the political representation of a country with a democratic constitution
		<i>Modern organizations:</i> Power of the board of directors

In reality, the above types of authority can be commonly mixed. For example, the Queen of the United Kingdom holds both rational-legal and traditional, and perhaps also charismatic authority.

In the realm of organizations, authority is one of the essential mechanisms which most organizations are enacted by. In that sense, Weber’s framework is accepted as a theoretical basis which his successors in the field of organizational sociology frequently build upon (Graham, 1991). Naturally, for the purpose of organizational analysis, the framework needs to be conceptually adapted. To give an example, rational-legal organizational authority can be viewed as “employees’ belief in the legality of [organizational] rules and the right of those elevated to positions of authority to issue commands” (Daft et al., 2010). Similarly, charismatic authority can serve as a conceptual basis for contrasting charismatic, transformational and servant leadership in organizations (Graham, 1991). We consider the remaining type of Weberian authority, i.e. traditional authority, as unsuitable for the analysis of modern software organizations.

### **Software Disciplines and Their Functional Orientation (i.e. “Silo-ism”)**

Within the broad field of computing (Denning, 2018), it is possible to recognize contours of several software engineering (SE) *disciplines*. (Note that the term “discipline” was introduced by RUP (Kroll & Kruchten, 2003) and as such it is used in a different, less encompassing sense than in the case of academic disciplines.) Their role has been commonly seen as fundamentally structuring the software development landscape. In metaphorical terms, particular SE disciplines can be thought of as abstract *containers* for various SE activities, roles, responsibilities and resulting artefacts within a given domain of expertise. Put simply, one SE discipline groups those SE activities that have a common denominator. Examples of the SE disciplines include software testing and software architecture. These are also commonly treated as two distinct SE professions in industry, despite the fact that both are often subsumed into the generic profession of software engineering (Miller & Voas, 2008). Importantly, not all roles

found within software development methods always exist in the reality of software development companies (Borges et al., 2012).

In the following text, for the sake of simplicity, we do not draw a clear-cut separator between the SE disciplines defined in the software development methods vs. practice-based professions. While their mutual relationship is complex and certainly not 1-to-1, the extent of this chapter does not allow a more nuanced discussion than a few notes below.

Regarding the issue of functional orientation in software development, creating a functional organizational structure means putting professionals with alike responsibilities together in an organizational sense (Quinnan, 2010). Management theory suggests that the key benefit of this type of structuring is in having “all human knowledge and skills with respect to specific activities ... consolidated, providing a valuable depth of knowledge” (Daft et al., 2010). Until recently, functional organizational structures had been arguably the most popular way of organizing the SE activities, resonating greatly with the disciplinary separation. Specifically, functional organizational structures were used to introduce a set-up when “several discipline-oriented departments” provided their professionals into software programs, with each of these professionals being “highly qualified in a fairly narrow area” (Quinnan, 2010).

With the advent of agile development, however, this thinking has been fundamentally reconsidered. In essence, the proponents of agile approaches argue that functional structures are being materialized as unwanted silos, which moreover results in a “deep-rooted dependency on the organizational hierarchy” (Schatz & Abdelshafi, 2005). This contrarian logic drives presently the boom of organizational efforts that are to “break down organizational silos” in order to introduce agility in software organizations (Motingoe & Langerman, 2019). However, breaking down silos should be followed by looking for new forms of structural mechanisms. This is due to the silos having not only unwanted aspects, but also a number of positive. In particular, they serve as a home base for disciplinary professionalism.

## **TPOLOGY DEVELOPMENT**

In this section, we develop a typology of basic organizational structures seen in traditional and agile environments.

### **Essentials**

Researchers typically use typologies as means for contrasting complex relationships among various instances of a concept. In that sense, typologies are considered to be effective means for theory-building (Reynolds, 2016), although many scholars do not consider typologies being theories *per-se* (Stol & Fitzgerald, 2015). Typology-development efforts can commonly be found in many areas of the social sciences, including management studies (Cornelissen, 2017). Aside from typologies, taxonomies have been frequently used for presenting a similar type of research outputs (Pugh et al., 1969). In the software engineering domain, the latter term seems to be more common than the former, especially when the presented output is based on empirical research results (Ralph, 2018).

In our case, we followed the idea that typologies and taxonomies can result from various theory-building endeavors, both empirical (Pugh et al., 1969) and conceptual (Cornelissen, 2017). Being the latter case, our approach to theory-building was driven by two underlying notions. Firstly, the discussions that revolve around the role of formal and informal authority in software development teams have



intensified recently, mainly thanks to the popularity of the self-organization notion (Hoda et al., 2010). We thus held that distinguishing between formally organized and self-organized teams is an important aspect of the analysis of organizational structures in software development. Secondly, functional specialization is among the top discriminating attributes that have been traditionally used to characterize organizational structures in management studies (Pugh et al., 1969). We thus chose this criterion as the second discriminator.

In sum, our initial position is that every organizational structure can be defined in terms of a unique combination of the authority and functional orientation modes. Moreover, it can be characterized by a set of three CCI mechanisms we have defined in Table 1. The reason for distinguishing among different authority and functional orientation modes is to demonstrate that managers and administrators carrying out their duties have a possibility to choose from a number of fundamentally different organizational means. Illustrating the nature of the choice, Morgan (2006) argued that one's preference of a concrete organizational mechanism is closely related with his/her inner beliefs about the world of work, and that the chosen mechanism also says something about his/her own personality. For example, in the software engineering domain, many professionals might tend to prefer largely mechanistic views of organizations based on the fact they imagine ideal organizations as, stated metaphorically, *well-oiled machines*. To the contrary, the proponents of agile methods have been strong advocates of seeing organizations as living organisms or, more precisely, complex adaptive systems (Kautz & Zumpe, 2008).

## **Typology**

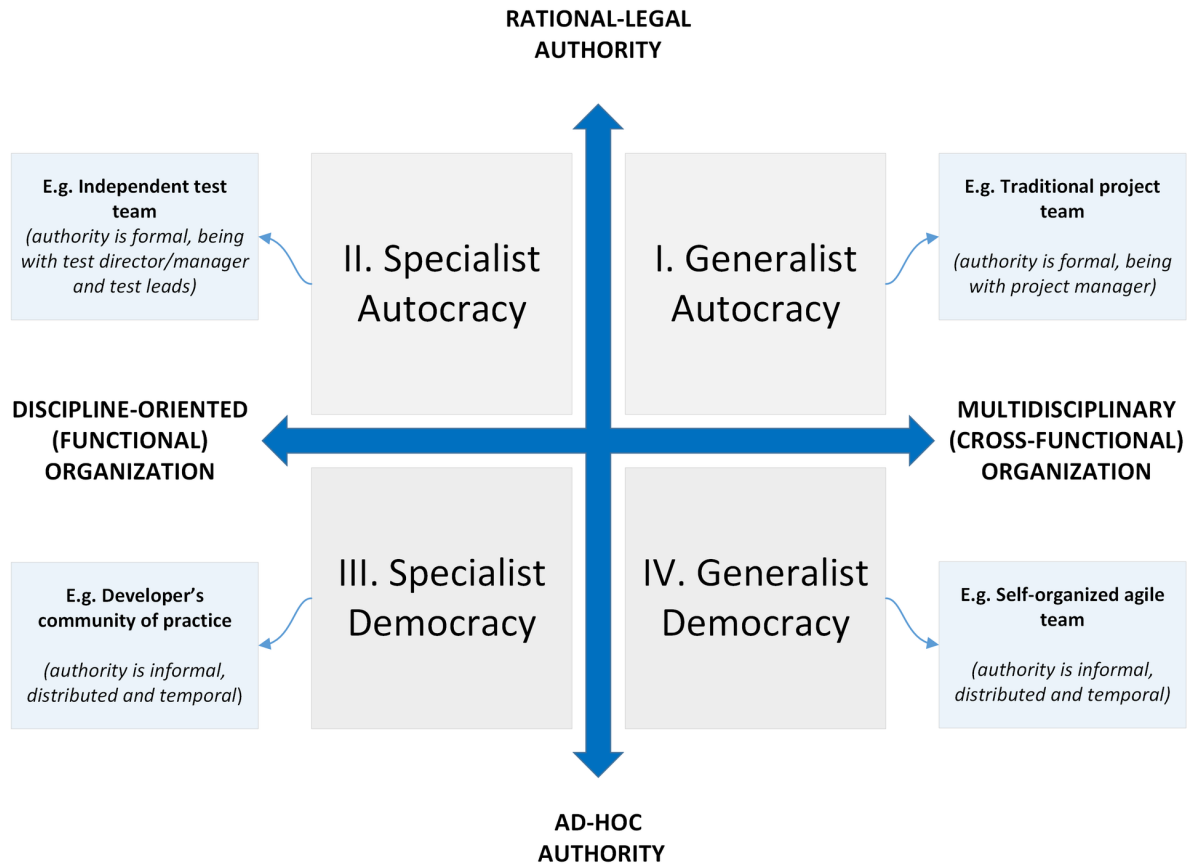
The resulting typology is portrayed in Figure 1, distinguishing four ideal types of organizational entities. The view proposed is that there are two fundamental decisions which must be made when organizing activities within the domain of software engineering. Put differently, there are two important variables for characterizing any organizational entity related to software development.

First, the type of authority discussed in Section 2 determines a number of distinct organizational features. How authority is exercised in a particular organizational entity is closely related to the issues of power and control, both being bound to the matter of formal chain of command. Interestingly, self-management modes of organizing seem to presently outgrow hierarchical modes, which were considered the dominant way in the past (Nalebuff & Brandenburger, 1997). Based on this finding, the dichotomy portrayed by the poles of *Ad-hoc* or non-formalized authority and *Rational-legal* or formalized authority has been chosen as the first discriminator. (Note that the term “ad-hoc authority” refers not only to the concept of charismatic authority defined in Section 2.2, but also to additional hypothetical forms of authority that are principally different from the ratio-legal one – e.g. *consensus-based authority*.)

Second, it should be considered whether the organizational entity is to promote segregation among different SE disciplines/professions. As suggested in Section 2, certain traces indicate the possibility that many SE professionals identify with a certain SE discipline (e.g. software testing, software architecture etc.). However, little consensus exists whether a discipline-oriented approach should be prioritized over cross-functionality when organizing software teams. Hence, it is meaningful to treat this criterion as an important discriminator.

Together, combining the above two variables, the typology matrix results in four possible organizational modes, which are discussed in detail below. A brief description of the entities commonly belonging to the four quadrants is also provided.

Figure 1. Typology matrix of organizational entities in traditional and agile organizational settings



**Generalist autocracy (Quadrant I).** For organizational entities in Quadrant I, *rational-legal authority* and *multidisciplinarity* are the key characteristics. These entities can be called *Generalist autocracies*. This is to say authority and decision-making are predominantly associated with a single person or a small subgroup of people (i.e. a single manager or a few managers), and this person (subgroup) is expected to be a generalist in terms of his/her ability of making sense of a wide range of problems brought by more specialized members of the entity. In general, forming a single organizational entity by putting together a team of specialists with a different background and work focus is to achieve synergies through their multidisciplinary collaboration (Quinnan, 2010). Arguably, based on this central promise, such an approach has become a de-facto standard in the domain of software project management during the past decades. By contrast, the potential disadvantage of this approach is differing expectations and espoused values of the team members, possibly resulting in conflicts (Zhang et al., 2014).

Despite the strong bonds of this organizational mode to the general project management principles, additional organizational entities falling under this category can be certainly identified. These may include, for example, a cross-functional Systems & Software Capability Board that govern the activities of engineering councils (Bannerman, 2009); static product development teams led by a product development manager in product-focused development organizations – i.e. when individual projects led by project

managers are spawned only for customer-centric customizations (Vähäniitty & Rautiainen, 2005); and some Software Process Improvement (SPI) initiatives carried-out through bureaucratic organizational infrastructures – e.g. when a generalist SPI department with a broad process focus is created (Ngwenyama & Nielsen, 2013).

### **Generalist Democracy (Quadrant IV)**

These organizational entities are defined by the combination of *ad-hoc authority* and *multidisciplinarity*. We call these entities *Generalist democracies*, as they do not exhibit a centralized authority. Instead, authority may be less obvious – either evenly distributed among the members of the entity, or even highly volatile. Being in this configuration principally on their own, specialists must find a common ground to be able to work effectively together. In other words, there is no central authority to negotiate and translate meanings among the members involved, or to solve their conflicts. This organizational mode will typically be associated with ASDMs and the concept of team self-organization (Cohn, 2010).

However, it is worth mentioning that while the concept of self-organization (i.e. ad-hoc authority) has been heavily popularized in connection with ASDMs, it has not come out of nowhere. For example, Humphrey introduced a very similar concept of self-directed teams in his work on *Team Software Process* in the late 1990s (Humphrey, 1998). Tracing the roots even more to the past, the concept of self-managed, voluntarism-driven quality circles was introduced by Kaoru Ishikawa as a part of the “Japanese way” to quality control in manufacturing in 1985 (Lillrank, 1995). While partly forgotten, these seminal organizational concepts (entities) are infrequently mentioned in the software engineering domain even nowadays.

### **Specialist Autocracy (Quadrant II)**

The entities labeled *Specialist autocracies* and located in Quadrant II are those with the key attributes of *rational-legal authority* and *discipline-oriented organization*. Similarly to *Generalist autocracy*, authority is expected to be in the hands of a single person or a small subgroup of people. In contrast to *Generalist autocracy*, the entities are uniform in terms of the profession or specialization they are dominantly formed by.

During the typology development process, we expected to see these entities being associated primarily with plan-driven or traditional methods. Nevertheless, exploring the state of knowledge regarding these entities, we found that extant SE research literature says surprisingly little about the structural organization of software-developing enterprises working in the plan-driven methods paradigm (see also Bannerman, 2009). Nonetheless, an implicit assumption seemed to be that organizational functions responsible for software development and software operations were frequently separated (Quinnan, 2010), at least in large enterprises. Moreover, software development professionals (e.g., developers, testers, architects) were frequently organized within additional disciplinary “silos” (Motingoe & Langerman, 2019). While the size of the silo could be variable (e.g. a team, group, department, functional division etc.), the key principle (i.e. putting alike skills together) remained the same. As explained in Section 2, similar organizational mechanisms were implemented largely with the aim to effectively and efficiently collect, keep and share specialized domain expertise (Daft et al., 2010). In addition, the mechanism might contribute to an increase in job satisfaction by offering staff members more diverse career opportunities (C. Jones, 2010).

### Specialist Democracy (Quadrant III)

Finally, within the last type of entities called *Specialist democracy*, *ad-hoc authority* is combined with *discipline-oriented organization*. In essence, authority is informal, possibly being repeatedly negotiated through various mechanisms. This may provide opportunities for charismatic leaders to get into power (see Section 2.2). Also, other means of acquiring power or reaching group consensus are possible in these entities (e.g., via polling).

We speculate that these entities may be slightly less common. In any case, examples in the software development world include some profession-oriented communities of practice (CoPs), for example Developer CoPs (Paasivaara & Lassenius, 2014) and CoPs focused on particular technologies (e.g. a programming language). Also, Architecture Councils, where cross-team decisions are expected to be formed on a peer-to-peer basis (Clerc et al., 2011) can be mentioned.

## **TYPOLGY APPLICATION**

This section illustrates how the typology can be used for demonstrating the importance of differences between formal and informal authority, and functional and non-functional orientation of selected organizational entities. For this purpose, we selected four examples: (i) Traditional project team (Section 4.1); (ii) Self-organizing Scrum team (Section 4.2); Independent test team (Section 4.3); Developer's Community of Practice (Section 4.4). By doing so, we also demonstrate the usability and viability of the typology.

### **Traditional Project Team as a Generalist Autocracy**

Historically, software development activities were structured through defining tasks and responsibilities, which together formed a conceptual basis of the first software development lifecycle models (Kneuper, 2017). Moreover, the domain of software engineering took a lot of inspiration from the project management world. This fact is quite apparent from the review of certain fundamental software engineering sources. For example, our reading of the Software Engineering Body of Knowledge (SWEBOK; IEEE, 2014) suggests the existence of a strong conceptual bridge to the Project Management Body of Knowledge (PMBOK). The former source notes that, in general, many professionals understand the term “software engineering management” as a synonym for the execution of project management activities within a software development context. In essence, such an understanding strongly promotes the view that a central figure (a project manager) should be designated, and a multidisciplinary entity directed by him/her (a project team) should be formed.

It is therefore not surprising that the organization of software teams has been rooted in common project management principles. Within the project management paradigm, every project is led by a project manager; he/she is “the leader of the team, regardless of what authority the project manager may have over its members” (PMI, 2013). A project team is then “comprised of individuals from different groups with specific subject matter knowledge or with a specific skill set to carry out the work of the project” (ibid.). For the sake of simplicity we assume that the project team works within a “projectized” organization structure, as defined by the PMBOK (PMI, 2013). The term “projectized” is used to denote a

set-up in which skills from different domains are brought into the project team by people with different specialization. Hence traditional project teams can be labelled as *Generalist autocracies*.

## Communication, Coordination and Integration

First and foremost, it should be noted that while different project managers may certainly exhibit very different management styles, a common belief is that many organizations appoint “a project manager (PM) with ‘command and control’ responsibilities” (Taylor, 2016). Without doubts, such a mentality would influence the atmosphere in the team to a great extent. Therefore, a typical project team will be mobilized by directive means of rational-legal authority. This management style fundamentally influences all CCI mechanisms, and it can be expected that those mechanisms will typically be centralized (Schwaber, 2004). For example, some project managers may insist on being the “hubs” in the majority of communication occurring within the project team, authorizing all key decisions. However, it is increasingly recognized that a number of common issues in the modern organizational world can be sorted-out by decentralizing decision-making through empowering people (Daft et al., 2010).

What we consider interesting is that previous research in management studies clearly shows that the project management profession represented a field with quite a convergent view on what constitutes appropriate professional knowledge, having strong roots in Tayloristic organizational notions (Hodgson, 2002) and the above mentioned command&control management styles. Yet, this mechanistic view seems to be significantly re-defined today (Fontana et al., 2014). This development opens up a possibility to move software development activities to another quadrant of our typology.

## Self-Organizing Scrum Team as a Generalist Democracy

As already stated in Section 1, the Agile Manifesto, published in 2001, articulated 4 values and 12 principles. Among the principles, team self-organization took a prominent position. For the proponents of this philosophy, relying on a directive project manager role is hardly acceptable (Cohn, 2010; Schwaber, 2004). In their research, Hoda, Noble and Marshall (2010) compare agile leadership with traditional project management. They argue that “Leadership in Agile teams is distributed, providing *subtle control and direction* to the team ... in contrast to *centralized management* in traditional [project] teams” (italics ours).

Speaking about the popularity of different agile methods, Scrum and its derivatives are among the most popular (Dolezel et al., 2019). While Scrum can be classified as a project management framework (Schwaber, 2004), it is important to clarify certain substantial differences between the concept of software project management exemplified above and Scrum. Notably, practitioner literature covering Scrum argues for a new type of leadership role – the Scrum Master (McKenna, 2016):

*A Scrum Master is the servant leader of the Scrum team. ... [But, note that the] Scrum Master is not a development manager with a cool, new name. In Scrum, the team is supposed to be self-managed. In other words, they manage themselves.*

According to the presented perspective, Scrum teams (and more broadly agile teams) are enacted by invoking a different form of authority than in the previous case. Our typology labels such a democratic type of authority as *ad-hoc authority*. As such, this authority would be based on entirely different

mechanisms than in the case of rational-legal authority. Namely, it may be the charisma of natural born leaders (be it or not the Scrum Master) or functional competencies of development team members. Given the broad spectrum of the roles involved, and the reliance on ego-less principles, we see self-organizing Scrum teams as *Generalist Democracies*.

## Communication, Coordination and Integration

A number of CCI mechanisms are relevant for self-organizing agile teams. For example in Scrum, a set of ceremonies is defined to facilitate coordination among team members (Schwaber, 2004). Aside from that, team members may coordinate themselves and others through informal communication means. Therefore, it is virtually impossible to put a strict borderline between the concepts of communication and coordination in agile teams. Importantly, communication and personal interactions are key, so that individual team members “can no longer sit in their cubicles and wait to be told exactly what to do” (Cohn, 2010). Naturally, large-scale software development that is based on agile principles brings its own set of additional CCI challenges (Bick et al., 2018).

Regarding integration, agile teams achieve unity, which is the central attribute of integration, primarily through the mechanism of cross-functionality. As Cohn puts it: “the team needs to include everyone necessary to go from idea to implementation” (Cohn, 2010). In other words, the team needs to be integrated, being not dependent on other teams in terms of basic development activities. Another aspect of cross-functionality is put forward by Schwaber: “in situations where everyone is chipping in to build the functionality, you don’t have to be a tester to test, or a designer to design” (Schwaber, 2004). Nevertheless, this comes with a price: “When you ask a team to be cross-functional, you are asking them to set aside some of their expertise to become more versatile” (McKenna, 2016). Both these theses seem to be in quite a stark contrast with the view of traditional project teams, representing a total of expertise acquired through involving various SE disciplines/professions.

## Independent Test Team as a Specialist Autocracy

The past few decades have seen establishing of certain software engineering professions as fully or partly autonomous. Software testing is an example of such a progress, resulting in many test professionals’ believing that a certain level of autonomy in software testing is desirable. For example, the Test Maturity Model Integration (TMMi; TMMi Foundation, 2012) strongly promotes the idea of centralizing the test expertise within the boundaries of a formal organizational entity. As the TMMi convincingly puts it: “Establishing a test organization implies a commitment to better testing and higher-quality software” (TMMi Foundation, 2012). Despite the fact that the proponents of ASDMs call for removing the tester-developer gap by including software testers into self-organized agile teams (Cohn, 2010), independent test teams are still common in many enterprises (TMMi Foundation, 2012).

While some test teams may be very flat, practitioner literature mostly argues that larger test teams are typically organized in a hierarchical manner (J. A. Jones et al., 2009).

*Industry specialists often recommend a more hierarchical structure for the testing team. At the top of the hierarchy is a test manager. (In some organizations there may also be a test director who is the managerial head of the test organization.) A typical testing team assigned to a project will consist of 3–4 or more test specialists with a test team leader at the head of each team.*

In test organizations and test teams respectively, test directors, test managers, and test team leaders are usually appointed based on their previous experience within the field. Therefore, the entity is ruled by specialists, hence independent testing teams can be called *Specialist Autocracies*.

## Communication, Coordination and Integration

In such an environment, designing effective CCI mechanisms may be tricky. In principle, there are two commonly used alternatives. First, work can be carried out within the test organizations (i.e. a silo-ed functional department), which “throw-over-the-wall” their outputs when they are “done” with their part. Arguably, this configuration is quite inflexible. Interestingly, while it brings significant challenges related to effective CCI, it is still being used (J. A. Jones et al., 2009), expectedly due to the alignment with plan-driven and functionally-oriented software development methods (Borges et al., 2012), or due to estimated cost benefits (J. A. Jones et al., 2009).

Second, project teams may be assembled by “borrowing” testers from their functional departments led by a different line manager (e.g. test director/manager) (Daft et al., 2010). The model in which specialists of certain kind formally share an organizational unit while factually working elsewhere is titled “Matrix organization” (Kolodny, 1979). (Imagine the core functional unit of software workers as their *home harbor*, while most of the *real work* happens during frequent *trips* to project teams.)

Broadly speaking, this is a frequent model. For example, C. Jones (2010) argues that

*In the software world, matrix organizations are found most often in large companies that employ between perhaps 1,000 and 50,000 total software personnel. These large companies tend to have scores of specialized skills and hundreds of projects going on at the same time.*

In this model, one important aspect strongly influences the CCI mechanism. Namely, the line managers still exercise certain level of power and authority over the workers. As there are a number of possible configurations (e.g. weak matrix, balanced matrix, strong matrix), distribution of authority between the line manager and the project manager generally differs (PMI, 2013).

An additional problem factor may arise when different line managers have different work-related goals driven by the corporate hierarchy (Palm & Lindahl, 2015). Then, looking at the project team as an aggregate of people, the goals inherited from their managers may be significantly misaligned. Also, further misalignments between the project goals and line-unit goals may occur (Kolodny, 1979). In sum, despite matrix organization being used for a number of decades, it is not entirely clear up to date how to effectively resolve potential organizational conflicts between the line manager and the project manager in real-world organizations.

## Developer’s Community of Practice as a Specialist Democracy

Communities of practice (CoP) is a concept that have attracted a particular attention of SE researchers (Paasivaara & Lassenius, 2014). In agile environments, communities of practice are established to help professionals working in cross-functional teams, in which they have limited possibility to deepen their disciplinary knowledge (McKenna, 2016). (This situation can be directly contrasted with the functional organization described in Section 3.4.) Alternatively, the communities are deployed to help solving certain coordination challenges (Bick et al., 2018).

## ***A Framework for Analyzing Structural Mechanisms Deployed to Support Traditional and Agile Methods***

In this section, we examine the situation when communities of practice arise as discipline-centered – i.e. there is one community for Scrum Masters, another for developers working with a specific technology, and yet other for testers (McKenna, 2016). While some of these communities may be officially sanctioned and formalized, the very essence of the concept as used within the SE domain seems to emphasize informal relations and personal authority stemming from individual expertise. In other words, communities of practice have a different purpose than “getting the work done”; they are primarily about personal development, passion and enjoyment (Gilley & Kerno, 2010). Yet, in most organizations they need to be cultivated by managers to survive in the long-run, because voluntarism may clash with formal authority (Wenger et al., 2002).

Stressing the knowledge management aspect of CoPs, Cohn (2010) argues that as communities span development teams they are

*a primary mechanism for spreading good ideas among teams and for ensuring desirable levels of consistency or commonality across a set of development teams.*

As sharing “good ideas” is often done primarily among members of the same profession, such communities may be labeled *Specialist Democracies*. That means, the community mostly consists of the *equal* members of a particular profession, so that it promotes a peer-to-peer organizational philosophy (Wenger et al., 2002). Nonetheless, some communities may be supported by competency leads or coaches (Paasivaara & Lassenius, 2014).

### **Communication, Coordination and Integration**

As suggested above, communities may be used as a home-base for nurturing diverse CCI mechanisms. When communities are used as a voluntary, personal-development platform, they typically stimulate communication among workers with the same interest. Community meetings may be quite relaxed, having “a loose agenda with discussion items collected by the participants to wiki pages” (Paasivaara & Lassenius, 2014). This suggests highly informal CCI mechanisms. By contrast, when communities have a more formalized role, typically in the coordination of large-scale software development, the CCI mechanism may be more rigorous.

## **DISCUSSION**

Below, we firstly summarize the key theses of this chapter (Section 5.1). Then, in Section 5.2, we sketch how the proposed typology can be used for theorizing. Finally, in Section 5.3, we suggest that the typology highlights an additional, *bigger* idea – that of workplace democratization.

### **Short Summary**

This chapter dealt with selected structural aspects of the software development organization in light of the shift from plan-driven to agile software development methods (ASDMs). To explore this issue, we introduced the novel classification schema of traditional and agile organizational structures, and briefly analyzed four concrete organizational entities which have been commonly implemented by the software



industry in relation to software development activities. Our overall goal was to demonstrate that such entities represent an important element of the organizational life of software professionals, who rely on various Communication, Coordination and Integration (CCI) mechanisms in order to get their work done.

Further, we postulated that the type of authority and the level of disciplinary (functional) orientation are the important discriminators when examining the current forms of organizational entities in software development companies. To reiterate, authority can be either rational-legal or ad-hoc, and the entities can be either disciplinary-oriented or multidisciplinary. Our analysis affirmed that the software industry seems to be on a continuing move from the rational-legal/disciplinary organizational configurations towards their ad-hoc/multidisciplinary cousins, as exemplified by the changing nature of organizational structures selected as the examples.

In the next section, we aim to briefly demonstrate how the typology can be used as an analytical vehicle for researchers and practitioners. Aside from demonstrating the usefulness of the typology, we also aim to formulate a rudiment of an emerging theory (Reynolds, 2016) of agile transformations.

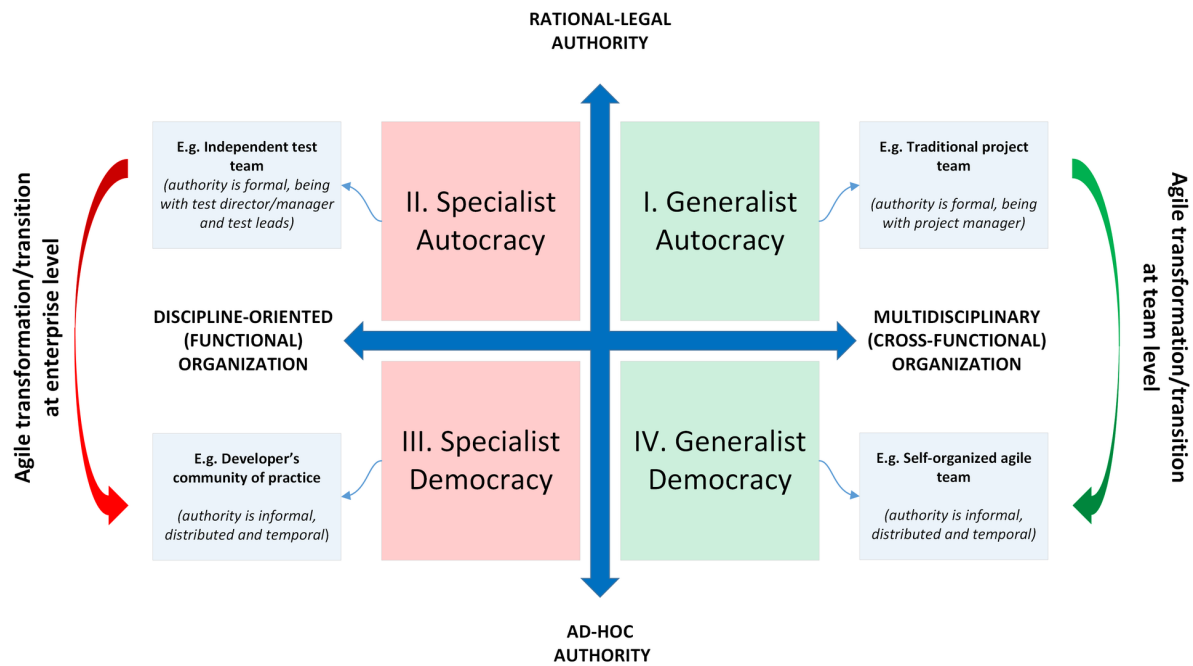
### **Using the Typology to Understand and Predict the Process of Agile Transformation/Transition**

To answer the question posed in Introduction, i.e. *What is the impact of the shift from plan-driven to agile methods on the selected structural aspects of the software development organization?*, we now put our findings into the context of a hypothetical agile transformation and adoption. These are two common terms to denote the necessity to change organizational culture and software development practices respectively (Diebold et al., 2019).

Using the proposed typology, we postulate that agile transformations and adoptions will go along the vertical line, following the top-to-bottom direction (i.e. Q.I -> Q.IV portrayed in green and Q.II -> Q.III portrayed in red, see Figure 2). So, in the course of an agile transformation and transition, *autocratic* organizational entities are to become *democratic* organizational entities. To this end, two transformation efforts will typically occur. First, at the individual team level, *Generalist Autocracies* (i.e. traditional project teams, Q.I) will be transformed to *Generalist Democracies* (i.e. self-organized agile teams, Q.IV). Second, at the enterprise level – i.e. with an expected impact on the existing organizational structures, *Specialist autocracies* (i.e. functional departments/divisions, Q.II), which previously served as specialist home-bases for knowledge exchanges, will be dissolved due to not being a fit for purpose in the agile world. Then, *Specialist Democracies* (Q.III) will become the key mechanism for coordination and knowledge sharing among software professionals. Based on the described scenario, we assume that the latter type of transition will be triggered by the former type of transition.

Nevertheless, our typology clearly counts only with ideal instances of the four archetypes, being primarily a theoretical tool (Reynolds, 2016). Therefore, it does not explicitly cover a real-world possibility such as an enterprise is to be transformed from traditional project management to Scrum, while the fact that the underlying management philosophy and culture must be changed as well is being ignored (Taylor, 2016). In principle, the result of such a situation could be a hybrid team entity, which contains fragments of both formal and ad-hoc authority. Then, it is for a careful consideration whether the resulting organizational entity belongs to either of the quadrants (Q.I and Q.IV), or rather overlaps them both. In that sense, it is important to reiterate that real-world reality is complex. Even if a company pursues a “faithful” agile transformation, ASDMs are rarely implemented in their “pure” form (Dolezel et al.,

Figure 2. Agile transformation/transition in light of the typology



2019). As a result, existing management hierarchies may be preserved, which may result in autonomy and self-organization being merely an illusion (Hodgson & Briand, 2013).

Elaborating on these issues, we believe that a number of academic disciplines have a lot to offer to software engineering at present days. In the following section we thus briefly turn to the phenomenon we term “democratization in the workplace”.

### Democratization in the Workplace – Insights from other Disciplines

By “democratization” we mean the fundamental change we have conceptualized as a move from the autocratic to democratic archetypes (Figure 2). In general, our perspective has been so far limited to organizational entities which are directly involved in software development. Along this line of thought, the original ideas about agility were formulated by a dissenting group of organizational contrarians (Hohl et al., 2018) . Then, after two decades, the impact of the agile philosophy crossed the borderline of computing. This has become quite obvious recently, looking at the current headlines such as “What to expect from Agile?”, which can today be found in respected business administration magazines (Birkinshaw, 2018). Importantly, we suggest that the “democratization process” is a phenomenon catching a significant interest not only in the domain of software engineering. An additional insight from different disciplines can portray a more complete picture. We therefore aim to conceptually connect our observations regarding organizational entities analyzed in this chapter to a global, on-going shift in the world of work.

Doing so, we are able to articulate that the idea of agility resonates well with certain concepts within that domain. Focusing only on a few selected concepts, we firstly mention the catchy ideas of *holacracy*

and *empowering organization*, being used by well-known companies such as Zappos and W. L. Gore & Associates respectively. (A particularly convincing story of a working experience in the “lattice” or flat organization, as deployed in the latter company, was given by Pacanowsky (1988) three decades ago). Both these ideas resonate with the seminal thoughts of Laloux presented in his work titled *Reinventing Organizations* (Laloux, 2014), where the concept of *teal organizations*, being “less driven by ego” and more by democratic principles, was introduced. Next, the inception of Lean startup has also attracted a significant scholarly and practitioner interest (Blank, 2013). Finally, Eckstein has recently introduced the concept of *sociocracy* to software engineering researchers by defining it as “an organizational model for scaling agile development” that “do not hinder, but foster agility” (Eckstein, 2016). It seems that both academia and industry currently exhibit quite a high level of enthusiasm with regards to these models.

However, despite all their promising aspects, the “SALT organizational models” (Sociocracy, Agile, Lean startup and Teal) seem to have a number of downsides too. For example, some Human Resources professionals suggest to remain cautious when it comes to the discussion about the applicability of these models at large scale (Østergaard, 2020).

*The SALT group of organizational models ... is really interesting and seductive, and clearly a great codification of how to embrace organizational democracy, personal engagement and business adaptability. They work, and cases of successful implementation keep popping up. And, yes, those models do have a scaling issue. They work very well in small teams or clusters of five, 25, or maybe even 80 employees, but applying that thinking to an organization of thousands of employees is overwhelming.*

The present chapter does not give a clear-cut answer on whether the above opinion holds true. In any case, we agree that scalability of every organizational model is a crucial aspect deserving a serious attention.

## **CONCLUSION**

In this chapter, we proposed an approach to analyzing key structural aspects associated with entities commonly used for the organization of software development activities. Furthermore, we highlighted the importance of selected formal and semi-formal organizational arrangements, being a crucial element of software development enterprises nowadays. Broadly speaking, structures are to establish order, which many people naturally look for, and therefore it is important to understand how structures are created, maintained and collapsed. In that sense, we offered a thesis that the software development workplaces seem to presently be “democratizing” themselves through redefining their structural portfolio.

While we believe we have added an important piece to the software engineering body of knowledge, we need to clarify that our work has three important limitations. First, it does not cover the problems of large-scale ASDMs in an adequate level of detail. Still, the presented typology can certainly be used to categorize emerging entities such as squads, guilds, chapters (Smite et al., 2019), and agile centers of excellence (Knaster & Leffingwell, 2019) etc., which are increasingly popular. Second, as we briefly noted in Section 3.2, certain structures support professional development of specialists better than others. While we did not intend to cover aspects such as personnel motivation, we point out to the fact that a future exploration of such factors against the backdrop of structures is of foremost importance. Third, our discussion of the impact with regards to hybrid software development methods (Kuhmann

et al., 2019) was very limited. Given that the representation of hybrid software development methods is significant (Marinho et al., 2019), it would therefore be very reasonable to explore questions such as how hybrid models of authority could look like. Together, these are important stimuli for future work. Overall, we expect to see a growing number of research initiatives focused on structural issues of software development soon, because creating effective organizational structures is an issue of central importance in large-scale ASDMs (Paasivaara & Lassenius, 2014; Smite et al., 2019).

## **ACKNOWLEDGMENT**

This research was supported by the University of Economics, Prague [internal grant number F4/23/2019].

## **REFERENCES**

- Balaji, S., & Brown, C. V. (2014). Lateral coordination mechanisms and the moderating role of arrangement characteristics in information systems development outsourcing. *Information Systems Research*, 25(4), 747–760. doi:10.1287/isre.2014.0556
- Bannerman, P. L. (2009). Software development governance: A meta-management perspective. *ICSE Workshop on Software Development Governance (SDG)*, 3–8.
- Begel, A., Nagappan, N., Poile, C., & Layman, L. (2009). Coordination in large-scale software teams. *ICSE Workshop on Cooperative and Human Aspects on Software Engineering (CHASE)*, 1–7.
- Bick, S., Spohrer, K., Hoda, R., Scheerer, A., & Heinzl, A. (2018). Coordination Challenges in Large-Scale Software Development: A Case Study of Planning Misalignment in Hybrid Settings. *IEEE Transactions on Software Engineering*, 44(10), 932–950. doi:10.1109/TSE.2017.2730870
- Birkinshaw, J. (2018). What to Expect From Agile. *MIT Sloan Management Review*, 59(2), 39–42.
- Bjørnson, F. O., Wijnmaalen, J., & Stettina, C. J. (2018). Inter-team Coordination in Large-Scale Agile Development: A Case Study of Three Enabling Mechanisms. *XP*, 216–231.
- Blank, S. (2013). Why the Lean Start-Up Changes Everything. *Harvard Business Review*, 91(5), 64–68.
- Borges, P., Monteiro, P., & MacHado, R. J. (2012). Mapping RUP roles to small software development teams. *Software Quality Days*, 59–70.
- Brown, A., & Grant, G. (2005). Framing the frameworks: A review of IT governance research. *Communications of the Association for Information Systems*, 15, 696–712. doi:10.17705/1CAIS.01538
- Clerc, V., Lago, P., & Van Vliet, H. (2011). Architectural knowledge management practices in agile global software development. *6th IEEE International Conference on Global Software Engineering Workshops*, 1–8. 10.1109/ICGSE-W.2011.17
- Cohn, M. (2010). *Succeeding with Agile*. Addison Wesley.

- Cornelissen, J. (2017). Editor's comments: Developing propositions, a process model, or a typology? Addressing the challenges of writing theory without a boilerplate. *Academy of Management Review*, 42(1), 1–9. doi:10.5465/amr.2016.0196
- Daft, R. L., Murphy, J., & Willmott, H. (2010). *Organization Theory and Design*. Cengage Learning.
- Denning, P. J. (2018). The profession of IT: The computing profession. *Communications of the ACM*, 61(3), 33–35. doi:10.1145/3182108
- Diebold, P., Theobald, S., Wahl, J., & Rausch, Y. (2019). Stepwise transition to agile: From three agile practices to Kanban adaptation. *Journal of Software: Evolution and Process*, 31(5), e2167.
- Dolezel, M., Buchalceva, A., & Mencik, M. (2019). The State of Agile Software Development in the Czech Republic: Preliminary Findings Indicate the Dominance of “Abridged” Scrum. *International Conference on Research and Practical Issues of Enterprise Information Systems*, 43–54. 10.1007/978-3-030-37632-1\_4
- Dolezel, M., & Felderer, M. (2018). Organizational Patterns Between Developers and Testers: Investigating Testers' Autonomy and Role Identity. *Proceedings of the 20th International Conference on Enterprise Information Systems*, 2, 336–344. 10.5220/0006783703360344
- Ebert, C., & Paasivaara, M. (2017). Scaling Agile. *IEEE Software*, 34(6), 98–103. doi:10.1109/MS.2017.4121226
- Eckstein, J. (2016). *Sociocracy - An organization model for large-scale agile development*. XP Workshops.
- Fontana, R. M., Fontana, I. M., Da Rosa Garbuio, P. A., Reinehr, S., & Malucelli, A. (2014). Processes versus people: How should agile software development maturity be defined? *Journal of Systems and Software*, 97, 140–155. doi:10.1016/j.jss.2014.07.030
- Fowler, M., & Highsmith, J. (2001). The Agile Manifesto. *Dr. Dobb's Journal*. <http://www.drdoobs.com/open-source/the-agile-manifesto/184414755>
- Gilley, A., & Kerno, S. J. Jr. (2010). Groups, teams, and communities of practice: A comparison. *Advances in Developing Human Resources*, 12(1), 46–60. doi:10.1177/1523422310365312
- Graham, J. W. (1991). Servant-leadership in organizations: Inspirational and moral. *The Leadership Quarterly*, 2(2), 105–119. doi:10.1016/1048-9843(91)90025-W
- Hoda, R., Noble, J., & Marshall, S. (2010). Organizing Self-Organizing Teams. *International Conference on Software Engineering*, 285–294.
- Hodgson, D. (2002). Disciplining the professional: The case of Project Management. *Journal of Management Studies*, 39(6), 803–821. doi:10.1111/1467-6486.00312
- Hodgson, D., & Briand, L. (2013). Controlling the uncontrollable: “Agile” teams and illusions of autonomy in creative work. *Work, Employment and Society*, 27(2), 308–325. doi:10.1177/0950017012460315
- Hohl, P., Klünder, J., van Bennekum, A., Lockard, R., Gifford, J., Münch, J., Stupperich, M., & Schneider, K. (2018). Back to the future: Origins and directions of the “Agile Manifesto” – views of the originators. *Journal of Software Engineering Research and Development*, 6(1), 1–27. doi:10.118640411-018-0059-z

## ***A Framework for Analyzing Structural Mechanisms Deployed to Support Traditional and Agile Methods***

- Humphrey, W. S. (1998). Why don't they practice what we preach? *Annals of Software Engineering*, 6(1/4), 201–222. doi:10.1023/A:1018997029222
- IEEE. (2014). *Guide to the Software Engineering Body of Knowledge - v3.0*. IEEE.
- Jesse, N. (2018). Organizational Evolution - How Digital Disruption Enforces Organizational Agility. *IFAC PapersOnLine*, 51(30), 486–491. doi:10.1016/j.ifacol.2018.11.310
- Jones, C. (2010). *Software Engineering Best Practices*. McGraw Hill.
- Jones, J. A., Grechanik, M., & Van der Hoek, A. (2009). Enabling and enhancing collaborations between software development organizations and independent test agencies. *ICSE Workshop on Cooperative and Human Aspects on Software Engineering (CHASE)*, 56–59. 10.1109/CHASE.2009.5071411
- Kalenda, M., Hyna, P., & Rossi, B. (2018). Scaling agile in large organizations: Practices, challenges, and success factors. *Journal of Software: Evolution and Process*, 30(10), e1954.
- Kalliamvakou, E., Bird, C., Zimmermann, T., Begel, A., DeLine, R., & German, D. M. (2017). What Makes a Great Manager of Software Engineers? *IEEE Transactions on Software Engineering*, 45(1), 87–106. doi:10.1109/TSE.2017.2768368
- Kautz, K., & Zumpe, S. (2008). Just Enough Structure at the Edge of Chaos: Agile Information System Development in Practice. *Agile Processes in Software Engineering and Extreme Programming*, 137–146.
- Knaster, R., & Leffingwell, D. (2019). *SAFe Distilled*. Scaled Agile, Inc.
- Kneuper, R. (2017). Sixty years of software development life cycle models. *IEEE Annals of the History of Computing*, 39(3), 41–54.
- Kolodny, H. F. (1979). Evolution to a Matrix Organization. *Academy of Management Review*, 4(4), 543–553. doi:10.5465/amr.1979.4498362
- Kroll, P., & Kruchten, P. (2003). *Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*. Addison Wesley.
- Kuhrmann, M., Diebold, P., Munch, J., Tell, P., Trektene, K., Mc Caffery, F., Vahid, G., Felderer, M., Linssen, O., Hanser, E., & Prause, C. (2019). Hybrid Software Development Approaches in Practice: A European Perspective. *IEEE Software*, 36(4), 20–31. doi:10.1109/MS.2018.110161245
- Laloux, F. (2014). *Reinventing Organizations: A Guide to Creating Organizations Inspired by the Next Stage in Human Consciousness*. Nelson Parker.
- Laurin, K., & Kay, A. C. (2017). The Motivational Underpinnings of Belief in God. In *Advances in Experimental Social Psychology* (Vol. 56, pp. 201–257). Academic Press. doi:10.1016/bs.aesp.2017.02.004
- Lawrence, P. R., & Lorsch, J. W. (1967). Differentiation and Integration in Complex Organizations. *Administrative Science Quarterly*, 12(1), 1–47. doi:10.2307/2391211
- Lillrank, P. (1995). The Transfer of Management Innovations from Japan. *Organization Studies*, 16(6), 971–989. doi:10.1177/017084069501600603

**A Framework for Analyzing Structural Mechanisms Deployed to Support Traditional and Agile Methods**

- Malone, T. W., & Crowston, K. (1994). The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, 26(1), 87–118. doi:10.1145/174666.174668
- Marinho, M., Noll, J., Richardson, I., & Beecham, S. (2019). Plan-Driven Approaches Are Alive and Kicking in Agile Global Software Development. *International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 10.1109/ESEM.2019.8870168
- McKenna, D. (2016). *The Art of Scrum*. CA Technologies. doi:10.1007/978-1-4842-2277-5
- Miller, K. W., & Voas, J. (2008). Computer Scientist, Software Engineer, or IT Professional: Which Do You Think You Are? *IT Professional*, 10(4), 4–6. doi:10.1109/MITP.2008.64
- Mishra, D., & Mishra, A. (2008). Workspace environment for collaboration in small software development organization. *International Conference on Cooperative Design, Visualization and Engineering*, 196–203. 10.1007/978-3-540-88011-0\_27
- Morgan, G. (2006). *Images of Organization* (3rd ed.). Sage.
- Motingoe, M., & Langerman, J. J. (2019). New Organisational Models That Break Silos in Organisations to Enable Software Delivery Flow. *International Conference on System Science and Engineering (ICSSE)*, 341–348. 10.1109/ICSSE.2019.8823257
- Nalebuff, B., & Brandenburger, A. (1997). Self-organization: The irresistible future of organizing. *Strategy and Leadership*, 25(6), 28–33. doi:10.1108/eb054655
- Ngwenyama, O., & Nielsen, P. A. (2013). Using organizational influence processes to overcome IS implementation barriers: Lessons from a longitudinal case study of SPI implementation. *European Journal of Information Systems*, 23(2), 205–222. doi:10.1057/ejis.2012.56
- Noordeloos, R., Manteli, C., & Van Vliet, H. (2012). From RUP to Scrum in global software development: A case study. *7th International Conference on Global Software Engineering (ICGSE)*, 31–40. 10.1109/ICGSE.2012.11
- Østergaard, E. K. (2020, Jan.). Organisations of the future. *HR Future*, 14–15.
- Paasivaara, M., & Lassenius, C. (2014). Communities of practice in a large distributed agile software development organization - Case Ericsson. *Information and Software Technology*, 56(12), 1556–1577. doi:10.1016/j.infsof.2014.06.008
- Pacanowsky, M. (1988). Communication in the Empowering Organization. In J. A. Anderson (Ed.), *Communication yearbook 11* (pp. 356–379). Sage.
- Palm, K., & Lindahl, M. (2015). A project as a workplace - Observations from project managers in four R&D and project-intensive companies. *International Journal of Project Management*, 33(4), 828–838. doi:10.1016/j.ijproman.2014.10.002
- PMI. (2013). *A Guide to the Project Management Body of Knowledge* (5th ed.).
- Pugh, D. S., Hickson, D. J., & Hinings, C. R. (1969). An Empirical Taxonomy of Structures of Work Organizations. *Administrative Science Quarterly*, 14(1), 115–126. doi:10.2307/2391367

## ***A Framework for Analyzing Structural Mechanisms Deployed to Support Traditional and Agile Methods***

- Quinnan, R. E. (2010). The management of software engineering, Part V: Software engineering management practices. *IBM Systems Journal*, *19*(4), 466–477. doi:10.1147/j.194.0466
- Ralph, P. (2018). Toward Methodological Guidelines for Process Theories and Taxonomies in Software Engineering. *IEEE Transactions on Software Engineering*, *45*(7), 712–735. doi:10.1109/TSE.2018.2796554
- Reynolds, P. D. (2016). *Primer in Theory Construction*. Routledge.
- Schatz, B., & Abdelshafi, I. (2005). Primavera gets Agile: A successful transition to Agile development. *IEEE Software*, *22*(3), 36–42. doi:10.1109/MS.2005.74
- Schwaber, K. (2004). *Agile Project Management with Scrum*. Microsoft Press.
- Schwaber K. (2013). *unSAFE at any speed*. <https://kenschwaber.wordpress.com/2013/08/06/unsafe-at-any-speed/>
- Shepard, J. (2013). *Sociology* (11th ed.). Wadsworth, Cengage Learning.
- Smite, D., Moe, N. B., Levinta, G., & Floryan, M. (2019). Spotify Guilds: How to Succeed With Knowledge Sharing in Large-Scale Agile Organizations. *IEEE Software*, *36*(2), 51–57. doi:10.1109/MS.2018.2886178
- Šmite, D., Moe, N. B., Šāblis, A., & Wohlin, C. (2017). Software teams and their knowledge networks in large-scale software development. *Information and Software Technology*, *86*, 71–86. doi:10.1016/j.infsof.2017.01.003
- Spencer, M. E., & Spencer, M. E. (1970). Weber on Legitimate Norms and Authority. *The British Journal of Sociology*, *21*(2), 123–134. doi:10.2307/588403 PMID:4928951
- Stol, K. J., & Fitzgerald, B. (2015). Theory-oriented software engineering. *Science of Computer Programming*, *101*, 79–98. doi:10.1016/j.scico.2014.11.010
- Taylor, K. J. (2016). Adopting Agile software development: The project manager experience. *Information Technology & People*, *29*(4), 670–687. doi:10.1108/ITP-02-2014-0031
- TMMi Foundation. (2012). *Test Maturity Model Integration R1.2*. <https://www.tmmi.org/tmmi-documents/>
- Vähäniitty, J., & Rautiainen, K. (2005). Towards an Approach for Managing the Development Portfolio in Small Product-Oriented Software Companies. *38th Hawaii International Conference on System Sciences*. 10.1109/HICSS.2005.636
- Wenger, E., McDermott, R., & Snyder, W. M. (2002). *Cultivating Communities of Practice*. Harvard Business School Press.
- Zhang, X., Stafford, T. F., Dhaliwal, J. S., Gillenson, M. L., & Moeller, G. (2014). Sources of conflict between developers and testers in software development. *Information & Management*, *51*(1), 13–26. doi:10.1016/j.im.2013.09.006



## KEY TERMS AND DEFINITIONS

**Ad-Hoc Authority:** A form of authority enacted and exercised without a formal basis.

**Cross-Functionality:** An organizational principle that follows the idea of putting *diverse* competencies and expertise together, expectedly resulting in a synergistic effect.

**Democratization:** A metaphorical term used to characterize the move from traditional to agile organizational structures and management principles (here examined within the context of software development activities).

**Functional Orientation:** An organizational principle that follows the idea of intensifying *alike* competencies in terms of their segregation from competencies of a different nature (e.g. software analysis from software testing).

**Rational-Legal Authority:** A form of authority that stems from a formal position or standing.

**Structure:** An arrangement that is to introduce stability and/or order.

**Typology:** A form of classification schema with possible theoretical implications.

## ENDNOTE

<sup>1</sup> <https://dictionary.cambridge.org/>