Journal of Software Engineering Research and Development, 2022, 9:21, doi: 10.5753/jserd.2021.1992

First step climbing the Stairway to Heaven Model - Results from a Case Study in Industry

Paulo Sérgio dos Santos Júnior [Federal Institute of Education, Science and Technology of Espírito Santo | paulo.junior@ifes.edu.br]
Monalessa Perini Barcellos [Federal University of Espírito Santo | monalessa@inf.ufes.br]
Rodrigo Fernandes Calhau [Federal Institute of Education, Science and Technology of Espírito Santo | calhau@ifes.edu.br]

Abstract

Context: Nowadays, software development organizations have adopted agile practices and data-driven software development aiming at a competitive advantage. Moving from traditional to agile and data-driven software development requires changes in the organization´s culture and structure, which may not be easy. The Stairway to Heaven Model (StH) describes this evolution path in five stages. *Objective*: We aimed to investigate how Systems Theory tools, GUT Matrix, and reference ontologies can help organizations in the first transition of StH, i.e., moving from traditional to agile development. *Method*: We performed a participative case study in a Brazilian organization that develops software in partnership with a European organization. We applied Systems Theory tools (systemic maps and archetypes) to understand the organization and identify undesirable behaviors and their causes. Thus, we used GUT Matrices to decide which ones should be addressed first and we defined strategies to change the undesirable behaviors by implementing agile practices. We also used the conceptualization provided by reference ontologies to share a common understanding of agile and help implement the strategies. *Results*: By understanding the organization, a decision was made to implement a combination of agile and traditional practices. The implemented strategies improved software quality and project time, and cost. Problems due to misunderstanding agile concepts were solved by using reference ontologies, process models, and other diagrams built based on the ontologies conceptualization, allowing the organization to experience agile culture and foresee changes in its business model. *Conclusion*: Systems Theory tools and GUT Matrix aid organizations to move from traditional to agile development by supporting better understanding the organization, finding leverage points of change, and enabling to define strategies aligned to the organization characteristics and priorities. Reference ontologies can be useful to establish a common understanding about agile, enabling teams to be aware of and, thus, more committed to agile practices and concepts. The use of process models and other diagrams can favor learning the conceptualization provided by the ontologies.

Keywords: *Stairway to Heaven, Agile, Systems Theory, GUT Matrix, Ontology*

1 Introduction

Typically, fast-changing and unpredictable market needs, complex and changing customer requirements, and pressures of shorter time-to-market are challenges faced by organizations. To address these challenges, many organizations have started adopting agile development methods with the intention to enhance the organization´s ability to respond to change. In emphasizing flexibility, efficiency and speed, agile practices have led to a paradigm shift in how software is developed (Williams and Cockburn 2003) (Olsson et al. 2012). Different flavors of the agile methods have become the *de facto* way of working in the software industry (Rodriguez et al. 2012). In allowing for more flexible ways of working with an emphasis on customer collaboration and speed of development, agile methods help organizations address many of the problems associated with traditional development (Dybå and Dingsøyr 2008).

The adoption of agile practices has enabled organizations to shorten development cycles and increase customer collaboration. However, this has not been enough. There has been a need to learn from customers also after deployment of the software product. This requires practices that extend agile practices, such as continuous deployment (i.e., the ability to deliver software more frequently to customers and benefit from frequent customer feedback), which enables shorter feedback loops, more frequent customer feedback, and the ability to more accurately validate whether the developed functionalities correspond to customer needs and behaviors (Olsson et al. 2012). Therefore, organizations should evolve from traditional development towards data-driven and continuous software development.

Continuous Software Engineering (CSE) aims to establish a continuous flow between software-related activities, taking into consideration the entire software life cycle. It seeks to transform discrete development practices into more iterative, flexible, and continuous alternatives, keeping the goal of building and delivering quality products according to established time and costs (Fitzgerald and Stol 2017). Therefore, a continuous software engineering approach is based on agile and continuous practices driven by development and customer data.

Considering that organizations struggle with the changes to be made along the path and with the order in which to implement them, Olsson et al. (2012) proposed the Stairway to Heaven Model (StH), which describes the typical successful evolution of an organization from traditional to continuous and customer data-driven development. The model comprises five stages, where the first transition consists in moving from traditional to agile development. This transition requires a careful introduction of agile practices, a shift to small development teams, and a focus on features rather than components.

In this paper, we report the experience of a Brazilian organization (here called Organization A for anonymity reasons) which decided to evolve from traditional to agile, continuous, and data-driven software development. For that, we have followed the StH model (Olsson et al. 2012). We selected this model because it represents in a simple way the

main stages an organization should follow to move from a traditional to a continuous software engineering approach based on data-driven and agile development. Moreover, StH does not prescribe the practices that should be performed at each stage, thus, there is flexibility to define and implement them according to the organization characteristics and priorities. In this paper, our focus is on the first transition of the StH model. Although there is an increasing number of organizations moving from traditional to agile, implementing the changes needed to the first transition prescribed in StH is not trivial because it involves changes not only in the development process, but also in the organization culture. Moreover, there is no "one and right" way to implement agile practices in an organization because each agile practice needs to be tailored to fit the business goals, culture, environment, and other aspects of the organization. Therefore, organizations should find their own way to go through the path from traditional to agile (Karvonen et al. 2015).

Organization A has a particular characteristic that needs to be considered when defining strategies to implement agile practices: the software projects of Organization A are built in partnership with a European organization (here called Organization B). In this partnership, Organization B is responsible for the software requirements specification process, while Organization A is responsible for design, coding, testing, and deployment processes. Furthermore, Organization B is responsible for the communication between Organization A and the project client. Both organizations A and B work in traditional but many times ad hoc manners. This way of working has brought problems, such as budget overloading, teams divided into disciplines (testers, architects, programmers, etc.) causing many intermediary delivery points in the organization and increasing delays between them, and large periods required to deploy new versions of the software products (Williams and Cockburn 2003) (Olsson et al. 2012)(Karvonen et al. 2015).

Organization A was in the first stage of StH and, in order to evolve, the first step was to go towards becoming an agile organization. Two main challenges were faced in this context: (i) how to move from a traditional development culture to an agile culture and (ii) how to implement agile practices in an organization that shares requirement-related activities with another organization and does not have direct access to the project client.

To overcome these challenges, it would be necessary to get to know the organization so that it would be possible to define suitable strategies to implement agile practices. Thus, we employed an approach that combined Systems Theory tools (mainly systemic maps and archetypes) (Meadows 2008) (Sterman 2010), GUT Matrix (Kepner and Tregoe 1981) and reference ontologies (Guizzardi 2007) to identify the path to implement agile practices and get into agile culture based on the organizational characteristics and context.

Systems Theory tools were chosen because they allow understanding how different variables relate to each other in an organizational environment. Thus, by using such tools, it is possible to understand how processes, practices, culture and other factors affect the software development process and produced results. This helps identify aspects that should be addressed in improvement actions. The first and third authors have knowledge of and experience with System Theory and saw an opportunity to apply it in Organization A. GUT Matrix was selected because it helps prioritize actions and was already known by Organization A. Finally, reference ontologies were used because they have been recognized as an important instrument to deal with knowledge-related problems, supporting communication and learning (Guizzardi 2007). The authors have successfully experienced the use of ontologies as knowledge artifacts in different contexts (e.g., (Ruy et al. 2017), (Santos et al. 2019), (Fonseca et al. 2016)). They developed the Scrum Reference Ontology (SRO) (Santos Jr et al. 2021a), which provides knowledge that aids in the understanding of Scrum in a broader Software Engineering context and is suitable for meeting a learning need identified in the study addressed in this paper.

As main results perceived from the experience reported here, we highlight: (i) it was possible to understand the organization behavior, identify behavior patterns and leverage points of change; (ii) strategies were defined to implement agile practices by changing undesirable behaviors and focusing on leverage points, taking the organization characteristics into account; (iii) by implementing the strategies, Organization A improved software quality, project time and cost and started to develop agile culture; (iv) by using the conceptualization provided by reference ontologies, the team learned agile concepts and practices, which is useful to implement strategies aiming at the agile organization; and (v) a process based on Systems Theory to aid organization define strategies to implement agile practices arose from the study.

This work brings contributions to researchers and practitioners. The study can serve as an example for other organizations similar to Organization A and the process resulting from the study can be used by other organizations. Moreover, the way ontologies were used to provide knowledge for the team can inspire others to make the most of this powerful instrument to knowledge structuring, representation and sharing. Furthermore, researchers can reflect and provide advances on the use of Systems Theory to support the definition of strategies in the agile software development context.

This paper extends (Santos Jr et al. 2020) mainly by exploring how reference ontologies were used to help the team learn about Scrum concepts and practices in the case reported here. We also illustrate the roles of Systems Theory tools, GUT Matrix, and reference ontologies in the study, present additional information about organizations A and B and a new systemic model produced during the study. The paper is organized as follows: Section 2 presents the theoretical background; Section 3 discusses related work; Section 4 presents the study planning, execution, and results; Section 5 discusses threats to validity and Section 6 presents our final considerations and future works.

2 Background

2.1 Stairway to Heaven

Traditional software development is organized sequentially, handing over intermediate artifacts (e.g., requirements, designs, code) between different functional groups in the organization. This causes many handover points that lead to problems such as time delays between handovers of different groups and amounts of resources are applied to creating these intermediate artifacts that, to a large extent, are replacements of human-to-human communication (Bosch 2014). In agile software development, the notion of cross-functional, multidisciplinary teams plays a central role. These teams have the

different roles necessary to take a customer´s need all the way to a delivered solution. Moreover, the notion of small, empowered teams, the backlog, and daily stand-up meetings and sprints guide software development through shorter cycles and help bring the software development closer to the client (Bosch 2014).

Moving from traditional to agile development is the first transition prescribed in Stairway to Heaven Model (StH) (Olsson et al. 2012). StH describes the evolution path organizations follow to successfully move from traditional to data-driven software development. It comprises five stages: traditional development, agile organization, continuous integration, continuous deployment, and R&D as an innovation system. In a nutshell, organizations evolving from traditional development start by experimenting with one or a few agile teams. Once these teams are successful, agile practices are adopted by the organization. As the organization starts showing the benefits of working agile, system integration and verification become involved, and the organization adopts continuous integration. Once it runs internally, lead customers often express an interest to receive software functionality earlier than through the normal release cycle. They want continuous deployment of software. The final stage is where the organization collects data from its customers and uses a customer base to run frequent feature experiments to support customer data-driven software development (Olsson et al. 2012).

Many organizations have moved from traditional to agile. There are many ways of doing that and each organization should consider its business goals, culture, environment and other aspects to find the best way to go through the path. In the experience reported in this paper we have used Systems Theory tools, GUT Matrix and reference ontologies, which are briefly introduced in the following.

2.2 System Theory

It has been used in industry and academy to support (re)design of organizations (Sterman 1994) (Meadows 2008) (Sterman 2010). It sees an organization as a system, consisting of elements (e.g., teams, artifacts, policies) and interconnections (e.g., the relation between the development team, the software artifacts it produces and the policies that influence their production) coherently organized in a structure that produces a characteristic set of behaviors, often classified as its function or purpose (e.g., the development team produces a software product aiming to accomplish its function in the organization)(Meadows 2008).

In the Systems Theory literature, there are several tools that support understanding the different elements and behaviors of a system, such as systemic maps and archetypes (Meadows 2008)(Sterman 2010). A systemic map (also known as causal loop diagram) allows representing the dynamics of a system by means of the system borders, relevant variables, their causal relationships, and feedback loops. A positive causal relationship means that two variables change in the same direction (e.g., increasing the number of bad design decisions causes increasing in software defects), while a negative causal relationship means that two variables change in opposite directions (e.g., increase test efficacy causes decreasing in software defects). Feedback loops are mechanisms that change variables of the system. There are two main types: balancing and reinforcing feedback loops. The former is an equilibrant structure in the system and is a source of stability and resistance to change. The latter compounds change in one direction with even more change.

One beneficial effect of using systemic maps is that they help identify archetypes. An archetype is a common structure of the system that produces a characteristic pattern of behavior. For example, the archetype *Shifting the Burden* occurs when a problem symptom is "solved" by applying a symptomatic solution, which diverts attention away from a more fundamental solution (Kim 1994). The archetype *Fix that Fail*, in turn, occurs when an effective fix in the short-term creates side effects, a "fail", for the long-term behavior in the system (Kim 1994). Usually, *Fix that fail* appears inside of another complex archetype as *Shifting the Burden*. Each archetype has a corresponding modeling pattern. Therefore, by analyzing a systemic map is possible to identify archetypes by looking for their modeling patterns. Archetypes and systemic maps can be useful to identify problems and possible leverage points to solve them. Leverage points are points in the system where a small change can lead to a large shift in behavior (Meadows 2008).

2.3 GUT Matrix

It allows to prioritize the resolution of problems, considering that resources are limited to solve them (Kepner and Tregoe 1981). The prioritization is based on: Gravity (G), which describes the impact of the problem on the organization; Urgency (U); referring to how much time is available to address the problem; and Tendency (T), which measures the predisposition of a problem getting worse over time.

2.4 Reference Ontology

Ontologies have been recognized as important instruments to solve knowledge-related problems. An ontology is a formal, explicit specification of a shared conceptualization (Studer et al. 1998). Ontologies can be developed for communication purposes (reference ontologies) or for computational solutions (operational ontologies). A reference ontology is a special kind of conceptual model representing a model of consensus within a community. It is a solution-independent specification with the aim of making a clear and precise description of the domain in reality for the purposes of communication, learning and problem-solving (Baskerville 1997).

In the work described in this paper, we used the Scrum Reference Ontology (SRO) (Santos Jr et al. 2021a), which addresses the main aspects of Scrum, such as, ceremonies, activities, roles, artifacts, and so on. The first and second authors of this paper are also authors of SRO. It is a reference ontology of the Software Engineering Ontology Network (SEON)[1] (Ruy et al. 2016). SEON is an ontology network that contains several integrated ontologies describing various subdomains of the Software Engineering domain (e.g., Software Requirements, Software Process, Software Measurement, Software Quality Assurance, Software Project Management, etc.). By providing a comprehensive and consistent conceptualization of the Software Engineering domain, SEON has been successfully used to solve knowledge-related and interoperability problems in that domain (e.g., (Fonseca et al. 2017)(Ruy et al. 2017)(Bastos et al. 2018) (Santos Jr et al. 2021a)). SRO reuses concepts of other SEON ontologies, namely: Software Process Ontology

[1] SEON specification is available at *http://nemo.inf.ufes.br/en/projects/seon/*

(SPO) (Briguente et al. 2011), Enterprise Ontology (EO) (Ruy et al. 2014) and Reference Software Requirements Ontology (RSRO) (Duarte et al. 2018). By doing that, SRO connects Scrum concepts to more general Software Engineering concepts, enabling it to better understand Scrum in a broader software development context.

SRO was developed by following the SABiO method (Falbo 2014) and it was evaluated through verification and validation activities. Detailed information about SRO, including its conceptual models, descriptions and a study in which we used SRO for semantic interoperability purposes can be found in (Santos Jr et al. 2021a).

3 Related Work

Some works have reported the use of Systems Theory in the agile development context. For example, Vidgen and Wang (2009) proposed a framework based on the Systems Theory that identifies enablers and inhibitors of agility and discusses capabilities that should be present in an agile team. Gregory et al. (2016) discuss challenges to implementing agile and suggest some organizational elements that could be used to do that. Considering the StH context, Karvonen et al. (2015) used BAPO categories (business, architecture, process, and organization) to identify some practices to each StH step. However, they do not discuss how to understand the organization to establish proper strategies to implement them.

Considering scenarios involving more than one organization to produce software, De Sousa et al. (2016) discuss agile transformation in Brazilian public institutions. Different from Organizations A and B, which work together to produce software for the client, Brazilian public institutions hire software organizations to develop software (i.e., the public institution is a client of the hired organization). Moreover, different from the scenario discussed in (De Sousa et al. 2016), in our study, Organization A needed to develop skills, processes, and culture that enabled it to work with multicultural issues, because Organization A, Organization B and clients are in different countries, and have different cultures. None of the aforementioned works use Systems Theory tools, GUT Matrix, and reference ontologies to help organizations to define strategies to agile practices, as we did in our study.

Some works address aspects related to developing software with distributed teams (Jim et al. 2009)(Prikladnicki and Audy 2010). They show that there are many challenges related to communication, knowledge management, coordination and requirement management caused by different location, time and culture. Aiming to address these issues, L'Erario et al. (2020) propose a framework that provides some concepts, a structure and a flow of communication in distributed software projects. Ali and Lai (2018), in turn, focus on requirements communication and propose to use a requirements graph combined with a software requirement specification document to help the stakeholders in the establishment of a better understating of software requirements. Similar to our work, the aforementioned works aim to support organizations in which the software development process is distributed. However, differently from our work, those works consider software development geographically distributed among several development teams of the same organization.

As we previously discussed, our work considered two organizations working as one in the projects, with two teams in different countries, and each team controlling part of the software development process. We propose to use System Theory tools, GUT Matrix and reference ontologies to create strategies that minimize the impact caused by culture, time, and distance and, sometimes, use them as a competitive advantage. We believe that our work can contribute to organizations that work with geographically distributed teams by providing useful knowledge to create tailored strategies. For example, they can be inspired by our strategy to communicate requirements, which uses BDD (Behavior Driven Development) (Wynne et al. 2017) as a protocol to specify, communicate and validate requirements.

4 Case Study, Planning, Execution, and Results

Participative case study was selected as the research method in this study because two researchers acted as consultants in Organization A and ,thus, were participants in the process being observed (Baskerville 1997). Together with other participants, they gathered information to understand the organization and defined strategies to implement agile practices. Thus, the researchers had some control over some intervening variables.

4.1 Study Design

4.1.1 Diagnosis

Organization A is a Brazilian software development organization that works together with a European organization (Organization B) to develop software products for European clients. It has 30 developers organized in teams managed by tech leaders. Organization B elicits requirements with clients and Organization A is in charge of developing the corresponding software. As a consequence of the increasing number of projects and team members, added to the lack of flexible processes, some problems emerged, such as projects late and over budget, increasing in software defects, overloading of the teams due to rework on software artifacts, and communication issues among client, Organization A, and Organization B.

Aiming to minimize these problems, in the first semester of 2019, Organization A decided to implement Scrum practices, but without success. According to the directors, the main difficulties were due to non-direct communication with the client and included: difficulty to define product backlog, select a product owner and carry out Scrum ceremonies that need the client's feedback. Furthermore, they pointed out that agile culture demands knowledge and its clients, business partners and developers were not prepared for it.

Other factors that harmed Scrum implementation were: (i) teams without self-management characteristics, (ii) difficulties in internal communication, (iii) lack of feedback culture, (iv) lack of openness and other Scrum's values. Moreover, Organization A has had a lot of systemic issues, such as: (a) directors have been much focused on operational and technological issues, (b) lack of management professionals, (c) focus on short-term issues instead of long-term ones and, (d) lack of focus on applying strategic and systemic thinking. In addition, the first and third authors noticed that Organization B has had a traditional culture based on linear and non-adaptative processes and methods.

This scenario indicated to us that a particular characteristic and context of the organization had not been considered in the first try to implement agile practices. Therefore, at the beginning of 2020, we proposed to use StH as a reference model to evolve Organization A from traditional to data-driven software development, in a long-term process improvement program. The first step: move from traditional to agile. Considering the peculiar scenario of Organization A, we decided to use Systems Theory to understand the organization in a systemic way. Then, we used GUT Matrix to support prioritization of problems resolution, and reference ontologies to provide common knowledge about agile development.

4.1.2 Planning

The study goal was to analyze the use of Systems Theory tools (particularly systemic maps and archetypes), GUT Matrix, and reference ontologies to help define strategies to implement agile practices when the organization is moving from traditional to agile development. By strategies, we mean actions or plans established to implement agile development. Aligned to this goal, the following research question was defined: *are Systems Theory, GUT Matrix, and reference ontologies useful to define suitable strategies for an organization to move from traditional to agile development?*

The expected outcomes were: (i) a view of important aspects of the organization by means of systemic maps; (ii) prioritization of problems and causes to be addressed; (iii) strategies to address problems and implement agile practices;(iv) artifacts built based on reference ontologies and that help the team to learn agile concepts and practices; (v) a Systems Theory based process to define strategies to move from traditional to agile.

Figure 1 illustrates how Systems Theory tools (particularly systemic maps and archetypes), GUT Matrix, and reference ontologies (blue circles in Figure 1) were used in the study. Reference ontologies and Systems Theory tools were used in the Problem Domain (represented in the yellow region in Figure 1). Ontologies provide the conceptual perspective, while systemic maps and archetypes afford a dynamic perspective. In other words, the former supports understanding the domain itself (agile) by providing structural knowledge, while the latter helps understand the organization in which the problems manifest and how they manifest. GUT Matrix, in turn, was used in the Solution Domain (represented in the green region in Figure 1) as a means to prioritize the problems to be addressed, providing, this way, a problem-solving perspective.

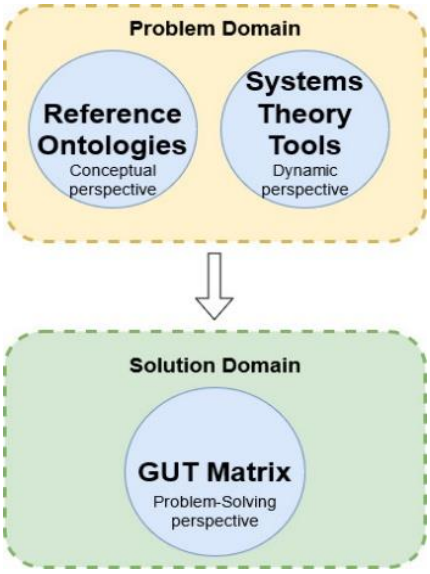

Figure 1. Overview of the approach used in this work.

To be more specific, ontologies were used to provide a common conceptualization to support communication among the organizations and their employees in the software development context. Systemic maps, in turn, aimed to make explicit the variables and relations present in the dynamic of the system between the organizations. Finally, GUT Matrix was used to support the decision-making process that guided the solution process.

The study participants who directly participated in interviews to data collection and results in the evaluation were the two directors (software development director and sales director), one tech leader, and two developers. The first and third authors worked as consultants in Organization A and, thus, also participated in the study. Working together with the other participants, they were responsible for creating systemic maps, GUT matrices, as well as for defining the strategies to be implemented to move from traditional to agile software development. Once these artifacts were created, they were validated with the team. For example, systematic maps were created based on information provided by the team. Then, the team evaluated them in meetings and provided feedback so that we reached the maps shown in the next section. The second author did not interact directly with Organization A. She worked as an external reviewer, evaluating the produced artifacts and helping other authors improve such artifacts.

4.2 Study Execution and Data Collection

4.2.1 Data Collection

Data collection involved interviews, development of systemic maps and GUT Matrix, and definition of strategies to implement agile practices.

A. Initial Interviews

Data collection started with interviews to gather general information about the organization. Six interviews were conducted, four with the directors and two with the developers, and the tech leader. Participants were told to feel free to talk as much as they wanted to. Each interview lasted about 90 minutes. The funnel questions technique was used, i.e., the interview started with general questions (e.g., "*What kind of*

software does the organization develop?", "*How is the software development process?*"), and then went deeper into more specific points of each one (e.g., "*Tell me more about the software test activity*"). The interviews were recorded, transcribed, and validated with each participant.

The interviews with the directors aimed to get information about the following aspects: organizational environment, culture, rules of relationship with partners, future plans, software development process, software development issues, and agile knowledge. Among the information provided by the directors, they pointed out that some problems were caused by misunderstood software requirements or project scope not clearly defined. According to them, Organization B did not describe requirements in a consistent and clear way.

The interviews with the tech leader and developers aimed at understanding software development problems under their perspective and how familiar they were with agile methods and practices. The problems mentioned by the directors were also reported by the tech leader and developers. When asked about team organization, they pointed out that the teams were not self-organized. Contrariwise, tech leaders were responsible for allocating tasks, coordinating team members, establishing deadlines, and monitoring projects. Moreover, the team knowledge of agile was limited.

B. Systemic Maps

Information obtained in the interviews was used to build systemic maps. Figure 2 shows a fragment of one of the developed systemic maps. The elements in blue in the figure form a modeling pattern that reveals the presence of the archetype *Shifting the Burden*.

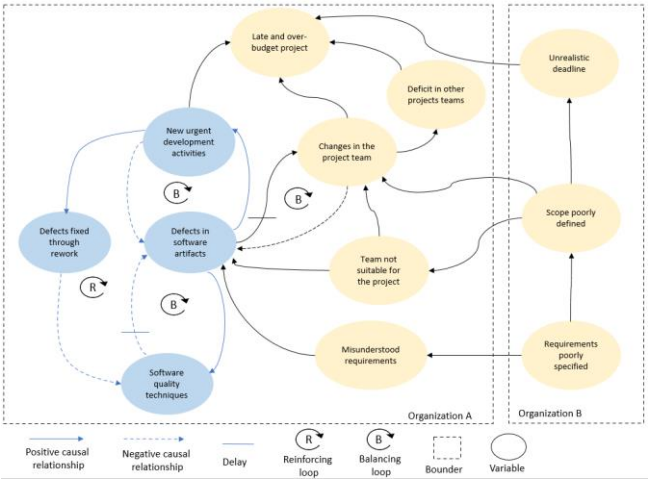

Figure 2. Fragment of systemic map (1).

As previously said, Organization B is responsible for eliciting requirements with the client, specifying and sending them for Organization A to develop the software. The development teams of Organization A often misunderstand requirements that describe the software, component, or functionality to be developed, since Organization B produces *Requirements poorly specified*, neither adopting a technique nor following a pattern to describe them. *Misunderstood requirements* contribute to increasing the number of *Defects in software artifacts*, since design, code, and test are produced based on the requirements informed by Organization B. *Defects in software artifacts* make Organization A mobilize (and often overload) the development team to fix defects by performing *New urgent development activities*, which decrease the number of *Defects in software artifacts*. These urgent activities are performed as fast as possible, aiming not to delay other activities. Thus, they do not properly follow software quality good practices. Moreover, they contribute to increasing the project cost and time (*Late and over-budget project*). *Defects in software artifacts* increase the need of using *Software quality techniques* that, when used, lead to fewer *Defects in software artifacts*. This causal relationship has a delay since the effect of using *Software quality techniques* can take a while to be perceived.

As shown in Figure 2, the archetype *Shifting the Burden* is composed of two balancing feedback loops and one reinforcing feedback loop. The balancing feedback loops (between *New urgent development* activities and *Defects in software artifacts*, and between *Defects in software artifacts* and *Software quality techniques*) mean that the involved variables influence each other in a balanced and stable way (e.g., higher/lower the number of *Defects in software artifacts*, more/less *New urgent development* activities are performed). In the reinforcing feedback loop, *New urgent development activities* are a symptomatic solution that leads to *Defects fixed* through rework, a side effect, because once urgent development activities fix the defects in software artifacts, Organization A feels like the problem was solved. This, in turn, decreases the need for using Software quality techniques, which is a more fundamental solution. As a result, software artifacts continue to be produced with defects, overloading the development team with new urgent development activities. *Shifting the Burden* is a complex behavior structure because the balancing and reinforcing loops move the system (Organization A) in a direction (New urgent development activities) usually other than the one desired (Software quality techniques). *New urgent development* activities contribute to increasing project cost and time (*Project is late and over-budget*) because these activities were not initially planned in the project.

When Organization B does not properly define the project scope (*Scope poorly defined*), Organization A may allocate a Team not suitable for the project, contributing to *Defects in software artifacts* and to *Changes in the project team* during the project. Usually, when the team is changed, the new members need to get knowledge about the project. Moreover, often the new members are more experienced and thus more expensive, which contributes to *Late and over-budget project*. To change the project team, members can be moved from one project to another, causing *Deficit in other project teams*. Furthermore, there is a balancing loop between *Changes in the project team* and *Defects in software artifacts*. The former may cause the latter due to instability inserted into the team. The latter, in turn, contributes to the former because *Defects in software artifacts* may lead to the need to change the team. There is a delay in this relationship because it can take a while to notice defects and the need to change the team. Finally, *Scope poorly defined* causes *Unrealistic deadlines*, which contributes to *Late and over-budget projects*.

Figure 3 illustrates another fragment of the developed systemic maps, showing variables related to different organizational levels. As observed in Figure 3, Organization B is responsible for the *Direct communication with the client*, i.e., Organization A depends on Organization B to obtain information from the client. This causes in Organization A *Lack of contact with the final client*, which contributes to

Low commitment from the team with the project´s goals, since the team is not empowered and loses motivation. This, in turn, leads to *Non-self-organized teams*, because the team members do not have the opportunity to implement values and practices to become self-organized, which keeps the team away from the final client. These three variables create a reinforcing loop that prevents the organization from having more proactive and committed teams.

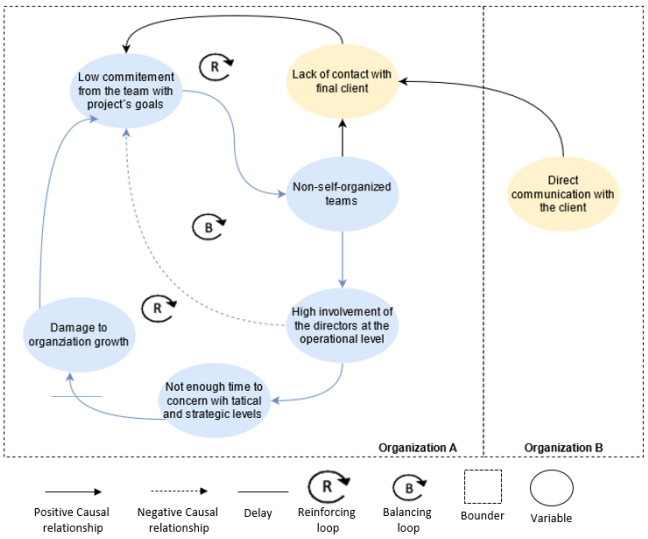

Figure 3. Fragment of the systemic map (2).

Non-self-organized teams and *Low commitment from the team with the project´s goals* contribute to a *High involvement of the directors at the operational level*, because they need to support the teams to solve problems (e.g., *Scope poorly defined*, *Unrealistic deadline* and *Late and over-budget project*, shown in Figure 2). As a consequence, they do *have Not enough time to concern with tactical and strategic levels*, which causes *Damage to organization growth*, because the directors do not have time to plan and implement strategies that allow getting new clients, reducing costs, etc.

The previous paragraph describes an example of the archetype *Fix that fails* that impacted the operational, tactical, and strategical level of Organization A. The archetype *Fix that fails* is composed of a balancing feedback loop that is intended to achieve a particular result or fix a problem, and a reinforcing feedback loop of the unintended consequences. The balancing feedback loop occurs when there is a *High involvement of directors at the operation level* trying to resolve problems of projects because of the *Low commitment from the team with the project´s goals*. The reinforcing feedback loop, in turn, occurs when the directors do have *Not enough time to concern with the tactical, and strategic level* because there is a *High involvement of the directors at the operational level*, resulting in *Damage to organization growth*. This loop affects different organizational levels, from operational to strategic, and hampers organization evolving and growing.

C. GUT Matrix

After getting a comprehensive view of the organization and how it behaves, we reflected on the behaviors on which the strategies should be focused. Thus, we created a GUT Matrix to identify and prioritize behaviors of the system that are not fruitful, i.e., undesirable behaviors. They were identified mainly from the systemic maps. For example, from the fragment depicted in Figure 2 based on the positive causal relationship between *Misunderstood requirements* and *Defects in software artifacts*, the following undesirable behavior was identified: *Software artifacts are developed based on misunderstood requirements*. From the *Shifting the Burden* archetype, we identified: *Software quality techniques are not often applied to build software artifacts*. To complement the information provided by the systemic maps, we used information from the interviews to look for behaviors the literature points out as desirable in organizations moving to agile (e.g., self-organized teams) (Leffingwel 2016).

After identifying the undesirable behaviors, the study participants validated and prioritized them considering the GUT dimensions. Each dimension was evaluated considering values from 1 (*very low*) to 5 (*very high*). 13 undesirable behaviors were identified. Table 1 shows a fragment of the GUT Matrix.

Table 1. Fragment of GUT Matrix.

#	Undesirable Behaviors	G	U	T	GxUxT
UB1	Software artifacts are developed based on misunderstood requirements	5	5	5	125
UB2	Software quality techniques are not often applied to build software artifacts	5	5	4	100
UB3	Projects are late and over budget	5	5	4	100
UB4	Organization has inconsistent knowledge of agile methods	5	5	4	100
UB5	Teams are not self-organized	5	4	4	80

For each undesirable behavior, we analyzed the systemic maps and the interviews and identified its causes. (UB1) *Software artifacts are developed based on misunderstood requirements because* (C1) Requirements are not satisfactorily *described* and (C2) *Poor communication between client and development team*. C1 was identified directly from the systemic map. C2 was based on information about the procedure followed by Organization A to communicate with the client. When there is any doubt about requirements, the contact was made mainly through email or comments on issues in the project management system. Only Organization B has direct contact with the client.

C1 and C2 are also causes of (UB2) *Software quality techniques are not often applied to build software artifacts*, since the lack of well-defined requirements and direct contact with the client impact verification and validation activities. Moreover, there is a (C3) *Lack of clear and objective criteria to evaluate results* and (C4) *Large deliverables*, which make it difficult to evaluate results.

As it can be noticed in Figure 1, *Projects are late and over budget* (UB3) mainly because C1 and (C5) *Unstable scope and deadline*. Moreover (C6) *Unsuitable team allocation* and C4 also affect projects cost and time. The former because low productivity impacts on project time and, thus, cost. The latter is because it is difficult to estimate large projects.

Regarding (UB4) *Organization has inconsistent knowledge of agile methods*, some members of the organization had previous experience with agile methods in other companies, others had a previous unsuccessful experience in Organization A and others did not have experienced agile methods. Most of the members were not sure about agile concepts and practices. Therefore, this undesirable behavior

is caused by (C7) *Organization's members had different experiences with agile* and (C8) *Agile concepts and practices are not well-known by the organization*. Finally, *Teams are not self-organized* (UB5) due to the (C9) *Traditional development culture* that produces functional and hierarchical teams. After identifying the causes of undesirable behaviors, the study participants validated them. Table 2 shows the identified causes and respective undesirable behaviors.

Table 2. Causes of undesirable behaviors.

#	Causes	UB1	UB2	UB3	UB4	UB5
C1	Requirements are not satisfactorily described	X	X	X		
C2	Poor communication between client and development team	X	X			
C3	Lack of clear and objective criteria to evaluate results		X			
C4	Large deliverables		X	X		
C5	Unstable scope and deadline		X	X		
C6	Unsuitable team allocation			X		
C7	Organization's members had different experiences with agile				X	
C8	Agile concepts and practices are not well-known by the organization				X	
C9	Traditional development culture					X

D. Strategies

The causes of undesirable behaviors and the prioritization made in the GUT Matrix showed us leverage points of the system, i.e., points that if changed could change the system behavior. Therefore, we defined strategies to help Organization A move towards the second stage of StH by changing leverage points of the system and thus creating new behaviors in the system in that direction. We started by defining strategies to change undesirable behaviors at the top of the GUT Matrix and causes related to more than one undesirable behavior. After we had defined the strategies, we presented them to the team in a meeting and they provided feedback that helped us to make the strategies more suitable for the organization. Next, we present four strategies defined to address the causes presented in Table 2.

Considering Organization A characteristics, mainly its partnership with Organization B, the strategies combined agile and traditional practices. Agile approaches bring the culture of self-organized teams, shorter development cycles, user stories, smaller deliverables, among other notions (Karvonen et al. 2015)(Leffingwel 2016). Traditional approaches were used to complement agile practices. After all, agile methods usually do not detail how to manage some aspects of a software project, such as costs and risks.

The first strategy, the *New procedure to communicate requirements* (S1), consisted in establishing a new procedure to be followed by organizations A and B regarding requirements and communication, aiming to address C1 and C2. Due to business agreements, a big change in Organization B was not possible. For example, we could not change the fact that only Organization B could directly contact the project client. Hence, it was defined that requirements would be sent from Organization B to the project tech leader, who would rewrite the requirements as user stories and validate them with Organization B. By representing requirements as user stories, the project tech leader also needs to represent their acceptance criteria, which aids to address C3. Moreover, to properly define the acceptance criteria, the tech leader needs to obtain detailed information about the requirement, stimulating Organization B to get such information from the client, which indirectly improves communication with the client. Only user stories defined according to the defined template and validated with Organization B follow to the next development activities. We also suggested the use of a template based on BDD (Behavior Driven Development) (Wynne et al. 2017) and Gherkin Syntax (Binamungu et al. 2020), describing business rules, acceptance criteria and scenarios to serve as a protocol to communicate requirements among organizations A, B and the client. It is worth mentioning that we were not allowed to ask Organization B to write the requirements itself by following the new guidelines, because this change was beyond the partnership agreements. In this strategy, we designated Organization B to play the Product Owner role. This way it is not only a business partner, but it represents the client interests and has responsibilities in this context. With this strategy, we also aimed to minimize the symptomatic solution (New urgent development activities) indicated in the Shift the Burden archetype identified in the systemic map. According to Meadows (2008), the most effective strategy for dealing with a Shifting the Burden structure is to employ the symptomatic solution and develop the fundamental solution. Thus, it is possible to resolve the immediate problem, and also work to ensure that it does not return. By improving requirements descriptions and defining clear acceptance criteria, software quality techniques (e.g., verification and validation), which are the fundamental solution identified in the Shifting the Burden, can be properly applied.

Another strategy, *Budget and time globally and locally managed through short development cycles* (S2), focused on changing the undesirable behavior UB3 (Projects are late and over budget). Again, to change that, Organization A depended on changes in Organization B. Therefore, it was established that at the beginning of a project, Organization A and B should agree on the project scope, deadline, budget and involved risks. The project characteristics (e.g., technologies, domain of interest, platform, etc.) should also be clearly established. The project team would not be allocated before this agreement. By properly aligning information about the project between organizations A and B, it would be possible to allocate a development team with skills and maturity suitable for the project. By doing that, C5 and C6 would be minimized. Complementary, it was defined to change the development process as a whole. In the Organization A business model, when a project is contracted by a client, usually there is a cost and time associated to it. This prevented us from using a pure agile development process, where costs are dynamically established. As a strategy to implement tailored agile practices, it was defined: after requirements are validated, the development team (tech leader and developers) selects the requirements to be developed in a short cycle of development (i.e., a sprint), defines tasks and estimates the time and costs related to them. This information is aligned between organizations A and B. This way, Organization B manages time and budget at a project level, while Organization A manages time and budget in the sprint context. Once a week, monitoring meetings are performed to

check time and budget performance. During the sprint, meetings based on the Scrum ceremonies are carried out in a flexible way. For example, if the team informs that there is nothing to report at the day, the daily meeting is not performed. Meetings that depend on the client's feedback should be carried out with Organization B (in the Product Owner role). By breaking the development process into shorter cycles, C4 is addressed, since the product is also decomposed in smaller deliverables. This strategy also contributes to treating C9, as it changes the traditional development culture.

Aiming to change the way teams are organized in Organization A (UB5) and thus address C9, the strategy *Self-organized teams* (S3) was defined to implement Squad and Guild concepts (Leffingwel 2016). A Squad is a team with all skills and tools needed to develop and release a project. It is self--organized and can make decisions about its way of working. For example, a Squad can define the project development timebox (sprint) and how to implement some practices of strategies S1 and S2 (e.g., the use of BDD and how flexible Scrum ceremonies can be in the project). The members are responsible for creating and maintaining one or more projects. A Squad is composed of developers and a tech leader, who is responsible for communicating with Organization B mainly regarding aspects related to budget, time, and requirements. A Guild is a team responsible for defining standards and good practices that will be used for all squads. A Guild is composed of members with expertise in the subject of interest (e.g., a senior programmer can define good programming practices). Its purpose is to record and share good practices among the squads in the organization, aiming at achieving a homogeneous level of quality in the projects.

To address C7 and C8, which cause the organization to have inconsistent knowledge of agile methods (UB4), we defined *Agile common conceptualization* (S4) as a strategy to use reference ontologies to provide a common conceptualization about the Software Engineering domain as a whole, and about the agile development process in particular. We used ontologies from SEON (Ruy et al. 2016) to extract the view relevant to understand agile development. It contains a conceptual model fragment, axioms and textual descriptions that provide an integrated view of agile and traditional development, defining concepts in a clear, objective, and unambiguous way. We suggested the use of SEON because its ontologies have been developed based on the literature and several standards, providing a consensual conceptualization. Moreover, as we discussed in Section 2, we have successfully used it in several interoperability and knowledge-related initiatives. The SEON view used in the study focuses on the Scrum Reference Ontology (SRO) and can be seen in (Santos Jr et al. 2021a). To make it easier for the teams to learn and apply the conceptualization provided by the ontology, the authors created complementary artifacts that combined graphical and textual elements. We show some of the produced artifacts in Section 4.3.2.

Table 3 summarizes the defined strategies, the leverage points (causes) addressed by them, and main agile concepts involved. It is worth noticing that some agile concepts were indirectly addressed. For example, although we did not directly use Product Backlog in S1, the set of requirements agreed with Organization B works as such. Similarly, in S3, when the team selects the requirements to be addressed in a development cycle, we are applying the Sprint Backlog notion. We decided not to use some of the original terms because Organization A had a previous bad experience trying to implement agile practices by following Scrum "by the book", which did not work and provoked resistance to certain practices. Thus, we tried to give some flexibility even to the practices' names, to avoid bad links with the previous experience.

Table 3. Strategies, Causes and Agile Concepts.

#	Strategies	Agile Concepts	Causes
S1	New procedure to communicate requirements	User Story, BDD, Product Owner and Product Backlog	C1, C2, C3
S2	Budget and time globally and locally managed through short development cycles	Sprint, Sprint Backlog, Scrum meetings and Small deliverables	C4, C5, C6, C9
S3	Self-organized teams	Squad and Guild	C9
S4	Agile common conceptualization	Concepts related to agile software development	C7, C8

After defining and validating the strategies with the team, they were executed by the organization in two projects with the supervision of the first and third authors. The first project started and finished during this study. The second project started before the study and was still ongoing at the time we wrote this paper. The new practices started to be used in early February 2020. About four months later, we conducted an interview to obtain feedback. At that point, one of the projects had already been concluded and the other was ongoing.

4.3 Study Analysis, Interpretation and Lessons Learned

In this section, we present results from the interviews that helped us to answer the research question, the resulting Systems Theory-based process that arose from this study and some lessons learned.

4.3.1 Results

To answer the research question, we carried out an interview with the software development director and the tech leader aiming to obtain their perception about the use of Systems Theory tools, GUT Matrix and reference ontologies, as well as to get information about results obtained from the use of the defined strategies.

They were interviewed together in a single section. The director said that, in his opinion, Systems Theory tools provided means to understand how different organizational aspects (e.g., business rules and quality software practices) are interrelated and influence each other, and how these aspects and interrelations produce desirable and undesirable behaviors. For example, he said that "*the systemic maps allowed me to understand how poorly specified requirements can negatively impact different parts of the project and of the organization".*

Moreover, according to him, "*Systems Theory helped create strategies to change undesirable behaviors, since it provided a comprehensive understanding of the organization behavior and supported identifying causes of undesirable behaviors*". For example, by knowing the impacts of poorly specified requirements, "*I perceived the need to implement practices to guarantee the quality of the requirements and that development tasks should only start if the developer truly understood the requirement*". Regarding GUT

Matrix, the director stated that it found it easy to use, and important to prioritize the undesirable behaviors to be changed first. According to him, using these tools "*was easier and clearer when compared to Ishikawa and Pareto diagrams, because systemic maps allow more comprehensive and freer views and GUT Matrix has a simple way of prioritization.*"

Concerning reference ontologies, he reported that they were useful to create a common communication among project stakeholders and business partners, eliminating some misunderstandings not only about agile practices but also about Software Engineering in general. For example, the director said that *"by using the conceptualization provided by the ontology, the team truly understood the "done" concept"*, commonly used in agile projects, in the sense that a software item (e.g., a functionality, a component) is done (i.e., ready to be delivered to the client) only if it met all the acceptance criteria established to the user stories materialized in that software item. The tech leader commented that *"by using the ontology conceptualization, it was clearer the necessary information a requirement description should contain so that it can be properly understood."* An interesting aspect pointed out by the interviewees was that the conceptualization provided by the reference ontologies was used by the development teams as a basis to quality rules in the projects (e.g., when a software item is done) and, also to business rules in new business contracts (e.g., acceptance criteria need to be defined).

The director and tech leader informed that the first project in which the strategies were implemented was considered a successful experience and served as a pilot. In similar projects, Organization A used to be 30% to 50% over time and budget due to spending extra resources on new urgent development activities to fix defects. By adopting the defined strategies, the project delivered a better product (at the moment of the interview, the client did not have reported any defect in the production environment). However, the project was about 15% over budget and time due to changes in the agreed requirements.

This may suggest that strategies S1 and S2 need adjustments. Although they seek to give some agility features to the development process, the project had its scope predefined by Organization B, which established it together with the client and set cost and time considering that scope. As Organization A started to develop the agreed requirements, Organization B noticed that some of them needed to change to better satisfy the client needs. Although the project was late and over budget, the deviation in relation to the agreed cost and time was smaller than in similar projects that did not follow the strategies. The director pointed out that being able to show this difference to Organization B, indicating the causes that contribute to increase or decrease it, was an important result and can even be used to motivate Organization B to be more involved in the changes to improve the software development process as a whole. This would make it possible, for example, to adjust strategies S1 and S2 to make requirements elicitation, cost, and time estimation more flexible.

The tech leader reported that using the strategies reduced misunderstandings in software requirements among the stakeholders and enabled better managing budget and time locally, in short development cycles. Moreover, according to him, in the second project adopting the strategies (ongoing project), the development team spent only 45 hours in new urgent development activities in a total of about 2000 hours of performed development activities. He also highlighted the use of user stories and BDD as an effective way to communicate requirements in this project.

In addition, the interviewees said that the self-organization culture has been developed in the teams and that the use of Squads has been very helpful. The use of Guilds was still in progress. Finally, they commented that, although the proposed strategies were used to address some undesirable behaviors by applying agile practices and concepts, they felt that *"changing the entire traditional culture can be a complex work*", mainly because it requires to change mental models, processes and culture that also involve the organization partners (particularly Organization B) and clients.

Aiming to obtain quantitative data to complement the feedback provided by the software development director and the tech leader and help us identify the effects of the adopted strategies, we collected data from the two projects (one finished and another ongoing) where the strategies were implemented and from other projects that did not use the strategies. Data was extracted from Jira, which is used by Organization A to support part of the software development process. Considering that the strategies were applied in the projects in different moments (the first project adopted the strategies from its beginning to its end, while the second adopted the strategies when it was already ongoing), we decided to analyze them separately.

First, we collected data regarding the tasks performed in the first project and in other 22 projects that did not adopt the strategies and were carried out in the same time-box of our study. The tasks were classified into development tasks, which create new features, and bug-fixing tasks, which fix problems (found by the quality assurance team or by the client) in the developed features. For each project, we calculated the percentage of effort spent on tasks dedicated to developing new features and the percentage spent on tasks performed to fix bugs. Thus, we calculated the median of the obtained values for the 22 projects that did not use the strategies, so that we could compare the resulting value with the project where the strategies were adopted. Table 4 shows the results.

Table 4. Effort spent on development and bug-fixing tasks in different projects.

Task	Project that adopted the strategies	Projects that did not adopt the strategies
Development	97,62%	81,07%
Bug-fixing	2,38%	18,93%

As it can be observed in Table 4, when compared with the other projects developed in the same time-box, the development team from the project that adopted the defined strategies spent more effort on developing new features (97,62%) than fixing problems (2,38%). This corroborates interviewees' perception that the proposed strategies improved product and process quality.

Aiming to verify changes caused by the strategies in the same project, we also collected data from the beginning of the second project (i.e., Jan/2019), until the last month of our study. Our purpose was to compare the effort spent on each type of task before and after applying the strategies. Table 5 presents the obtained values.

Table 5. Effort spent on development and bug-fixing tasks before and after applying the strategies in the project.

Task	Before the strategies	After the strategies
Development	62,15%	88,21%
Bug-fixing	37,85%	11,79%

As it can be noticed, after applying the strategies, there was an increase in the effort spent on developing new features and a reduction in the effort to fix bugs, which is consistent with the interviewees' perception. It is worth noticing that there was more time spent on the project before applying the strategies (about one year) than after that (about four months). This should be considered together with the obtained data (e.g., we do not know if the amount of effort spent on which type of tasks may significantly change over time).

4.3.2 Using Reference Ontologies to learn Scrum

Although reference ontologies are a good way to structure and represent knowledge, it may not be much easy for some people to capture and internalize the conceptualization represented in the ontology. Thus, in the case reported in this paper, we used some complementary artifacts to help in this matter. First, we asked the team which artifacts they were used to. Based on their answers, we decided to use mainly textual descriptions and process models, since the team considered them user-friendly, and they were present in its daily activities. We also used other diagrams to illustrate Scrum concepts and a Kanban board to map Scrum concepts to concepts already familiar to the team.

The SEON extract addressing agile aspects and connecting them to traditional aspects provided the common conceptualization and knowledge about the domain of interest. For example, the ontology makes it explicit that only deliverables (i.e., software items, such as a functionality or a component) that met all the acceptance criteria established to the user stories they materialize can be added to the sprint deliverable (e.g., a software module) and, thus, to the project deliverable (e.g., a software product). The complementary artifacts, in turn, present the conceptualization to the team by using alternative representations. As we previously said, the SEON extract used in this study focuses mainly on the Scrum Reference Ontology (SRO) and can be found in (Santos Jr et al. 2021a)). Table 6 summarizes some concepts from SRO used in this study.

Table 6. Some concepts from SRO.

Concept	Description
Scrum Project	Software Project that adopts Scrum in its process.
Sprint Backlog	Artifact that contains the Requirements of the product to be developed in the Scrum Project.
Planning Meeting	Ceremony performed in a Sprint where the Development Team plans it.
User Story	Requirement Artifact (i.e., a requirement recorded in some way) that describes Requirements in a Scrum Project. It indicates a goal that the user expects to achieve by using the system and, thus, represents value for the client. A User Story can be an Atomic User Story, when it is not decomposed into others, or an Epic, when it is composed of other Use Stories.
Acceptance Criteria	Requirement established to a User Story and that must be met when the User Story is materialized. Thus, it is used to verify if the User Story was developed correctly and meets the client needs.
Intended Scrum Development Task	Development Task planned to be performed in a Sprint.
Performed Scrum Development Task	Development Task performed in a Scrum Project.
Deliverable	Software Item that materializes User Stories.
Accepted Deliverable	Deliverable that is in conformance to all the Acceptance Criteria established to the User Stories materialized by that Deliverable.
Not Accepted Deliverable	Deliverable that is not in conformance to at least one Acceptance Criteria established to the User Stories materialized by that Deliverable.
Sprint Deliverable	Accepted Deliverable resulting of a Sprint.

Figure 4 illustrates the relationship between the reference ontologies and the complementary artifacts. As a result of this approach, we shortened the distance between the team and the conceptualization provided by the ontologies, improving domain understanding and communication.

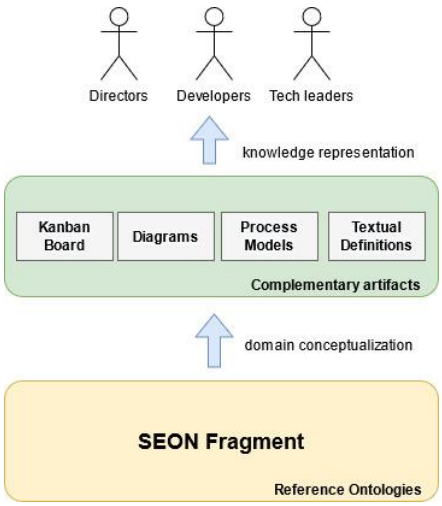

Figure 4. Reference ontologies and complementary representation artifacts.

To address behavioral aspects of Scrum (e.g., activities, the flows between them and objects they manipulate), we created process models. For that, we first mapped concepts from SEON to constructs of BPMN (OMG 2013), which was the modeling language used to represent the process models. Put it simply, we identified which BPMN constructs should be used to represent SEON concepts or their instances. For example, the *Activity* BPMN construct should be used to represent Performed Scrum Development Task, Ceremony, Planning Meeting and other SEON concepts referring to activities, tasks or processes. The *Actor* BPMN construct, in turn, should be used to represent SEON concepts referring to people or roles, such as Developer and Product Owner. Then, we represented and complemented knowledge provided by the reference ontologies by creating process models like the one illustrated in Figure 6. In the process models, by following the approach suggested in (Guizzardi et al. 2016),

we used the *Event* construct to represent the *state of affairs* (i.e., a situation) caused by the execution of an *Activity* or when a temporal constraint started or ended.

The process model shown in Figure 5 was used to illustrate the creation of the **Sprint Backlog** in the **Planning Meeting** ceremony, the selection of **User Stories** to be implemented in **Performed Scrum Development Tasks** and materialized by **Deliverables**, and the validation of the deliverables that, if accepted (**Accepted Deliverable**), are integrated into the **Sprint Deliverable**. If not accepted (**Not Accepted Deliverable**), they must be addressed in new tasks. The **bold** terms aforementioned refer to the SEON concepts addressed in the process model presented in Figure 5. The process complements the conceptualization provided by the ontology by making explicit some activities, the flow between them and the *state of affairs* resulting from the activities execution.

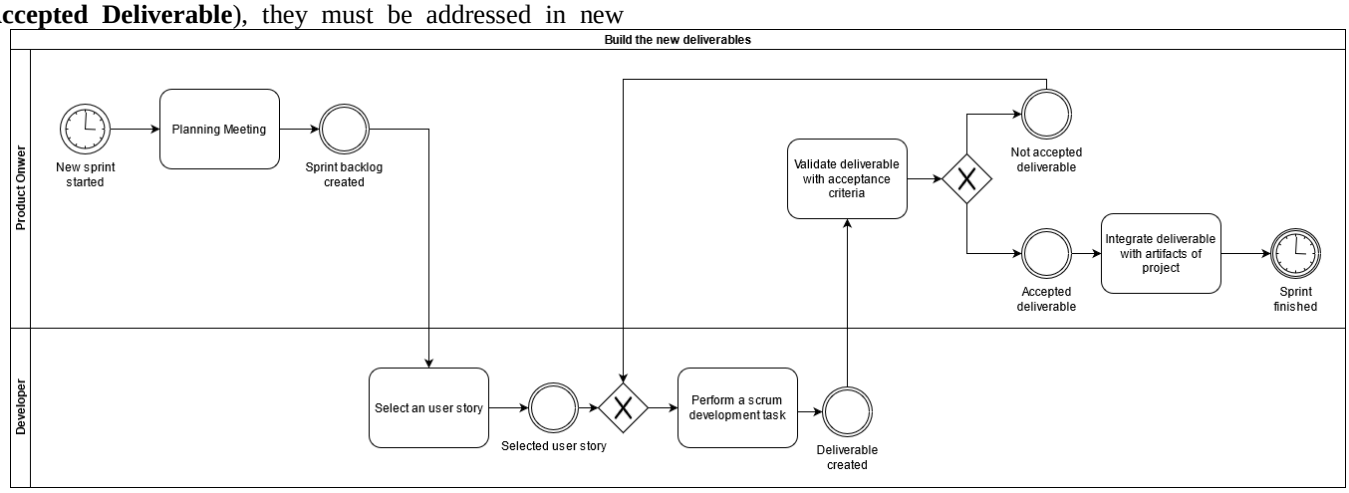

Figure 5. Example of process model created based on SEON conceptualization.

In addition to process models, we also used some diagrams to better illustrate some concepts. For example, to help the team visualize that (i) an **Epic** is a complex **User Story** composed of others, (ii) **User Stories** must have **Acceptance Criteria** established to them, and (iii) in the **Sprint Backlog**, tasks are planned (i.e., **Intended Scrum Development Tasks**) to implement the **User Stories**, we used the diagram shown in Figure 6.

Organization A did not have a clear semantic distinction between epic, user story and task. Many times, these concepts were treated in the same way, being considered as a simple issue by the developers. This lack of conceptual distinction caused problems in project management, estimation, requirements prioritization and communication with Organization B and the client. By using the conceptualization provided by SEON and a simple diagram (as the one shown in Figure 6), the team better understood these concepts and was able to properly use them in backlog management.

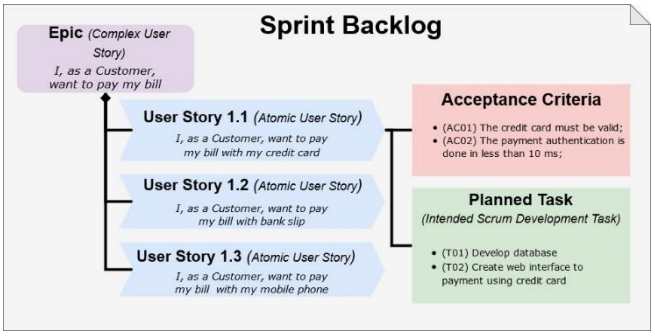

Figure 6. Diagram used to illustrate the relation among Sprint Backlog, Epic, User Story and Intended Task.

We also used a Kanban board to illustrate some concepts. For example, Figure 7 depicts a sketch where we explored tasks and deliverables, showing that if a card is moved to the "*Done*" column, that means that the deliverable produced by the corresponding task must have been evaluated (considering acceptance criteria related to the respective user story) and accepted.

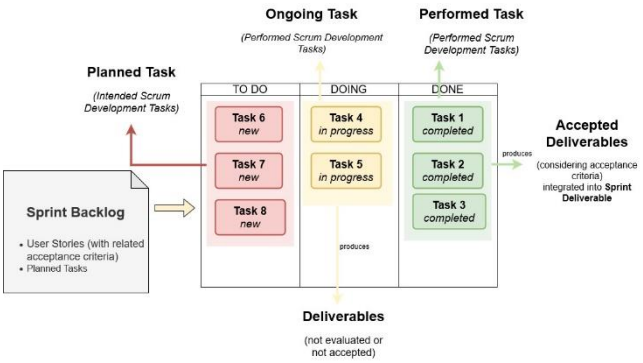

Figure 7. Kanban board illustration used to explore task and deliverable concepts.

To complement the created artifacts, we also created a dictionary of terms (similar to Table 6) containing textual definitions of SEON concepts and some constraints. The ontology and complementary artifacts were used in two workshops where the first and third authors presented the reference ontology and explained its conceptualization by using its conceptual model and the complementary artifacts.

4.3.3 Systems Theory-based process

An important result that arose from this study is a process that combines Systems Theory tools and GUT Matrix to aid organizations to move from traditional to agile. Figure 8 shows the process, and we briefly explain it next.

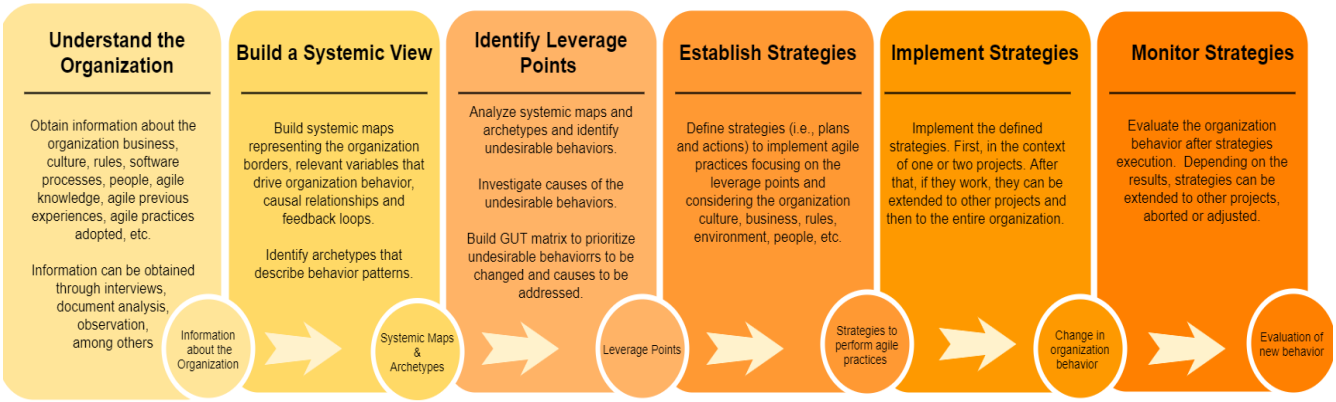

Figure 8. Process to aid defining strategies and implementing agile practices.

Understand the Organization: it consists in obtaining information to understand the organization as a whole so that it will be possible to define strategies to implement agile practices in a suitable way for the organization, considering its culture, environment, business rules, software processes, agile experience and knowledge, people, and so on. Information can be obtained by using techniques such as interviews, document analysis and observation, among others.

Build a Systemic View: this consists in using information obtained in the previous step to build systemic maps to understand organization behaviors relevant in the agile development context. Organization borders, relevant variables that drive organization behavior, causal relationships between them and feedback loops must be represented. Archetypes describing behavior patterns must also be identified from the systemic maps.

Identify Leverage Points: this involves analyzing systematic maps and archetypes to identify undesirable behaviors and their causes. At this point, desirable behaviors in agile organizations suggested in the literature can also be used to verify if the organization fits them. Undesirable behaviors should be prioritized by using a GUT Matrix, so that it is possible to identify which ones represent leverage points and will be addressed in the strategies.

Establish Strategies: this consists in defining strategies (i.e., plans and actions) to implement agile practices focusing on the leverage points and considering the organization culture, business, rules, environment, people, etc.

Implement Strategies: this involves implementing the defined strategies. It is suggested to start with one or two projects. After that, if the strategies work, they can be extended to other projects and then to the entire organization.

Monitor Strategies: this consists in evaluating if undesirable behaviors changed as expected after strategies execution. The new behaviors caused by the strategies need to be evaluated and, depending on the results, strategies can be extended to other projects, aborted or adjusted.

4.3.4 Lesson Learned

In this section, we discuss some lessons we learned in the study. In the lessons learned, we adopt terms such as *should* and *may* instead of mandatory terms such as *must* because we learned the lessons from a single case study. Thus, we believe that other studies are needed to corroborate what we have learned.

Systemic maps should be built with a goal in mind: since systemic maps allow to represent a comprehensive view of how the organization behaves and this may involve many aspects, it is important to focus on variables relevant to the goal to be achieved from the use of the systemic maps. Otherwise, the maps can be too complex and involve variables that do not provide meaningful information for the desired purpose.

The boundaries of the system should be clearly identified: to understand how external elements can influence organization behaviors, it is important to identify the organization boundaries as well the elements that the organization controls and the ones controlled by external agents. This way, it will be possible to create suitable strategies considering both the organization and the external agents.

Changes in leverage points may change the system as a whole: we noticed that when the changes are made in leverage points, particularly in the ones connected to undesirable behaviors with higher priority, the changes tend to provoke a meaningful shift in the organization behavior as a whole, changing existing behaviors and creating others. For example, by changing the way organizations A and B deal with project scope, time and budget, there were also changes in the way Organization A allocates teams and selects requirements to be implemented, and the need for changes in the partnership rules with Organization B was perceived.

Strategies should be integrated into the software processes: for strategies to be performed as part of the organization daily activities, it is important that they are incorporated into the processes performed by the organization. In the study, the strategies were incorporated into the organization software process, involving development, management, and quality assurance activities.

Strategies should be gradually implemented and start in relevant projects: implementing the changes gradually and starting with one or two projects it was positive and the obtained results contributed for the organization to keep the intention of expanding the changes to other projects. We selected projects in which the teams were interested in using agile practices and that was important for the organization, so that the commitment of the team would be higher. This helped to minimize resistance to the new practices. Once they experienced the benefits of following the strategies, team members became disseminators of the new practices and concepts, helping to extend agile culture to other team members.

Strategies results should be measurable: when defining the strategies, we did not define any indicator to measure its effectiveness. However, the tech leaders used some metrics in the projects (e.g., number of hours spent in new urgent development activities, budget deviation, etc.) that helped us

to evaluate the strategies. Thus, when defining the strategies, it is important to define the indicators to be used to evaluate them.

Using System Theory tools may be costly and not trivial: although System Theory tools were very useful to provide an understanding of the organization, they may be a costly choice, because they demand time, effort and knowledge of the tools and organization. Hence, depending on the scope to be considered, it may be difficult or unfeasible to use them. Other methods can be helpful in this context. Considering this learned lesson, we created *Zeppelin* (Santos Jr et al. 2021b), a diagnosis instrument that helps get a "big picture" of the organization by identifying software practices performed by it. Thus, Zeppelin can be used to provide initial knowledge about the organization scenario, allowing to narrow the scope to be further investigated through System Theory tools.

Representing the ontology conceptualization using process models, textual descriptions and simple diagrams can be more palatable than conceptual (structural) models: the reference ontologies of SEON are represented by means of conceptual (structural) models, textual descriptions, and axioms. Although the conceptual model of the SEON view used in the study provides an abstract view showing all the relevant concepts and relations in a single model (Santos Jr et al. 2021a), we noticed that the team preferred textual descriptions and other representations to the SEON conceptual model. Thus, we prepared a document containing the concepts relevant to the study and their detailed description, also including information about constraints and relationships. We also prepared complementary artifacts using process models and other diagrams to illustrate and complement knowledge provided by SEON. This way, the conceptualization provided by the ontologies was represented in a more palatable way for the team.

A consolidated and accessible body of knowledge may help achieve a common conceptualization: in the study, we used ontologies as a reference to establish a common conceptualization of agile development. We are very familiar with ontologies and two of the authors are also authors of the ontologies used in the study, which were established based on the literature and standards and, thus, provide a consensual view of the domain of interest. Considering Organization A needs and the participation of the authors in the study, the used ontologies perfectly fit. However, we are aware that, for an organization not familiar with ontologies, using an ontology as the starting point to establish a common conceptualization can be challenging. We believe that organizations should use a body of knowledge suitable for its characteristics to establish a common conceptualization about the domain of interest. For example, some organizations may prefer to use textual references, such as the Scrum Body of Knowledge (Satpathy 2013).

Changes involving business partners can be hard to implement and demand more flexibility and time: the way Organization B works directly affects Organization A. Due to business arrangements, Organization A does not have enough influence to make changes in Organization B. It can suggest changes, but it cannot demand them. Thus, it was necessary to define strategies that caused only small changes in Organization B (e.g., help to better describe requirements, allow shared control of time and cost). By noticing improvements from the use of the proposed strategies, Organization B may be more willing to further changes.

Squads should have autonomy to choose methods and tools: the organization can have a set of tools, techniques and methods to be adopted in the projects. Guilds can help define that. According to the project team and characteristics, some tools, methods and techniques can suit better. We noticed that the squad became more self-organized when its members could choose the techniques to solve the project problems. For example, in the study, a squad decided to adopt user stories and BDD to describe requirements, while the other used the complete user story template. In both cases, information about requirements was clear and complete. However, each squad chose the technique more suitable for the project and team characteristics.

Agile-related human aspects need to be developed gradually: agile culture demands some soft skills (e.g., self-organized teams, proactivity, empathy) (Lima and Porto 2019) that not are much common in a traditional plan-driven environment. We observed that some members had problems materializing what it means to be self-organized, proactive and empathetic, because they were used to command-control from traditional culture. We noticed that by using short-, medium- and long-term actions (what is short, medium and long is established by the organization), it is possible to gradually develop agile culture. Short-term actions should focus on understanding the needed skills (e.g., promoting debates about soft skills in software development) and practicing them in the projects. Medium-term actions should empower the use of soft skills combined with hard skills (e.g., Human-Centered Design (Smith et al. 2012)). Finally, long-term actions should institutionalize the soft and hard skills and truly change the whole organization culture.

5 Threats to Validity to the Study Results

The validity of a study denotes the trustworthiness of the results. Every study has threats and limitations that should be addressed as much as possible and considered together with the results. In this section, we discuss some threats considering the classification proposed in (Runeson et al. 2012).

The main threat in this study is related to the researchers who conducted the study. Participative case studies are biased and subjective as their results rely on the researchers (Baskerville 1997). The first and third authors acted as consultants in Organization A and were responsible for conducting the interviews, creating systemic maps and GUT Matrix, and defining strategies. Moreover, the authors created the complementary artifacts used to share knowledge of Scrum. Since the authors were very familiar with SEON, they did not have difficulties creating the artifacts. Other people, less familiar with SEON, could have difficulties to create the artifacts or could have created different artifacts. Furthermore, to create the artifacts, the authors took the team preferences into account (the team told us that process models and diagrams were a good choice for it). The researchers participation affects *Internal Validity*, which is concerned with the relationship between results and the applied treatment; *External Validity*, which regards to what extent it is possible to generalize the results from the case specific findings to different cases; and *Reliability Validity*, which refers to what extent data and analysis depend on specific researchers. Aiming to reduce researchers' bias, the members of Organization A that participated in the study (i.e., two directors, one tech leader and two developers) participated in the activities

and validated results. Moreover, another researcher (the second author), external to the organization, evaluated data collection and analysis and was involved in discussing and reflecting on the study and results.

Concerning *Construct Validity*, which is related to the constructs involved in the study, the main threat is that we did not define indicators to evaluate results. Data collection was performed through interviews, which are subjective. To minimize this threat, we used some measures collected in the projects to evaluate the new behaviors caused by the proposed strategies. However, since the measures were not previously defined, they are limited to enable a proper evaluation of the strategies and the effects caused by them. Another threat concerns the notations used to create the complementary artifacts, since the team could misunderstand the represented concepts due to different semantics assigned to the constructs. To address this threat, the authors asked the team to choose the notations to be used and types of artifacts to be created, so that it was possible to produce artifacts consistent with the team knowledge.

In case-based research, after getting results from specific case studies, generalization can be established for similar cases. However, the threats aforementioned constraint generalization. Moreover, the study involved only one organization. Thus, it is not possible to generalize results for cases without researcher intervention or for organizations not similar to Organization A.

6 Conclusions, Future Work and Implications

This paper presented a case study carried out in a Brazilian organization towards the first transition in the path prescribed by the Stairway to Heaven (StH) model (Olsson et al. 2012). Organization A develops software in partnership with a European organization (Organization B) and it does not have direct contact with clients. After an unsuccessful attempt to implement agile practices "by the book", the organization started a long-term process improvement program. To support it, we have used StH to describe the evolution path to be followed by Organization A. To aid in the first transition and move from traditional to agile, we combined Systems Theory tools, GUT Matrix and reference ontologies.

In summary, Systems Theory tools and GUT Matrix were helpful to better understand the organization, find leverage points of change and define strategies aligned to the organization characteristics and priorities. Reference ontologies were useful to establish a common understanding of agile methods, enabling teams to be aware of and, thus, more committed to agile practices and concepts. By using process models, textual descriptions and other diagrams, the conceptualization provided by SEON, the Software Engineering Ontology Network (Ruy et al. 2016), became more palatable to the team, helping achieve a common understanding.

As a result of the initiative, the organization has implemented agile practices in a flexible way and combined with some traditional practices, which is more suitable for the organization characteristics. Due to the obtained results, the organization kept its intention to continue evolving by following the StH stages. In the first transition, it was not possible to propose big changes in the way Organization B works. However, Organization A expects that considering the positive results, Organization B will be more willing to be involved in the evolution path. This will be crucial in the more advanced stage, where data from the clients are needed to support decision-making and identify new opportunities.

Regarding human aspects, we focused mainly on soft skills related to agile culture. Strategy S3 is directly related to human aspects, being responsible for implementing Squads and Guilds. Squads promoted self-organization, trust, leadership, and other important skills in agile organizations. Guilds promoted the creation of processes and organizational culture that enabled sharing and managing knowledge at individual, team, and organizational levels. This knowledge is valuable to the continuous improvement of Organization A. By changing human aspects, S3 enabled Organization A to create processes, vocabulary, and mindset, i.e., an organizational culture that supported the movement from traditional to agile. Moreover, the soft skills developed by S3 supported other strategies. For example, S1 and S2 were possible because S3 developed some soft skills (e.g., effective communication, self-organization and adaptability) that supported S1 and S2.

As for the limitations of our approach, we highlight that it involves a lot of tacit knowledge and judgment. Besides knowledge about System Thinking tools and GUT Matrix, it is necessary to have organizational knowledge to apply them (e.g., one must be able to properly identify problems, investigate causes, define strategies etc.). Moreover, the evaluation of our proposal was limited. We have used it only in the study reported in this paper, which involved the participation of the authors. Furthermore, the evaluation was mainly based on qualitative data. Thus, new studies are necessary to evaluate the proposal in other organizations and quantitatively evaluate the effects of using it.

As future work, we plan to add knowledge (e.g., by means of guidelines) to help others to use our approach. We also intend to explore other Systems Theory tools and combine them with Enterprise Architecture Models to connect system variables, undesirable behaviors and causes to elements of the organization architecture. Concerning Organization A, we plan to monitor the implemented strategies and extend them to other projects. Once the new practices become solid, we plan to aid Organization A in the next transitions, where continuous integration and continuous deployment are performed.

Concerning the use of SEON, we must point out that the authors were familiar with its conceptualization. In fact, as we previously said, the first and third authors are also authors of the Scrum Reference Ontology (Santos Jr et al. 2021a), the SEON ontology concerning Scrum that provided the central concepts explored in the study. This made it easier to create the complementary artifacts and use them to share knowledge with the team to achieve a common understanding and conceptualization in Organization A. It is also worthy highlighting that, although the complementary artifacts are simple artifacts, the conceptualization behind them, provided by SEON is the key point to achieve a common conceptualization and understanding. We have applied the portion of SEON used in the case reported in this paper to integrate data from different applications and provide consolidated information to support decision making in agile organizations, as we reported in (Santos Jr et al. 2021a). We intend to use SEON with this purpose in Organization A. Since the

team has learned SEON conceptualization, we believe that the first step towards this goal has already been given.

Finally, the contributions of this paper have implications for practice and research. Regarding implication for practice, this paper promotes the use of Systems Thinking tools as a means to identify leverage points relevant to moving an organization from traditional to agile development. Furthermore, the proposed strategies can be used by practitioners and organizations to address problems similar to the ones of Organization A. In addition, we showed how ontologies could be used to create artifacts and share a common conceptualization and understanding of agile development. Other people can be inspired by that to solve knowledge problems in agile and other contexts. The Systems Theory-based process also has implications for practice, since it can be used by other organizations to help the transition from traditional to agile development.

Concerning implications for research, this paper introduces the combined use of Systems Theory tools, GUT Matrix and reference ontologies to support the transition from traditional and agile development. The combined use of them and the proposed System Theory-based process can bring new research questions to be explored in further research. Moreover, the successful use of ontologies to create more palatable artifacts to practitioners can be a starting point to new research aiming to make the most of this powerful instrument of knowledge structuring and representation. Using reference ontologies in the industry is still a challenge. The use of operational ontologies is more common in this context, mainly due to the Semantic Web and also to data and systems interoperability solutions. However, reference ontologies are also valuable artifacts and provide structured, common and well-founded knowledge useful to learning and communication. We believe that new research should be conducted to investigate how to make reference ontologies more palatable for the industry. In the study reported in the paper, we gave the first step towards that. However, other advances are needed. In this sense, we believe that new research aiming to overcome the challenges of using ontologies in industrial settings are necessary.

References

Ali N, Lai R (2018) Requirements Engineering in Global Software Development: A Survey Study from the Perspectives of Stakeholders. J Softw 13:520–532. https://doi.org/10.17706/jsw.13.10.520-532

Baskerville R (1997) Distinguishing action research from participative case studies. J Syst Inf Technol 1:24–43. https://doi.org/10.1108/13287269780000733

Bastos EC, Barcellos MP, Falbo R (2018) Using semantic documentation to support software project management. J Data Semant 7:107–132. https://doi.org/10.1007/s13740-018-0089-z

Binamungu LP, Embury SM, Konstantinou N (2020) Characterising the Quality of Behaviour Driven Development Specifications. Springer International Publishing

Bosch J (2014) Continuous Software Engineering: An Introduction. In: Continuous Software Engineering. Springer International Publishing, Cham, pp 3–13

Bringuente AC, Falbo R, Guizzardi G (2011) Using a Foundational Ontology for Reengineering a Software Process Ontology.In: Journal of Information and Data Management (JIDM), vol. 2, pp. 511–526

De Sousa TL, Venson E, Figueiredo RMDC, et al (2016) Using scrum in outsourced government projects: An action research. In: 2016 49th Hawaii International Conference on System Sciences (HICSS). IEEE, pp 5447–5456

Duarte BB, Leal Castro AL, Falbo R, Guizzardi G, Guizzardi RSS, Souza VS (2018) Ontological foundations for software requirements with a focus on requirements at runtime. In: Applied Ontology, vol.13, pp. 73-105

Dybå T, Dingsøyr T (2008) Empirical studies of agile software development: A systematic review. Inf Softw Technol 50:833–859. https://doi.org/https://doi.org/10.1016/j.infsof.2008.01.006

Falbo R (2014) SABiO: Systematic approach for building ontologies. In: ONTO. COM/ODISE@ FOIS.

Falbo R, Ruy F, Guizzardi G, Barcellos MP, Almeida JPA (2014) Towards an Enterprise Ontology Pattern Language. In: Proceedings of the 29th ACM Symposium on Applied Computing (ACM SAC 2014)

Fonseca V, Barcellos MP, Falbo R (2017) An ontology-based approach for integrating tools supporting the software measurement process. Sci Comput Program 135:20–44. https://doi.org/10.1016/j.scico.2016.10.004

Fitzgerald B and Stol K. (2017). Continuous software engineering: A roadmap and agenda. Journal of Systems and Software, Volume 123, pp 176-189, ISSN: 0164-1212, https://doi.org/10.1016/j.jss.2015.06.063

Guizzardi G (2007) On Ontology, Ontologies, Conceptualizations, Modeling Languages, and (Meta)Models. In: Proceedings of the 2007 Conference on Databases and Information Systems IV: Selected Papers from the Seventh International Baltic Conference DB&IS'2006. IOS Press, NLD, pp 18–39

Guizzardi G, Guarino N, Almeida JPA (2016) Ontological Considerations About the Representation of Events and Endurants in Business Models. In: 14th International Conference, BPM 2016. Rio de Janeiro, pp 20–36

Jim M, Piattini M, Vizca A (2009) Challenges and Improvements in Distributed Software Development : A Systematic Review. 2009:. https://doi.org/10.1155/2009/710971

Karvonen T, Lwakatare LE, Sauvola T, et al (2015) Hitting the Target: Practices for Moving Toward Innovation Experiment Systems. In: International Conference of Software Business (ICSOB 2015). Springer, pp 117–131

Kepner CH, Tregoe BB (1981) The new rational manager. Princeton research press Princeton, NJ

Kim DH (1994) Systems archetypes I. Diagnosing systemic issues and designing highleverage interventions.(Toolbox Reprint Series) Cambridge MA: Pegasus Communications

L'Erario A, Gonçalves JA, Fabri JA, et al (2020) CFDSD: a Communication Framework for Distributed Software Development. J Brazilian Comput Soc 26:. https://doi.org/10.1186/s13173-020-00101-7

Leffingwel D (2016) SAFe® 4.0 Reference Guide: Scaled Agile Framework® for Lean Software and Systems Engineering.

Lima T, Porto J (2019) Análise de Soft Skills na Visão de Profissionais da Engenharia de Software. In: Anais do IV Workshop sobre Aspectos Sociais, Humanos e Econômicos de Software. SBC, Porto Alegre, RS, Brasil, pp 31–40

Meadows DH (2008) Thinking in systems: A primer. chelsea green publishing

Olsson HH, Alahyari H, Bosch J (2012) Climbing the Stairway to Heaven: A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software. In: 2012 38th euromicro conference on software engineering and advanced applications. IEEE, pp 392–399

OMG (2013) Business Process Model and Notation (BPMN). Version 2.0.2, Object Management Group (Technical report, Object Management Group)

Prikladnicki R, Audy JLN (2010) Process models in the practice of distributed software development: A systematic review of the literature. Inf Softw Technol 52:779–791. https://doi.org/10.1016/j.infsof.2010.03.009

Rodriguez P, Markkula J, Oivo M, Turula K (2012) Survey on Agile and Lean Usage in Finnish Software Industry. In: Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. Association for Computing Machinery, New York, NY, USA, pp 139–148

Runeson P, Host M, Rainer A, Regnell B (2012) Case Study Research in Software Engineering: Guidelines and Examples, 1st edn. Wiley Publishing

Ruy F, Souza E, Falbo R, Barcellos M (2017) Software Testing Processes in ISO Standards: How to Harmonize Them? In: In Proceedings of the 16th Brazilian Symposium on Software Quality (SBQS). pp 296–310

Ruy FB, Falbo R, Barcellos MP, et al (2016) SEON: A software engineering ontology network. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). pp 527–542

Santos LA, Barcelos MP, Falbo R, Reginaldo CC, Campos PMC (2019) Measurement Task Ontology. In 12nd Seminar on Ontology Research in Brazil (Ontobras 2019).

Santos Jr PS, Barcellos MP, Calhau RF (2020) Am I going to Heaven? In: Proceedings of the 34th Brazilian Symposium on Software Engineering. ACM, Natal, Brazil, pp 309–318

Santos Jr PS, Barcellos MP, Falbo R de A, Almeida JPA (2021a) From a Scrum Reference Ontology to the Integration of Applications for Data-Driven Software Development. Inf Softw Technol 136:106570. https://doi.org/https://doi.org/10.1016/j.infsof.2021.106570

Santos Jr PS, Barcellos MP, Ruy FB (2021b) Tell me: Am I going to Heaven? A Diagnosis Instrument ofContinuous Software Engineering Practices Adoption. In: Evaluation andAssessment in Software Engineering (EASE 2021). ACM, Trond-heim

Satpathy T (ed) (2013) A Guide to the Scrum Body of Knowledge : SBOK Guide. Scrumstudy a brand of VMEdu, Inc

Schwaber, Ken; Sutherland J (2013) The scrum guide-the definitive guide to scrum: The rules of the game

Smith PJ, Beatty R, Hayes CC, et al (2012) Human-Centered Design of Decision-Support Systems. In: Jacko JA (ed) The Human Computer Interaction Handbook, 3rd edn. CRC Press, Boca Raton, FL, pp 589–622

Sterman J (2010) Business dynamics. Irwin/McGraw-Hill c2000..

Sterman JD (1994) Learning in and about complex systems. Syst Dyn Rev 10:291–330

Studer R, Benjamins VR, Fensel D (1998) Knowledge engineering: principles and methods. Data Knowl Eng 25:161–197. https://doi.org/10.1016/S0169-023X(97)00056-6

Williams L, Cockburn A (2003) Agile software development: it's about feedback and change. IEEE Comput 36:39–43

Wynne M, Hellesoy A, Tooke S (2017) The cucumber book: behaviour-driven development for testers and developers. Pragmatic Bookshelf