Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA IN

Computer Science & Engineering

Ciclo XXI

Settore Concorsuale: 01/B1 - INFORMATICA

Settore Scientifico Disciplinare: INF/01 - INFORMATICA

SOCIO–TECHNICAL SOFTWARE ENGINEERING: A QUALITY–ARCHITECTURE–PROCESS PERSPECTIVE

Presentata da: Daniel RUSSO

Coordinatore Dottorato — **Supervisore**

Prof. Paolo Ciaccia — **Prof. Paolo Ciancarini**

Esame finale anno 2019

"*Particularly alarming is the seemingly unavoidable fallibility of large software.*"

Ed David and A.G. Fraser, 1968 NATO Conference

ALMA MATER STUDIORUM

School of Science
Department of Computer Science & Engineering

Abstract

Doctor of Philosophy

Socio–Technical Software Engineering:
a Quality–Architecture–Process Perspective

by Daniel Russo

This dissertation provides a model, which focuses on Quality, Architecture, and Process aspects, to manage software development lifecycles in a sustainable way. Here, with sustainability is meant a context-aware approach to IT, which considers all relevant socio-technical units of analysis. Both social (e.g., at the level of the stakeholders community, organization, team, individual) and technical (e.g., technological environments coding standards, language) dimensions play a key role to develop IT systems which respond to contingent needs and may implement future requirements in a flexible manner. We used different research methods and analyzed the problem from several perspectives, in a *pragmatic* way, to deliver useful insights both to the research and practitioners communities. The Software Quality, Architecture, and Process (SQuAP) model, highlights the key critical factors to develop systems in a sustainable ways. The model was firstly induced and then deduced from a longitudinal research of the financial sector. To support the model, SQuAP-ont, an OWL ontology was develop as a managerial and assessment tool. A real-world case study within a mission-critical environment shows how these dimensions are critical for the development of IT applications. Relevant IT managers concerns were also covered with reference to software reuse and contracting problems. Finally, a long-term contribution for the educational community presents actionable teaching styles and models to train future professionals to act in a Cooperative Thinking fashion.

Acknowledgements

This research journey was not straightforward from many perspectives. It may sounds pleonastic, but I would not be here, writing these acknowledgments, without the help of great people. Surely, I would not be the scholar I am now without the continuous mentoring of Corrado Mangione. He believed and supported me when few people did at that time. Indeed, Corrado is my model of university professor. My greatest achievement would become, one day, like him.

My relation to Computer Science can be seen as a love story. Until I was 26, I was quite unimpressed from technology, which I truly considered a commodity. For this reason I studied (with great passion) Economics & Management. I do not regret this decision, even today, since I believe that it provided me great insights for my research. Moreover, I learned to find interesting problems to work on, and make them interesting also for other people. After my Master's, I got aware about the impact and the role technology in general, and Computer Science in particular, will have on our daily lives. I am very grateful to Giancarlo Succi to have mentored me in this crucial phase of my life.

Starting a PhD in Computer Science & Engineering with a MSc in Innovation & Entrepreneurship, and a BSc in Politics, Philosophy, and Economics was not an obvious move. For some unclear reasons, Paolo Ciancarini took me with him to Bologna. Beyond being my advisor, Paolo is much more a mentor. He educated me to become a computer scientist, reshaping my mindset. He taught me what is important and what is not in Computer Science research, and how to contribute to the community's discussions. Remarkably, he never exercised a strong degree of control on my work. I experienced what academic freedom means. Still he supported and criticized in a constructive way my choices.

Moreover, I would like to thank for the useful suggestions the members of my PhD committee: Roberto Baldoni, Fabio Vitali, and Maurizio Sobrero. For his steady availability and help, I like to thank also the coordinator of the PhD program in Computer Science & Engineering, Paolo Ciaccia and all its members.

My visiting period at the Wirtschaftsuniversität Wien, in Jan Mendling's group was truly inspiring. Working with an international group of people, at the highest academic level taught me the real sense of being part of an academic community.

My academic achievements would not have been possible without the support and work of my co-authors. Here, I would like to thank Paolo Ciancarini, Franco Raffaele Cotugno, Tommaso Falasconi, Franco Fiore, Saverio Giallorenzo, Ivan Lanese, Vincenzo Lomonaco, Angelo Messina, Marcello Missiroli, Andrea Giovanni Nuzzolese, Francesco Poggi, Valentina Presutti, Mario Ruggiero, Alberto Sillitti, Giancarlo Succi, Gerolamo Taccona, Massimo Tomasi, and Paolo Torricelli.

I had also the great luck to work with a great teaching team at the Faculty of Humanities in the Computer Science courses, thanks to Aldo Gangemi, Damiana Luzzi, Federico Montori, Matteo Pascoli, Francesco Poggi, and Elisa Turrini.

I learned a lot how to become a computer scientists also from my fellow colleagues of the *Hardcore Underground*. For the (something more and something less) inspiring discussions I am grateful to Luca Bedogni, Francesco Gavazzo, Saverio Giallorenzo, Micheal Lodi, Vincenzo Lomonaco, Vincenzo Mastroandrea, Federico Montori, Stefano Giovanni Rizzo, Luca Sciullo, Liu Tong, Angelo Trotta, Stefano Pio Zingaro, and obviously the BergaBot.

ACM SIGSOFT and the Heidelberg Laureate Forum Foundation granted me full scholarships for the participation to two world class Computer Science events which were truly inspiring for my PhD: the 50th Turing Award Ceremony and the 2018

Heidelberg Laureate Forum. To such awarding institutions I am particularly thankful for those opportunities.

I also like to acknowledge the Autonomous Region of South Tyrol (Italy), which provided me the financial support for this PhD.

Living between Bologna, Bolzano, Milano, and Vienna was not easy. Not only for me but also for my family. My parents never missed to support me in any of my enterprises since my early years: *Grazie papà e mamma!* Laura, *amore della mia vita*, thank you for all these great years. You were at my side for every decision I took, supporting and my bearing moods.

It was a truly great time in Bologna. Indeed, it was one of the most enjoyable period of my life. For all the people which made this experience so great I want just to say: *grazie di cuore!*

Contents

List of Figures

List of Tables

List of Abbreviations

AV	**A**gile **V**alues
CT	**C**omputational **T**hinking
CooT	**Coo**peration **T**hinking
ISQ	**I**nformation **S**ystems **Q**uality
IPR	**I**ntellectual **P**ropriety **R**ights
KBS	**K**nowledge **B**ased **S**ystems
OWL	**O**ntology **W**eb **L**anguage
PLS	**P**artial **L**east **S**quares
SE	**S**oftware **E**ngineering
SEM	**S**tructural **E**quation **M**modelling
SQuAP	**S**oftware **Qu**ality **A**rchitecture **P**rocess model
SWEBOK	**S**oft**W**are **E**ngineering **B**ody **O**f **K**nowledge
STSE	**S**ocio **T**echnical **S**oftware **E**ngineering

Chapter 1

Research Objectives

1.1 Motivation and Problem Statement

Software Engineering is a human–intensive activity. As such, different social and technical aspects are involved while engineering software. Collaboration among developers, and cooperation with different stakeholders are crucial activities to lead complex software projects to success [377]. Indeed, the relation between software development and the structure of the organizations where the software is effectively developed, has been empirically observed several times in literature [331, 451]. This kind of relation is among the oldest Software Engineering laws, known as Conway's Law. Melvin Conway discovered in 1967 that "organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations" [107]. According to this insight, software is almost isomorphic to the organizational communication structure, where it is build.

The consequences of this wisdom has a great impact on the entire Software Engineering management area. Recently, a long list of scholars and contributions, which are partially listed in the next chapters, supported this critical issue in different ways. In order to have a broad idea of the types of contributions, consider that Nagappan et al provided evidence for causation between organizational structures and software quality [331]. Similarly, Tamburri et al discovered patterns of sub-optimality across organizational structures [451]. Cataldo et al formalized socio-technical congruence, intended as the degree to which technical and social dependencies match when coordination is needed, and empirically investigated its impact on software quality [86]. Bird et al used a social network analysis among developers to discover socio–technical dependencies [56]. Whereas, de Souza et al focused on coordination to maintain legacy systems [430].

Indeed, the impact of sub–optimal coordination on software quality is the biggest concern in literature. A better management of socio–technical dependencies through the coordination activity is considered as a key activity to improve software quality, both for development and maintenance. Several solutions have been proposed by literature to tackle this issue. Chapter 2 provides a mapping, where such approaches have been listed.

However, a comprehensive perspective of the subject matter is still missing. This means that there is no Software Engineering model to deal with the quality of information systems. It is a matter of perspectives and unit of analysis. Especially for complex or legacy systems managing dependencies means to deal with different unit of analysis at the level of methods, classes, files, packages, or even systems. Therefore, solutions proposed by scholars can hardly cover the degree of complexity of such subject matter. In particular, conceptual analysis and conceptual implementation papers obviously addressed single issues of the debate. Several layers of both technical (e.g., technology used, granularity of dependencies) and social (e.g., organizational issues,

communication, cultural barriers) nature are involved in Socio–Technical Software Engineering. For this reason there is no such one 'silver bullet' in this area due to its complexity.

Although we will not solve all problems related to Socio–Technical Software Engineering, this dissertation proposes a model to firstly understand and manage the subject matter, with particular regard to quality issues. Moreover, typical barriers of practical nature are addressed with the idea to serve, in particular, the practitioners community. Among others, we explained a real-world case study where socio–technical concerns have been solved through a Knowledge Engineering approach. Moreover, two key legal issues, like that related to contracting and software cloning are considered to improve software quality from an organizational perspective. Indeed, the way software developers cooperate with an organization impacts on systems' quality. Thus, we developed a contractual framework to improve software quality. Also, we provide guidance for developers regarding software cloning related to Intellectual Property Rights issues. Finally, we considered the pedagogical of future computer scientists generations to solve computationally-complex problems in a cooperative fashion. Even though it might be a neglected topic in this research area, we can not think to tackle socio-technical congruence without educating future professional what cooperation means, since we use to teach students to program in an individual way, and not in teams.

1.2 Contribution

This dissertation provides to the reader an understanding on how socio-technical issues can be treated in several ways. Clear take-aways are provided at the end of each chapter, of particular interest for practitioners. Most work done is of empirical nature, following well-defined methodological approaches, described in each chapter.

Among the most relevant contribution of this thesis is the Quality-Architecture-Process model, induced and deduced in Chapter 3, and conceptualized through an OWL ontology in Chapter 4. However, we did not narrow the discussion on Quality, Architecture, and Process aspects and their interaction. Rather we were interested to investigate socio-technical related aspects of the Quality-Architecture-Process model. The model, as such, suggests key constructs of software artifacts and their relations.

This work offers a perspective on managing software through Software Engineering's key constructs Quality, Architecture, and Process; and key actions to support this perspective. In our view, socio-technical means that software is a (computer-aided) social artifact, which relies on technically-recognized constructs. Accordingly, engineering software means to develop artifacts which are build in a Quality, Architecture, and Process reliable fashion. As highlighted in Chapter 3, we do not focus on these constructs *per se*. Respective standards have been developed by the community and are deeply discussed in literature. Interestingly, as any construct, they change over time, and are not fixed. Although this might sound surprising, it is not. Before the XXI century, Waterfall was considered the most reliable process. After the Agile Manifesto [209] this is not true anymore. Similarly, monolithic mainframe architectures were mainstream before the Service-Oriented or Microservices paradigms. Alike quality attributes, which evolved other time.

Nevertheless, Quality, Architecture, and Process (in their evolution) are at the center of the Software Engineering process. Moreover, it is still a man-made process. In this perspective, software can be considered as a *social construct*, intended as an object that has been developed and accepted by people in a given environment. As such, Software Engineering deals with several units of analysis of social nature:

organization, team, individual; and of technical ones: code, test, documentation, design, etc. The reason is that that it is a complex and interconnected discipline. As Mel Conway pointed out in 1968, software can not be considered as detached from its (organizational, social and technological) environment. Thus, in this context, *sustainability* is intended to adopt a holistic socio-technical perspective on engineering software, having care on all its unit of analysis. A bug-free software with useless requirements for the own organization is useless. However, also a software that have implemented all functional requirements for an organization without any consideration of architectural layering or non-functional requirements will be hardly maintainable.

For these reasons, we are concerned to develop new research insights which help to manage software in a sustainable way. Accordingly, the Meta Research Question (MRQ) of this dissertation is the following:

- *MRQ: How can we engineer software in a sustainable fashion?*

Reasonably, we do not provide a deterministic answer to our MRQ. Since the concept itself of sustainability changes in time, this question will remain reasonably unanswered. The reason may appears quite straightforward. Assuming that software is a social construct, its characteristics evolve according to future challenges. Here, the understanding of the context plays a major role. Pretending to develop one-catch-all solutions may lead to severe pitfalls. Addressing functional aspects without an understanding of e.g., the organization, customers, developers, as also coding and testing standards, environment, libraries; is misleading, especially for critical applications. Indeed, there is a growing understanding in the community that through the context–awareness of a domain, it is possible to recognize, explain, experiment, and build systems in a sustainable way. Recently, Gail Murphy suggested that "study, definition and use of context can improve the flow of software development work" [329]. In her vision, software development is not an abstract and deterministic process, rather it is deeply shaped by socio-technical aspects. Therefore, the notion of sustainability is here of particular importance, since it takes into account the need to embed the awareness of the context within the development process, to deliver useful functionalities (for users) which are also easily maintainable. Thus, providing a holistic vision over the engineering process is a contingent need for all IT managers.

For this reason, this work focuses on providing actionable models, tools, case studies to (some) practitioners' contingent needs.

1.3 Thesis Organization

This thesis includes eight chapters, plus the present introduction and the conclusion and future works chapter. All chapters benefited from a joint contribution of fellow colleagues, with the only exception of Chapter 2. In order to address our MRQ, we investigated several aspects of our subject matter. Consequently, our methodological approaches were tailored to our different research questions, according to our *pragmatic* research perspective [358, 113]. Therefore, each chapters outlines and explains the followed approach. Hence, each chapter in contextualized in its problem statement and related literature. At the end of each chapter, we present our preliminary conclusions, outlying future directions.

1.3.1 Chapter 2

A literature review is provided in a systematic way. Considering the broadness of the topic, we used an appropriate research method, namely a Systematic Mapping or

Scoping study. Here, we addressed Socio–Technical Software Engineering (STSE) as the way to manage, design, develop, test, and maintain complex software systems with a high degree of technical and human dependencies. This chapter aims at surveying existing research on Socio–Technical Software Engineering to identify the literature debate about this broad topic, providing a rigorous starting point for future contributions. Our systematic mapping is launched to find as much literature as possible, finding 94 chapters, which were classified according to their Software Engineering Area, Methodological Philosophy, and Research Method. Most areas of Software Engineering are concerned with STSE, although 90% of the chapters deals with issues related to Software Engineering management, Software design, Software construction, and Software requirements. Leading publication fora are ICSE, CHASE and IST. Only 29% of the articles recognized at least one relevant contributions, suggesting poor theorization. Most papers are concept implementation, conceptual analysis and literature reviews. To conclude, better theorization and theoretical understanding is needed for scholars working in this area, since the picture is too jeopardized.

1.3.2 Chapter 3

Information Systems Quality (ISQ) is a critical source of competitive advantages for organizations and a primary Socio–Technical Software Engineering issue. In a scenario of increasing competition on digital services, ISQ is a competitive differentiation asset. In this regard, managing, maintaining, and evolving IT infrastructures has become a primary concern of organizations. Thus, a socio–technical perspective on ISQ provides a useful guidance to meet current challenges. The financial sector is paradigmatic, since it is a traditional business, with highly complex business–critical legacy systems, facing a tremendous change due to market and regulation drivers. We carried out a Mixed-Methods study, performing a Delphi-like study on the financial sector. We developed a specific research framework to pursue this vertical study. Data were collected in four phases starting with a high level randomly stratified panel of 13 senior managers and then a target-panel of 124 carefully selected and well-informed domain experts. We have identified and dealt several quality factors; they were discussed in a comprehensive model inspired by the ISO 25010, 42010, and 12207 standards, corresponding to software quality, software architecture, and software process, respectively. Our results suggest that the relationship among quality, architecture, and process is a valuable technical perspective to explain the quality of an information system. Thus, we introduce and illustrate a novel meta-model, named SQuAP (Software Quality, Architecture, Process), which is intended to give a comprehensive picture of ISQ by abstracting and connecting detailed individual ISO models.

1.3.3 Chapter 4

Building on the previouse chapter, In Chapter 4, we formalized SQuAP-Ont. The SQuAP model (Software Quality, Architecture, Process) describes twenty-eight main factors that impact on software quality in financial systems, and each factor is described as a relation among some characteristics from the three ISO standards, since their interaction has been scarcely studied, so far. Hence, SQuAP makes such relations emerge rigorously, although informally. In this chapter, we present SQaAP-Ont, an OWL ontology designed by following a well established method based on the re-use of Ontology Design Patterns (i.e. ODPs). SQuAP-Ont formalises the relations emerging

from SQuAP in order to represent and reason via Linked Data about Software Engineering in a three-dimensional model consisting of quality, architecture, and process ISO characteristics.

1.3.4 Chapter 5

Chapter 5 explains explains how cooperation issues were solved through ad hoc technologies, namely Knowledge-Based System (KBS). Here, we discuss how a mission critical KBS has been designed and implemented for a real case study of a governamental organization. This case study represents a factual reality, where KBS have been used in a real-world scenario, where requirements were changing rapidly and developers' effort to implement them were non–trivial. The KBS is based on an ontology used to merge the different mental models of users and developers. Moreover, the ontology of the system is useful for interoperability and knowledge representation. Both the ontology and the main mission critical functionalities have been developed in agile iterations. The KBS has been used for three development activities: (i) requirement disambiguation, (ii) interoperability with some legacy systems, and (iii) information retrieval and display of multiple informative sources. Moreover, the KBS has been developed using a specific agile software development method inspired by Scrum but tailored for Command and Control systems. Due to fast changing operational scenarios and volatile requirements, traditional procedural development methods perform poorly. Thus, a Scrum-like method, called *iAgile*, has been exploited.

1.3.5 Chapter 6

Contracting issues have often been neglected by software engineer. However, legal concerns are compelling to improve socio-technical congruence, in particular collaboration, within real-world contexts. As discussed in the previous chapter, Agile is a suitable methodology to improve collaboration both among development teams and management. Although Agile is a well established software development paradigm, major concerns arise when it comes to contracting issues between a software consumer and a software producer. How to contractualize the Agile production of software, especially for security & mission critical organizations, which typically outsource software projects, has been a major concern since the beginning of the "Agile Era." In literature, little has been done, from a foundational point of view regarding the formalization of such contracts. Indeed, when the development is outsourced, the management of the contractual life is non–trivial. This happens because the interests of the two parties are typically not aligned. In these situations, software houses strive for the minimization of the effort, while the customer commonly expects high quality artifacts. This structural asymmetry can hardly be overcome with traditional "Waterfall" contracts. In this work, we propose a foundational approach to the Law & Economics of Agile contracts. Moreover, we explore the key elements of the Italian procurement law and outline a suitable solution to merge some basic legal constraints with Agile requirements. Finally, a case study is presented, describing how Agile contracting has been concretely implemented in the Italian Defense Acquisition Process. This work is intended to be a framework for Agile contracts for the Italian public sector of critical systems, according to the new contractual law (*Codice degli Appalti*).

1.3.6 Chapter 7

During the development, most code is reused from previous projects, frameworks or libraries. A compelling cooperation problem is to manage properly Intellectual Propriety Rights (IPRs). Although Software Cloning is a widely studied aspect of Software Engineering, little research has been done in its analysis from an IPR perspective An interdisciplinary approach is crucial to better understand the legal implications of software in the IPR context. Interestingly, the academic community of software and systems deals much more with such IPR issues than courts themselves. In this chapter, we analyze some recent legal decisions in using software clones from a Software Engineering perspective. In particular, we survey the behavior of some major courts about cloning issues. As a major outcome of our research, it seems that legal *fora* do not have major concerns regarding copyright infringements in software cloning. The major contribution of this work is a case by case analysis of more than one hundred judgments by the US courts and the European Court of Justice. We compare the US and European courts case laws and discuss the impact of a recent European ruling. The US and EU contexts are quite different, since in the US software is patentable while in the EU it is not. Hence, European courts look more permissive regarding cloning, since "principles," or "ideas," are not copyrightable by themselves.

1.3.7 Chapter 8

Effective cooperation should be properly taught in Computer Science classes. Accordingly, we explore the concept of Computational Thinking in a Software Engineering perspective. Computational Thinking is probably one of the most important skills for XXI centuries citizens, in particular for programmers and software engineers but also for scientists at large. The current teaching practices focus on Computational Thinking and individual programming first, and only later address students to work in teams. However, training students to Computational Thinking might not be enough to tackle contemporary complex challenges in software development, especially those which cannot be won individually. Based on prior studies, we describe and compare four software development learning approaches: solo programmer, pair programmers, self-organized teams, and directed teams. These approaches have been explored in a number of teaching experiments, involving hundreds of students, over several years. We show that Computational Thinking can be effectively enhanced with with Agile values, extending it with social skills and teaming practices. We introduce a model of a competence that we call Cooperative Thinking, grounded in literature and validated by our experiments. This paper provides a research synthesis of previous works contextualized in a pedagogical framework, and proposes a new learning paradigm for Software Engineering education.

1.3.8 Chapter 9

In this chapter we use an exploratory Structural Equation Modeling technique to introduce and analyze *Cooperative Thinking* (CooT), a model of team-based computational problem solving. Computational Thinking (CT) and Agile Values (AV) focus respectively on the individual capability to think in an algorithmic way, and on the principles of collaborative coding. Although these two dimensions of Computer Science education complement each other, very few studies explored the interaction of CT and AV. We ground our model on the existing literature and validate it through Partial Least Square modeling. Cooperative Thinking is not just the sum of CT and

AV, rather it is a new overarching competence suitable to deal with complex Computer Science problems. This chapter suggests to tackle the CooT construct as an education goal, to train new generations to Pareto-optimize both their individual and teaming performances.

Chapter 2

Mapping Socio-Technical Software Engineering

2.1 Introduction

Several aspects of both social and technical nature come into question when developing software. How to collaborate with fellow developers, stakeholders, testing and review activities, design, requirements elicitation, are just a few problems software organizations face every day. The aim of the emerging Socio-Technical Software Engineering (STSE) paradigm is to provide a solution to problems which includes both social and technical aspects of engineering software. Indeed, most of the Empirical Software Engineering effort deals with this topic, at large. Thus, clearly defining this area would misleading. Rather, it might be of interest to understand how the term *socio-technical* is used in literature, finding out how such literature refer to previous works. With all the limitations of this approach, this is a pragmatic approach to map the debate about STSE.

Also defining previous work is not a trivial task. After a first round analysis of the selected literature, we knowledge three major works, which had a clear impact on literature. In 1968 Mel Conway introduced the idea of software as a socio-technical problem [107]. He recently affirmed that "Conway's law was [...] a valid sociological observation, [...] a consequence of the fact that [...] interface structure of a software system necessarily will show a congruence with the social structure of the organization that produced it"[1]. Later on, Fred Brooks built on this wisdom for his pivotal work, providing software management insights [77]. Brooks recognized the management corollaries of Conway's law, suggesting that "structuring an organization for change is much harder than designing a system for change" [77, p. 242]. Conway's and Brooks' ideas highlight the fact that software is a human-made abstraction, and as such, a social construct based on the collective experiences and actions of a community of developers merging with technical instances of the underlying technology involved. The last relevant work on this topic is proposed by Cataldo et al. in 2008 [86]. They build on the idea of congruence, to examine the relationship between the structure of technical and work dependencies, as also the impact of dependencies on software development productivity.

STSE is a interdisciplinary topic, where little systematization has been pursued up to now in literature. Considering the very nature of the topic, it is very hard to find an unified definition about what is meant with *socio-technical*. Moreover, there is no general agreement about the notion of *socio-technical* within Software Engineering. Although most empirical studies deals with social and technical aspects, they might use different notions of phrasing. Indeed, many authors may refer to it

[1] www.melconway.com Accessed on 10.03.2018

without explicitly mention it. As a result, there are probably thousands of studies dealing with this paradigm. For such reason, claiming to map the entire field of STSE would be an overstatement. However, providing an understanding of how scholars use the notion of STSE, and while doing it, which relevant work they are referring to, still provide a first understanding on the subject matter. Building on this awareness, future researcher will be advised in positioning their work within the domain of STSE.

For this reason, planning a mapping study on this subject matter has several inherent limitations. Accordingly, we have just to focus on these papers who explicitly acknowledged *Socio-Technical Software Engineering* within the paper's narrative. Although it is a big shortcut, it is a pragmatic solution to map the literature debate. Scholars, which use this paradigm, do not always refer explicitly to a recognized theoretical framework. The analysis of the bibliography suggests a high variety of different references when hinting to the STSE paradigm. This is a clear signal that a work providing a first mapping of the literature is needed. In order to get a picture of existing literature, we launched a systematic mapping study on the use of the notion of STSE.

Therefore, our aim is to get an overview of existing research in order to find deeper insights about the use of this emerging paradigm and identify future research needs.

Systematic mapping is an alternative research method compared to systematic reviews. Especially if the topic is too broad or poorly systematized for a systematic review, systematic mapping is feasible a methodological approach. Indeed, mapping studies are typically performed at a higher granularity level, mainly to identify research gaps and future research direction. From a methodological perspective, guidelines for systematic mapping have been proposed for the Software Engineering domain since 2008 by Petersen et al. [362], and afterwards expanded by Kitchenham et al. in different contributions [257, 258, 362, 361]. Accordingly, relevant studies has been pursued on Software Engineering topics, like Software Product Lines testing [151]. This chapter follows the recommendation by [362]. To improve reproducibility, we tried to be as compliant as possible to the chapter structure of [151], adapting the mapping scheme for our topic.

This chapter is organized as follows. The methodology of our mapping is explained in Section 2.2. In Section 2.3 a brief literature review on Socio Technical Systems and Software Engineering is provided. The classification scheme is discussed in Section 2.4. Then, in Section 2.5, all studies are discussed by the categories proposed in the SWEBOK. The answer to the research questions are discussed in Section 2.6. Finally, we outline our conclusion in Section 2.7 along with our future works.

2.2 Research Method

2.2.1 Research Questions

The goal of this study is to get an overview of the use of the notion of Socio–Technical Software Engineering. Accordingly, we identified the following research questions (RQ):

- RQ_1 *Which area of Software Engineering have been investigated and to what extent by using the STSE paradigm?* Indeed, STSE relates to several areas of Software Engineering (with reference to the SWBOK [71]), so our aim is to identify which ones have been addressed in previous research, in order to support complementary research.

- RQ_2 *Which is the most used methodological philosophy and research method?* Methodological aspects are the basis for new investigations. This is a pivotal

aspects to strengthen the credibility of research contributions and suggest future space for investigation.

- RQ_3 *Which research fora publish topics where the notion of Socio-Technical Software Engineering is used?* Notably, there is no one forum devoted to it. Thus, future scholars may be find interesting where to submit their contributions.

- RQ_4 *Does the community recognize milestone contributions?* Three works shaped the Software Engineering Socio-Technical paradigm: Conway's intuition that the way people communicate and collaborate frames software design [107]; Brook's experience with complex development [77]; and the socio-technical congruence framework of software projects by Cataldo et al [86]. In order to advance investigations on this topic, awareness on crucial literature is important to criticize or build on it.

- RQ_5 *Which topics regarding Socio-Technical Software Engineering have been identified by studies using this notion?* Challenges for STSE may be identified in several works, helping us to identify relevant past and future research.

2.2.2 Systematic Mapping

Socio-Technical Software Engineering is broad and interdisciplinary topic which would benefit from a first systematization. Accordingly, we carried a systematic mapping study as alternative to a systematic review for the following reasons, as suggested by [362, 361].

Our **goal** is not to establish the state of evidence, since we did not study papers in great detail. Rather, we focused on classification, Software Engineering area analysis, and identifying publication fora. Moreover, we did not consider where particular evidence is missing or is insufficiently reported in existing articles.

The research **process** is also different compared to systematic reviews. Indeed, we did not evaluate papers as such, as we would have made with a literature review. We just assured a baseline quality provided by peer-review and quality indexing by reliable scholarly repositories. From a data extraction perspective, we focused on thematic analysis, to see which areas of Software Engineering are covered by the use of the STSE paradigm. Systematic literature reviews include a deeper level of data selection and extraction to draw homogeneous conclusions based on the same specific subject matter.

As a consequence, there is a **difference in breadth and depth**. We analyzed a large number of papers, compared to a review. The search string and inclusion criteria reflect this, since our aim was to find as many studies as possible to provide to the community the broadest picture. Reviews have a much more stringent search string and exclusion criteria, since the aim is to filter only the most qualified articles for the chosen subject matter.

Also in terms of **classification per topic area and research approach** maps have a broader understating of the subject matter. For this reason we used the SWEBOK areas, which are quite broad, while systematic reviews focus only on one particular sub-area. Nevertheless, we spent also some effort to make a finer granular analysis, depicting also the topic. Similarly, we mapped both research philosophy and the research methods used. We believe that this choice strengthen our work, since it provides at a both high and low level a factual representation about STSE. Again, our aim was not to filter studies and represent only the relevant one for e.g., a comparison

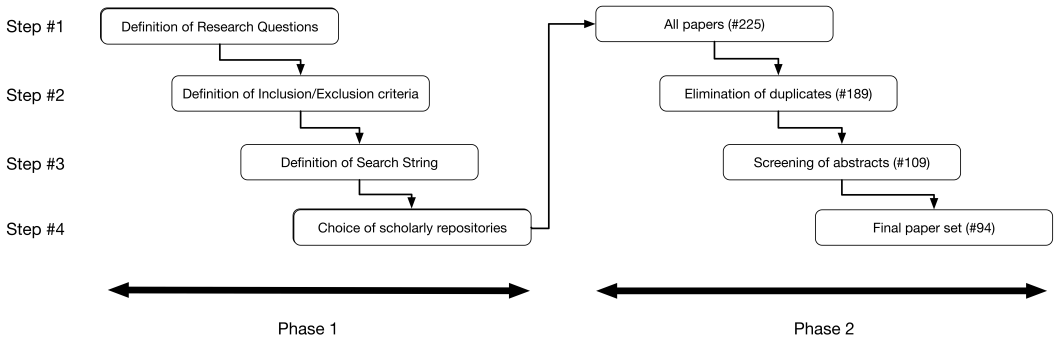


FIGURE 2.1: Systematic mapping search protocol and outcome

about fault prediction metrics [380]; but to provide an overview of ongoing research within one subject matter.

For all these reasons, we pursed a systematic mapping study, pursuing the following activities: search for relevant publications, definition of a classification scheme, and mapping of publications, as suggested by [362].

To do so, we followed the search protocol represented in Figure 2.1. It is composed by two phases, the first defines the search criteria, the second applies them, filtering the results according to them. Each phase has three steps, which narrows the search protocol.

At the very beginning of our research journey we questioned about what literature meant with Socio-Technical Software Engineering. Is it a homogeneous paradigm? Which areas are mostly involved? Which are the most relevant advances? Which are publication fora? How does the community deal with STSE from a methodological perspectives? All these question led to the development of our five research question, listed in Sub Section 2.2.1.

Since this is a mapping study, we tried to be as inclusive as possible, still maintaining an acceptable quality standard, to not bias our outcome. Consequently, our criteria were the following:

- Inclusion: Peer reviewed publications (including short papers) with a focus on Socio-Technical Software Engineering.
- Exclusion: Publications completely out of focus, with no relation to either socio-technical issues or to Software Engineering. Non-peer reviewed publications. Articles not written in English. Posters, doctoral symposium, editorials, panel summaries, and keynotes, since they do not implement a research process.

Also the search string represented the general inclusiveness of our mapping. We searched for ((“socio technical” OR “socio-technical”) AND “Software Engineering”) applied for all titles, abstracts, main texts, and bibliographies. We excluded articles not in English. No time limitation were set in the search.

The choice of the databases was based on the criteria of quality and reproducibility. Accordingly, we choose ACM Digital Library, IEEE Explorer, Scopus, and ISI Web of Science. We explicitly excluded Google Scholar, since it is non-deterministic and does not exclude non–peer review publications.

Once the protocol set up of Phase 1 was completed, we moved to the next one. The first outcome of our string in the chosen repositories produced 225 hints. We did this operation on April, 30 2018. All results are updated to that date. Relevant contributions after that date are not considered in our mapping.

We excluded 36 duplicates. Indeed, we found some semantic inconsistencies among the results of our four repositories. Therefore, we double checked results by both title and DOI. So, duplicates were discarded and abstracts analyzed.

The relevance of the abstract followed the Inclusion/Exclusion criteria rational. We labeled it accordingly, reading each single abstract. To ensure the validity of this screening, the labeling was pursued a second time after one month. In that occasion, the outcome of the first labeling were unknown to the author, to avoid biases. When unclear cases emerged (6 cases), the full paper were read to find out the degree of relevance to our mapping. Borderline papers were kept for this phase. As suggested in [362], we draw a preliminary classification scheme through keywording of abstracts and data extraction.

Finally, we read all full papers. Here, all posters, doctoral symposium, editorials, panel summaries, and keynotes papers, which were not eliminated in the previous step were excluded. The classification scheme were finalized. Two borderline papers were discarded since they were too far from Software Engineering topics. At the end we analyzed 94 articles, which is in line with other similar studies (e.g., Engström & Runeson found 64 contributions for their mapping [151]). Therefore, considering the high number of relevant hits, compared also to other benchmarking studies, the inclusiveness of our search query, we concluded that the search for publications was sufficiently extensive and that the set of publications gives a good picture of the state of art in Socio–Technical Software Engineering. For this reason, we did not opt for snowballing-like techniques [490] in this study.

After all relevant papers were found and the classification scheme were in place, articles were sorted and data extraction took place into the scheme. To maximize validity, this operation was repeated three months later by the author, without knowing the initial mapping to avoid biases. Disagreements about mapping choices were solved with the help of a senior scholar, which led to reclassification and revalidation of previously classified publications.

2.2.3 Threats to Validity

We analyze our threats to the validity, according to the taxonomy used in [151]: construct validity, reliability, internal validity and external validity.

Construct validity has been considered, in order to be sure that the studied object represents what we wanted to investigate, and if it is coherent with our RQs. In this regard, we notice that the Socio–Technical paradigm in Software Engineering, intended as the link between social and technical dimension, is quite established. Still, a unified use of the notion of STSE missing. Most empirical studies may investigate the social and technical dimension of Software Engineering, without explicitly recognizing the notion of STSE. Since the topic is *per se* of interdisciplinary nature, and the Software Engineering domain is quite broad, reasonably, different aspects of the paradigm are investigated by literature. Moreover, we took into consideration also the assurance that all relevant papers regarding the mapping have been retrieved. To do so, we relied on all major Computer Science repositories, namely ACM DL, IEEE Explorer, Scopus, and ISI Web of Science, which index most well reputed publication peer-reviewed fora. We believe that the long list of the founded publication fora is a significant indicator that the width of the searching is enough.

Reliability, to assure repeatability of the study has also been considered. Search strings and outcomes have been made publicly available. We explicitly did not use non-deterministic databases (i.e., Google Scholar) to avoid non-straightforward results. Although Google Scholar entails a huge volumes of scholarly publication, we

compensated it by using relevant and transparent databases. Furthermore, our research was quite broad, since we pursued a mapping study, and not a systematic literature review. Therefore, our inclusion and exclusion criteria were not really stringent. Our main point was to retrieve peer-reviewed publications regarding the Socio–Technical paradigm in Software Engineering. With regard to the classification, we are aware that other scholars may come up with slightly different schemes. However, we tried to strengthen our scheme by using standard classification labels (e.g., from the SWEBOK), and validating it after three months of the first mapping, also with the help of a senior scholar.

Internal validity is not considered harmful, since it is mostly related to the data analysis. In this regard, we just used some quite basic descriptive statistics. To plot figures, we used R and MS Excel.

External validity is also not considered harmful for our mapping study, since we do not draw any generalization. Rather, we describe the state of the art, answering to our five research questions.

In summary, the most severe limitation of this study is to take into account only papers that explicitly mention the term *socio-technical*. However, defining the topic in such a way is the only feasible way to investigate the use of this notion by the literature. As such, we do not overstate conclusions regarding STSE, rather the way scholars use it and refer themselves to literature.

2.3 Socio–Technical Systems

2.3.1 Overview

Socio–Technical Software Engineering can be considered a sub-discipline of Socio–Technical Systems, since most of its insights have been transferred to Software Engineering. The importance of socio–technical aspects in the work organization emerged in 1951, in contrast to the scientific management principles of Taylor [460]. According to this paradigm, any production process is primarily human-driven and can not be rigidly planned in advanced [7]. The focus is on the role of people to turn needs into products or services. So, self–organization is a key driving factor for socio–technical systems. Teams adopting this paradigm have both the knowledge and the authority to set goals and accomplish them [328].

Four are the research streams of Socio–Technical Systems [40]:

1. **Work & Work organization**. Scholars in this area focus on the work design to support human-centric manufacturing systems [327]. After the introduction of computers on the workplace, this stream moved towards the analysis of its impact on the new work organization [141].

2. **Management Information Systems**. This stream is concerned about the use of information systems within organizations, thus these scholars [453] stressed firstly the relevance of socio-technical issues. Most effort is posed on the integration side of these systems within the business, rather than computer-aided work implementation aspects [24].

3. **Computer–Supported Cooperative Work**. Researchers investigate, mainly through ethnographic approaches, the way of working thorough computers [443]. Cooperation and collaboration issues on a qualitative level are analyzed to improve work quality, without considering the impact of organizations' systems requirements and design [197].

4. **Cognitive Systems Engineering**. The relationship between human and organizational issues and systems failure is the focus of this area [212]. Information systems as such are not greatly studied, scholars of this domain are mainly concerned with control and health care systems.

2.3.2 Socio–Technical Software Engineering

Also the Computer Science community started to raise concerns about the way of engineering software, since it is a complex human–intensive activity. The increase of complexity and the degree of interdependencies among systems raised the importance of this paradigm. According to Badham et al [28], five are the characteristics of such systems:

1. Components interdependence.
2. Adaptation to external environments.
3. Interdependence with technical and social subsystems.
4. Systems goals are achieved by more than one means where design decisions are taken during the development (equifinality).
5. Performance based on the joint optimisation of the technical and social subsystems, since the focus on one subsystem would degraded the overall performance.

Accordingly, Socio–Technical Software Engineering addresses the way to manage, design, develop, test, and maintain such systems.

Nowadays, nearly all big organization rely on large legacy software systems with a huge number of interoperable components causing greatest concerns to IT managers [405, 404]. The degree of dependencies are one of these reasons [413]. As a corollary of Conway's Law [107], the best possible way to manage effectively dependencies, in order to build high quality software with a resilient design, is through coordination. Therefore, the coordination of developers whose technical dependencies affect respectively their subsystem improves system's quality. Consequently, the alignment of the technical structure with social interactions is the ground of socio–technical congruence [203].

Socio–technical congruence has been defined as the match between the coordination needs established by the technical domain and the coordination among developers [87]. In this scenario a gap emerges if developers have a coordination need but do not actually coordinate. Research in this domain is concerned to let such gaps emerge to minimize them through a variety of techniques aiming to improve coordination or decrease technical dependencies [410].

To conclude, although socio–technical congruence has probably gained more momentum within the community, still the paradigm of Socio–Technical Software Engineering is quite broad. Any investigation which focus on Software Engineering issues regarding complex systems probably deals at leas with one aspect of Socio–Technical Software Engineering, although they might not recognized it. The way developers interact [72], how they acquire knowledge [148], or deal with geographical separation [152] are STSE concerns. In this sense, Socio–Technical Software Engineering can be intended as a traversal research domain of complex human-driven software systems.

2.4 Classification Schemes

We classified our publications into the following three dimensions: Area of Software Engineering, according to the SWEBOK; Research Philosophy, to see if studies are rather inductive, deductive, or if they follow a Mixed Methods approach; and Research Method, based on the classification scheme of 22 methods by Glass et al [176].

Since STSE impacts potentially on several areas of the Software Engineering domain, a first concern is about the studies distribution, to address RQ1. From the 22 (15 knowledge areas and 7 related areas) Areas of the SWEBOK [71], 11 were relevant for our mapping, namely Software Engineering management, Software design, Software construction, Software requirements, Software Engineering process, Systems engineering, Project management, General management, Software testing, Software Engineering economics, and Software configuration management.

Methodological Philosophy and Research Method of the articles are useful indicators to assess already pursued research and build new one. Indeed, a complementarity of approaches suggests an advanced level of research on the subject matter. The philosophical instances we found were: Deductive, Inductive, and a mix of both. With regard to the Method, we followed the taxonomy of Glass et al [176] according to which we found 11 types, namely: Case study, Literature review/analysis, Concept implementation, Field study, Discourse analysis, Ethnography, Grounded theory, Laboratory experiment (human subjects), and Simulation. We used this taxonomy, in stead of that one proposed by Wieringa et al. [485], since we wanted to provide a deeper understanding about the research methods used. Indeed, great effort have been made to map correctly the papers, to answer RQ2, since our aim is to highlight methodological instances of STSE. In doing so, we believe to stick better to out research question, being more precise.

Finding out where articles where published and of which type (Journal, Conference, Workshop, Magazine) does also provide an interesting insight for future scholar. They could specifically target those venues, having already a pre-screening of the publications. As suggested by RQ3, this is quite valuable, when writing down the related literature section.

According to RQ4, through a bibliography review, we looked if authors considered the following publications of: Conway [107], Brooks [77], and Cataldo et al [86].

Topics discussed in literature were also mapped at a high level, to provide a fast and effective reference to each work, following the rational of RQ5.

2.5 Mapping

The map of our study is represented in Figure 2.2. Following the recommendations of [362], and the example of [151], we used a bubble plot (two scatteplots with bubbles in category intersection, plotted with GGPLOT2 in R) with the following three variables: Methodological Philosophy, Research Method, and Area of Software Engineering, where the size of the bubbles is proportional to the number of papers that are in the pair categories corresponding to the bubble coordinates.

An overview of the Area of Software Engineering, where STSE paradigm is more relevant is presented in Section 2.5.1. Accordingly, we discuss all relevant papers in the relative Subsections, per Area. Afterward, in Section 2.5.2 we discuss the publication fora which host STSE discussions. Finally, we analyze which research type acknowledges more relevant works, in Section 2.5.3.

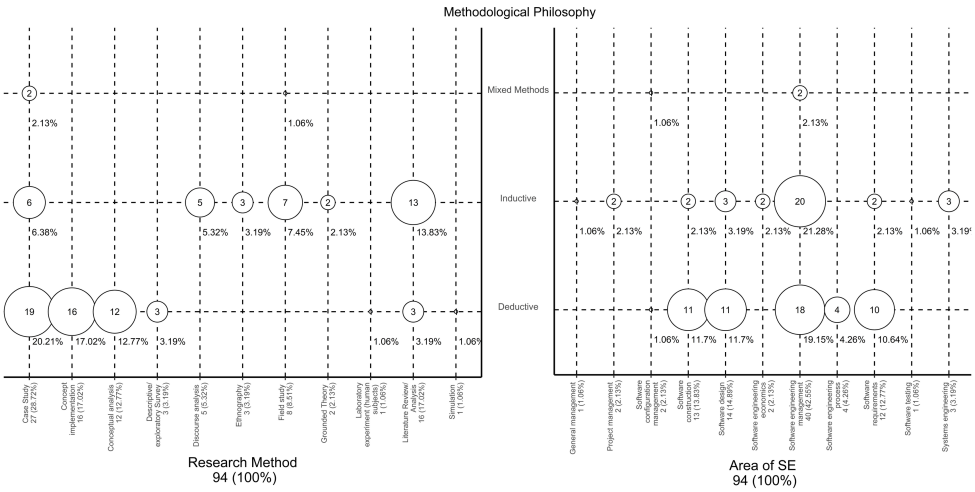


FIGURE 2.2: Map of Methodological Philosophy on Socio-Technical Software Engineering. Methodological Philosophy on the Y axis; Research Method on the left side of the X axis, and Area of Software Engineering on the right type of the X axis

2.5.1 Research Area of Software Engineering

A first distribution of the most relevant research areas of Software Engineering involved to develop the notion of Socio–Technical Software Engineering is presented in Figure 2.3. From this first insight, almost half of the papers are concerns with Software Engineering management issues. Adding also Software design, construction, and requirements, almost 90% of all articles are covered from these four areas. A first conclusion we can draw from this evidence is that although STSE covers 11 SE research areas, only 4 of them were relevant by previous scholars.

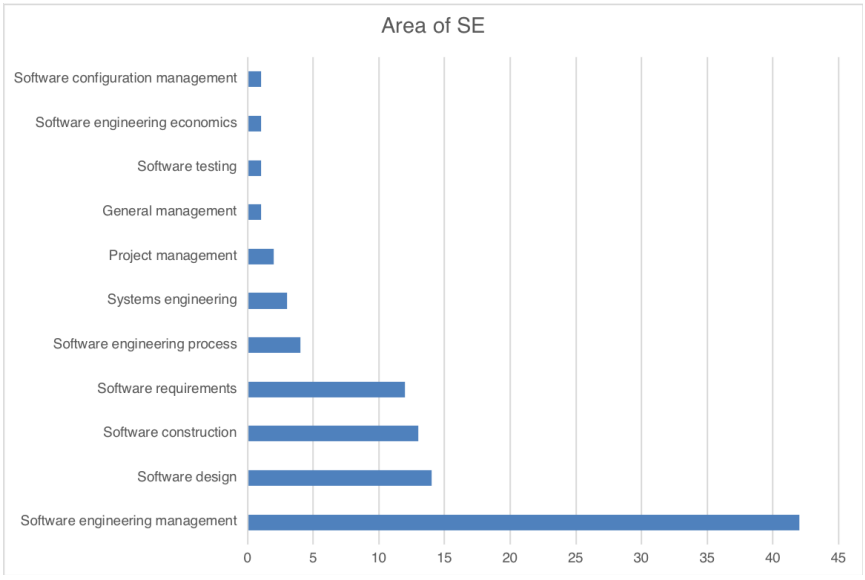


FIGURE 2.3: Distribution over SE research Areas

Although some papers may have cited e.g., quality or maintenance aspects, they were not the focus of the paper. The assignment to an Area is unique, i.e., we made a choice whenever one paper's contribution was more relevant to one Area or another.

Table 2.1: Coverage of SWEBOK's Knowledge Areas

SWEBOK V.3 KAs	Covered
Software Engineering management	✓
Software design	✓
Software construction	✓
Software requirements	✓
Software Engineering process	✓
Systems engineering	✓
Project management	✓
General management	✓
Software testing	✓
Software Engineering economics	✓
Software configuration management	✓
Software maintenance	X
Software configuration management	X
Software Engineering models and methods	X
Software Engineering professional practice	X
Computing foundations	X
Mathematical foundations	X
Engineering foundations	X

In order to provide an overview of the covered Knowledge Areas (KAs), according to Version 3 of the SWEBOK, we marked them accordingly in Table 2.1.

Moreover, contributions over years show a peak of interest between 2011 and 2016, as a general positive trend, like displayed in Figure 2.4.

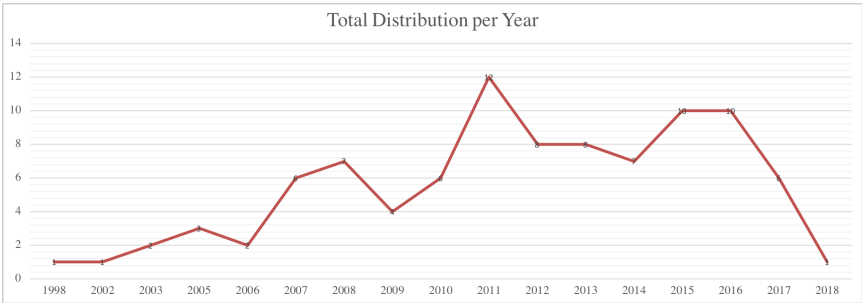


Figure 2.4: Publication Distribution over years

To provide a better understanding when and on which area previous researchers contributed to the STSE paradigm, Table 2.2 shows at a lower granularity the article's distribution per year and area.

Software Engineering Management

Both Table 2.3 and Table 2.4 list articles on Software Engineering management. Considering the amount of papers, we split the two tables.

With reference to Table 2.3, Balasubramanian et al [30] described and explained implementation of Agile in Botswana by observing behavioral patterns. Bider et al [53]

TABLE 2.2: Distribution of research SE Areas's evolution per year

Research focus	1998	2002	2003	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	Total	Percentage
Software Engineering management	1		1	2	2	3	3	2	2	5	2	5	3	3	5	3		42	45%
Software design							1		2	1	1	1	2	4		1	1	14	15%
Software construction						2	1	2	1	2	1		1	2	1			13	14%
Software requirements		1	1	1					1	2	2	2	1		1			12	13%
Software Engineering process										1	1					2		4	4%
Systems engineering										1					2			3	3%
Project management							1				1							2	2%
General management						1												1	1%
Software testing							1											1	1%
Software Engineering economics														1				1	1%
Software configuration management															1			1	1%
Total	1	1	2	3	2	6	7	4	6	12	8	8	7	10	10	6	1	94	100%

developed a modeling technique to explicate and represent various kinds of distances between the functional components to determine which of them constitute risk factors. Tamburri et al[451] found out that social debt is strongly correlated with technical debt and both forces should be reckoned with together during the software process. Yu et al [498] performed a confirmation study of the ability to designate risks through dimension of task, actors, structure and technology. Howinson et al [216] developed a theory of collaboration through open superposition: the process of depositing motivationally independent layers of work on top of each other over time. Gallina et al [162] pursued a model-based analysis that allows analysts to interpret humans and organizations in terms of components and their behavior in terms of failure logic, building on the CHESS-FLA tool for component-based system architectures. Storey et al [440] examined the past, present, and future roles of social media in Software Engineering. Damian et al [119] found out that communication only partially matched task dependencies and that team members that are boundary spanners have extensive domain knowledge and hold key positions in the control structure. Mac Kellar et al [298] described a tool that uses socio-technical congruence measures to support and give advice to students who are learning how to effectively coordinate activities on a group project. Betz et al [51] got the insight that changes in the communication structure alone sooner or later trigger changes in the design structure of the software products to return the socio-technical system into the state of congruence. According to Syeed et al [448], the congruence measure is significantly high in OSS and the congruence value remains stable as the project matured. The paper by Dorn et al [137] proposes a language and methodology for specifying and simulating large-scale collaboration structures, example individual and aggregated pattern simulations, and evaluation of the overall approach. Wen et al [478] performed a socio-technical study of how Swedish municipalities utilizes information channels to handle the OSS security incident and their security posture before, during and after the incident. Global Teaming process areas and associated threats presented in the paper by Richardson et al [385] provides both a guide and motivation for software managers to better understand how to manage technical talent across the globe. A proposal of an aggregated socio-technical congruence measurement that can be used to specify multiple relationships, such as awareness relationships, as interactions that satisfy technical dependencies was outlined by Kwan et al [272]. Zhou et al [501] supported the importance of the initial environment in determining the future of the developers and may lead to better training and learning strategies in software organizations. Kwan et al [273] discovered that there is a relationship between socio-technical congruence and build success probability, but only for certain build types; in some situations, higher congruence actually leads to lower build success rates. Kazam et al [248] presented the Metropolis Model, which attempts to capture and provide a framework for reasoning about this new and increasingly important form of software-intensive system production. According to Al et al [14], development processes adopted to develop tools need to reflect the cooperative dimension. Trainer et al [459] supported that mentees working on user

TABLE 2.3: Papers on Software Engineering management – 1

Article	Title	Methodological Philosophy	Research Method	Cataldo et al	Conway	Brooks
Balasubramanian et al [30]	An evaluation to determine the extent and level of Agile Software Development Methodology adoption and implementation in the Botswana Software Development Industry	Inductive	Case Study	N	N	N
Bider et al [53]	Modeling a global software development project as a complex socio-technical system to facilitate risk management and improve the project structure	Deductive	Conceptual analysis	N	N	N
Tamburri et al[451]	Social debt in Software Engineering: insights from industry	Inductive	Grounded Theory	Y	Y	N
Yu et al [498]	The roots of executive information system development risks	Inductive	Field study	N	N	N
Howinson et al [216]	Collaboration through open superposition	Inductive	Case Study	N	N	N
Gallina et al [162]	Towards safety risk assessment of socio-technical systems via failure logic analysis	Deductive	Concept implementation	N	N	N
Storey et al [440]	The (r) evolution of social media in Software Engineering	Inductive	Literature Review/ Analysis	N	N	Y
Damian et al [119]	The role of domain knowledge and cross-functional communication in socio-technical coordination	Mixed Methods	Field study	Y	N	N
Mac Kellar et al [298]	Analyzing coordination among students in a Software Engineering project course	Deductive	Concept implementation	Y	N	N
Betz et al [51]	An evolutionary perspective on socio-technical congruence: The rubber band effect	Deductive	Literature Review/ Analysis	Y	Y	N
Syeed et al [448]	Socio-technical congruence in OSS projects: Exploring Conway's law in FreeBSD	Deductive	Case Study	Y	Y	Y
Dorn et al [137]	Analyzing design tradeoffs in large-scale socio-technical systems through simulation of dynamic collaboration patterns	Deductive	Conceptual analysis	N	N	N
Wen et al [478]	A Case Study: Heartbleed Vulnerability Management and Swedish Municipalities	Inductive	Field study	N	N	N
Richardson et al [385]	A process framework for global Software Engineering teams	Inductive	Case Study	N	N	N
Kwan et al [272]	Extending socio-technical congruence with awareness relationships	Mixed Methods	Case Study	Y	N	N
Zhou et al [501]	Does the initial environment impact the future of developers?	Deductive	Case Study	Y	Y	N
Kwan et al [273]	Does socio-technical congruence have an effect on software build success? a study of coordination in a software project	Deductive	Case Study	Y	Y	N
Kazam et al [248]	The metropolis model and its implications for the engineering of software ecosystems	Inductive	Discourse analysis	N	N	N
Al et al [14]	Continuous coordination within the context of cooperative and human aspects of Software Engineering	Deductive	Case Study	N	N	Y
Trainer et al [459]	e-mentoring for Software Engineering: a socio-technical perspective	Inductive	Field study	N	N	N
Meneely et al [306]	Socio-technical developer networks: Should we trust our measurements?	Deductive	Simulation	N	N	N

facing, interdependent software form a balance of ties that facilitate both goals, while mentees working on non-user facing software mostly form ties important for building technical skill. For Meneely et al [306], connections in the developer's network are statistically associated with the collaborators whom the developers name.

Passing to Table 2.4, Whitworth et al [484] supported an understanding of how social identity and collective effort are supported by agile methods. Nakakoji et al [333] presented a framework to describe a participative system capable to characterize the evolution of an OSS community through changing the participants' perceived value and types of engagement. Dignum et al [130] provided a conceptual framework to integrate organizational and individual perspectives, such that agent-based models for simulation and the engineering of multi-agent systems can be integrated. Boxer at al [72] presented a case study within the military domain. Scacchi [415] reviewed what is known about free and open source software development work practices, development processes, project and community dynamics, and other socio-technical relationships. Paruma et al [354] provided evidence about the existence of some relationships among

personality traits projected by the committers through their e-mails and the social and technical activities they undertake. According to Herbsleb [202], many tools and practices could be effective for multi-site work, but none seemed to work under all conditions. A proposal of a deductive measure of socio-technical congruence as an indicator of the performance of an organization in carrying out a software development project was proposed by Valetto [463]. Key barriers to collaboration are geographic, temporal, cultural, and linguistic distance; the primary solutions to overcoming these barriers include site visits, synchronous communication technology, and knowledge sharing infrastructure to capture implicit knowledge and make it explicit, according to Noll et al [341]. Balijepally et al [31] illustrated the utility of social capital, organizational learning and knowledge based view of the firm by articulating a research model that captures the IT value created by software development teams practicing different methodologies. For Sawyer et al [414] software development methods reflect theories of how people should behave, how groups of people should interact, the tasks that people should do, the order of these tasks, the tools needed to achieve these tasks, the proper outcomes of these tasks. Brier [76] presented tools which can help in the analysis and synthesis of change which impacts on an organisation's socio-technical systems. Cataldo et al [86] is the most significant work on STSE which build on the idea of congruence, to examine the relationship between the structure of technical and work dependencies and the impact of dependencies on software development productivity. Svanæs et al [447] suggested that user-centered-design projects give priority to an early identification of factors in the context of design that pose risks to end-product usability. For Bendik et al [45] early integration increases the likelihood of implementation success, it also increases the complexity of the projects. Herrmann et al [207] found out that every pattern of a groupware application has to combine the description of social as well as technical structures, and that a single pattern can only be understood in the context of a pattern language. Lyytinen et al [296] examined software risk management, emphasizing the ways in which managers address software risks through sequential attention shaping and intervention. According to Arumugam et al [21], a Socio Technical Systems responsibility model is designed for global practitioners to exhibit the relationship between global practitioners within an organization structure and represent the risk of global practitioners using a set of graphical notations. Finally, Hall et al [192] proposed a design theory for Software Engineering, extending it to a design theory for socio-technical systems.

Software Design

Papers on Software design are displayed in Table 2.5. Here, the focus is primarily of architectural nature. Ahmad et al [10] pursued a systematic literature review on usability guidelines for smartphone applications, studying 148 articles which proposed a total of 359 usability guidelines. Lentzsch et al [285] presented a method which allows multiple stakeholders to reflect on process models they design collaboratively over multiple sessions. Baraki et al [33] proposed patterns to address interdisciplinary concerns in a tightly interwoven manner and are intended to facilitate the development of accepted and acceptable applications that in particular deal with sensitive user context information. Martini et al [300] proposed a taxonomy of the most dangerous socio-technical items for software design. Ogunyemi et al [346] found out that there is a knowledge limit regarding HCI practices in Nigerian software industry. Simpson et al [425] revised the concept of responsibility modeling, which models social technical systems as a collection of actors who discharge their responsibilities, whilst using and

TABLE 2.4: Papers on Software Engineering management – 2

Article	Title	Methodological Philosophy	Research Method	Cataldo et al	Conway	Brooks
Whitworth et al [484]	The social nature of agile teams	Inductive	Grounded Theory	N	N	N
Nakakoji et al [333]	Understanding the Nature of Collaboration in Open-Source Software Development	Deductive	Case Study	N	N	N
Dignum et al [130]	Multi agent simulation for control and autonomy in complex socio-technical systems	Deductive	Conceptual analysis	N	N	N
Boxer at al [72]	Building organizational agility into large-scale software-reliant environments	Deductive	Case Study	N	N	N
Scacchi [415]	Free/open source software development: recent research results and emerging opportunities	Inductive	Literature Review/ Analysis	N	N	N
Paruma et al [354]	Finding relationships between socio-technical aspects and personality traits by mining developer e-mails	Deductive	Case Study	N	N	N
Herbsleb [202]	Building a socio-technical theory of coordination: why and how (outstanding research award)	Inductive	Field study	Y	Y	Y
Valetto [463]	Using software repositories to investigate socio-technical congruence in development projects	Deductive	Concept implementation	Y	Y	N
Noll et al [341]	Global software development and collaboration: barriers and solutions	Inductive	Discourse analysis	N	N	N
Balijepally et al [31]	IT value of software development: A multi-theoretic perspective	Inductive	Literature Review/ Analysis	N	N	N
Sawyer et al [414]	Methods as theories: evidence and arguments for theorizing on software development	Inductive	Literature Review/ Analysis	N	N	N
Brier [76]	Problem-based analysis of organisational change: a real-world example	Deductive	Concept implementation	N	N	N
Cataldo et al [86]	Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity	Deductive	Conceptual analysis	X	Y	Y
Svanæs et al [447]	Understanding the context of design: towards tactical user centered design	Inductive	Case Study	N	N	N
Bendik et al [45]	Four integration patterns: IS development as stepwise adaptation of technology and organisation	Inductive	Case Study	N	N	N
Herrmann et al [207]	Concepts for usable patterns of groupware applications	Deductive	Conceptual analysis	N	N	N
Lyytinen et al [296]	Attention shaping and software risk– A categorical analysis of four classical risk management approaches	Inductive	Literature Review/ Analysis	N	N	Y
Arumugam et al [21]	Global Software development: An approach to design and evaluate the risk factors for global practitioners	Inductive	Literature Review/ Analysis	N	N	N
Hall et al [192]	A design theory for Software Engineering	Inductive	Discourse analysis	N	N	N

TABLE 2.5: Papers on Software design

Article	Title	Methodological Philosophy	Research Method	Cataldo et al	Conway	Brooks
Ahmad et al [10]	Perspectives on usability guidelines for smartphone applications: An empirical investigation and systematic literature review	Deductive	Literature Review/ Analysis	N	N	N
Lentzsch et al [285]	Integrating a Practice Perspective to Privacy by Design	Inductive	Field study	N	N	N
Baraki et al [33]	Interdisciplinary design patterns for socially aware computing	Deductive	Concept implementation	N	N	N
Martini et al [300]	The danger of architectural technical debt: Contagious debt and vicious circles	Inductive	Case Study	N	N	N
Ogunyemi et al [346]	HCI practices in the Nigerian software industry	Inductive	Field study	N	N	N
Simpson et al [425]	Formalising Responsibility Modelling for Automatic Analysis	Deductive	Conceptual analysis	N	N	Y
Dorn et al [138]	Coupling software architecture and human architecture for collaboration-aware system adaptation	Mixed Methods	Conceptual analysis	Y	Y	N
Taveter et al [452]	Engineering societal information systems by agent-oriented modeling	Deductive	Case Study	N	N	N
Georgas et al [171]	STCML: an extensible XML-based language for socio-technical modeling	Deductive	Concept implementation	N	N	N
dos Santos et al [409]	Revisiting the concept of components in Software Engineering from a software ecosystem perspective	Deductive	Descriptive/ exploratory Survey	N	N	Y
Fabiano et al [154]	Applying Tropos to Socio-Technical System Design and Runtime Configuration	Deductive	Case Study	N	N	N
Bicocchi et al [52]	A self-aware, reconfigurable architecture for context awareness	Deductive	Case Study	N	N	N
Norta et al [342]	An agent-oriented method for designing large socio-technical service-ecosystems	Deductive	Case Study	N	N	N
Lock et al [290]	Modelling and Analysis of Socio-Technical System of Systems	Deductive	Concept implementation	N	N	N

producing resources in the process. Mapping mechanism and corresponding framework that enables a system adaptation manager to reason upon the effect of software level changes on human interactions and vice versa was proposed by Dorn et al [138]. Taveter et al [452] proposed agent-oriented modeling as a suitable Software Engineering approach for developing open and adaptive societal information systems. Georgas et al [171] presented STCML: an XML-based, highly-extensible modeling language that makes extensive use of linking and inheritance in order to provide an interoperable data representation with particular support for architectural concerns. Dos Santos et al [409] revisits the concept of components in Software Engineering through a socio-technical construction. Fabiano et al [154] illustrate a number of Tropos features to support the development and runtime reconfiguration of Socio-Technical Systems. Bicocchi et al [52] suggests a framework to evaluate a meta-classification scheme based on state-automata for improving energy efficiency, improving classification accuracy and improving Software Engineering of aware systems. For Norta et al [342] there is a lack of socio-technical design methods for generating agent-based architectures that get deployed on platforms as a service in clouds. Finally, a presentation of an approach to help end users graphically identify and analyze the hazards and associated risks that can arise in complex Socio-Technical System of Systems, with particular emphasis on the role of system dependencies is pursued by Lock et al [290].

Software construction

Regarding software construction, Table 2.6 presents all 13 articles. They focus on coding, verification, unit testing, integration testing, and debugging. The main contribution of Santos et al [408] was to treat non-technical issues of component repositories in the software ecosystems context. Kilamo et al [252] presented the evolution of social dimensions in the light of Software Engineering methodologies and associated tools. Oliva et al [348] identified key developers and characterize them in terms

TABLE 2.6: Papers on Software construction

Article	Title	Methodological Philosophy	Research Method	Cataldo et al	Conway	Brooks
Santos et al [408]	Supporting negotiation and socialization for component markets in software ecosystems context	Deductive	Conceptual analysis	N	N	N
Kilamo et al [252]	The social developer: now, then, and tomorrow	Deductive	Descriptive/ exploratory Survey	N	Y	N
Oliva et al [348]	Evolving the system's core: a case study on the identification and characterization of key developers in Apache Ant	Deductive	Case Study	Y	Y	N
Syeed et al [449]	Socio-technical congruence in the ruby ecosystem	Deductive	Case Study	Y	Y	Y
Wang et al [475]	Which bug should I fix: helping new developers on board a new project	Deductive	Concept implementation	N	N	N
Bird et al [55]	Empirical Software Engineering at Microsoft research	Inductive	Discourse analysis	N	Y	Y
Kuhn [268]	Immediate Search in the IdE as an Example of Socio-Technical Congruence in Search-driven development	Deductive	Case Study	N	N	N
Wermelinger et al [480]	Using formal concept analysis to construct and visualize hierarchies of socio-technical relations	Deductive	Concept implementation	N	Y	N
de Souza et al [429]	An empirical study of software developers' management of dependencies and changes	Inductive	Ethnography	N	N	N
Borici et al [68]	Proxiscientia: Toward real-time visualization of task and developer dependencies in collaborating software development teams	Deductive	Concept implementation	Y	Y	N
Ye et al [496]	A socio-technical framework for supporting programmers	Deductive	Concept implementation	N	N	N
Sarma et al [411]	Tesseract: Interactive visual exploration of socio-technical relationships in software development	Deductive	Concept implementation	Y	Y	N
de Souza et al [430]	Supporting collaborative software development through the visualization of socio-technical dependencies	Deductive	Concept implementation	N	Y	N

of their social activity and contributions. Syeed et al [449] provided an empirical study of the relationships between the development coordination activities and the project dependency structure in the Ruby ecosystem. Wang et al [475] enabled users to identify bugs of interest, resources related to that bug, and visually explore the appropriate socio-technical dependencies for the selected bug in an interactive manner. Bird et al [55] presented the Empirical Software Engineering (ESE) group at Microsoft Research. Kuhn [268] explored the socio-technical congruence of immediate search, i.e. unification of tasks and breakpoints with method calls, which leads to simpler and more extensible development tools. Wermelinger et al [480] presented an application of formal concept analysis, in order to compute and visualize the hierarchical ordering of socio-technical relations. De Souza et al [429] described the strategies used by software developers to handle the effect of software dependencies and changes in their work. Borici et al [68] introduced ProxiScientia, a visualization tool that provides awareness support to developers, as they engage in collaborative software development activities. Ye et al [496] proposed the STePIN (Socio-Technical Platform for In situ Networking) framework to guide the design of systems that support information seeking during different phases of programming. Sarma et al [411] developed Tesseract, an interactive exploratory environment that utilizes cross-linked displays to visualize the myriad relationships between artifacts, developers, bugs, and communications. De Souza et al [430] developed Ariadne, a plug-in for Eclipse that analyzes software projects for dependencies and collects authorship information about projects relying on configuration management repositories.

Software requirements

Table 2.7 is concerned with software requirements papers, which deals with establishing the needs of stakeholders solved by software through the STSE paradigm. Kim et al [253] proposed a methodology for systematically organizing knowledge with a security requirements recommendation framework using the Problem Domain Ontology. While Hassine et al [195] proposed a structural metric to measure actor external dependencies in GRL (Goal-oriented Requirement Language) models. Dalpiaz et al [118] presented in their work a requirements driven architecture, which extend the Tropos goal models to diagnose failures as well as to identify alternative strategies to meet requirements. Dos Sanots et al [139] explained an approach to support software ecosystems definition and modeling based on their domains. Morales et al [321] pursued a retrospective analysis of the requirements engineering process of a project in the domain of ambient assisted living, where several techniques were used to elicit the requirements of a Socio-Technical System. Schneider et al [419] present an approach supporting organizational learning on security requirements by establishing company-wide experience resources and a socio-technical network to benefit from them. According to Kamaruddin et al [243], the application of Activity Theory (which focuses the human practices of development process, both the individual and social levels) is ideally suited to handle mobile application requirement. Bourimi et al [70] introduced the AFFINE framework which considers non-functional requirements early in the development process, balance end-users' with developers' needs, and provide a reference architecture support for non-functional requirements. Gregoriades et al [182] described a method and a tool for validating nonfunctional requirements in complex Socio-Technical Systems. Pedell et al [357] presented a method for using ethnographic field data to substantiate agent-based models for socially-oriented systems. Bresciani et al [75] introduced an agent-based Requirements Engineering Framework (REF), devised to deal with Socio-Technical Systems, and support stakeholders' participation. Sutcliffe et al [445] presented an approach to develop a probabilistic model of system reliability as a Bayesian Belief Network.

Software Engineering process

With regard to process issues, Table 2.8 represent the relevant papers. According to Kakar [242], the level of self-organization varies across Agile projects, on each of the nine dimensions the level of self-organization in agile teams was found to be significantly higher than those using plan-driven methods. Furthermore, self-organization was found to positively affect the motivation and innovativeness of software development teams. The model proposal by Jiang et al [228] uses three congruence measures to examine the levels of social-technical congruence in software development processes. Hudert et al [219] proposed a service-centric life cycle model acting as a conceptual basis for automated service management at both, build and run time. Finally, Wen [479] found out that system verification is the most cited security area in OSS research; the socio-technical perspective has not gained much attention in this research area; no research has been conducted focusing on the aspects of security knowledge management in OSS development.

Systems engineering

A few papers focused also on how to design and manage complex systems over their life cycles. Table 2.9 presents the following works. Nardin et al [334] proposed a sanction typology and a conceptual sanctioning process model providing a functional structure

TABLE 2.7: Papers on Software requirements

Article	Title	Methodological Philosophy	Research Method	Cataldo et al	Conway	Brooks
Kim et al [253]	Analytical study of cognitive layered approach for understanding security requirements using problem domain ontology	Mixed Methods	Conceptual analysis	N	N	N
Hassine et al [195]	Measurement of Actor External Dependencies in GRL Models	Deductive	Concept implementation	N	N	N
Dalpiaz et al [118]	Adaptive socio-technical systems: a requirements-based approach	Deductive	Conceptual analysis	N	N	N
dos Sanots et al [139]	On the Impact of Software Ecosystems in Requirements Communication and Management	Inductive	Literature Review/ Analysis	N	N	N
Morales et al [321]	Revealing the obvious?: A retrospective artifact analysis for an ambient assisted-living project	Deductive	Case Study	N	N	N
Schneider et al [419]	Enhancing security requirements engineering by organizational learning	Deductive	Case Study	N	N	N
Kamaruddin et al [243]	Using activity theory in analyzing requirements for mobile phone application	Deductive	Case Study	N	N	N
Bourimi et al [70]	AFFINE for enforcing earlier consideration of NFRs and human factors when building socio-technical systems following agile methodologies	Deductive	Concept implementation	N	N	N
Gregoriades et al [182]	Scenario-based assessment of nonfunctional requirements	Deductive	Concept implementation	N	N	N
Pedell et al [357]	Substantiating agent-based quality goals for understanding socio-technical systems	Inductive	Ethnography	N	N	N
Bresciani et al [75]	The Agent at the Center of the Requirements Engineering Process	Deductive	Conceptual analysis	N	N	N
Sutcliffe et al [445]	Validating functional system requirements with scenarios	Deductive	Conceptual analysis	N	N	N

TABLE 2.8: Papers on Software Engineering process

Article	Title	Methodological Philosophy	Research Method	Cataldo et al	Conway	Brooks
Kakar [242]	Assessing Self-Organization in Agile Software Development Teams	Deductive	Descriptive/ exploratory Survey	N	N	Y
Jiang et al [228]	Assessing team performance from a socio-technical congruence perspective	Deductive	Laboratory experiment (human subjects)	Y	Y	N
Hudert et al [219]	A Proposal for a Life Cycle Model for Electronic Service Markets	Deductive	Case Study	N	N	N
Wen [479]	Software security in open source development: A systematic literature review	Deductive	Literature Review/ Analysis	N	N	N

TABLE 2.9: Papers on Systems engineering

Article	Title	Methodological Philosophy	Research Method	Cataldo et al	Conway	Brooks
Nardin et al [334]	Classifying sanctions and designing a conceptual sanctioning process model for socio-technical systems	Inductive	Literature Review/ Analysis	N	N	N
Gulden et al [187]	A Research Agenda on Visualizations in Information Systems Engineering	Inductive	Literature Review/ Analysis	N	N	N
Baxter et al [40]	Socio-technical systems: From design methods to systems engineering	Inductive	Literature Review/ Analysis	N	N	N

TABLE 2.10: Papers on Project management

Article	Title	Methodological Philosophy	Research Method	Cataldo et al	Conway	Brooks
Harrison et al [193]	Collaboration infrastructure for the learning organization	Inductive	Literature Review/ Analysis	N	N	N
Crofts et al [114]	Using the Sociotechnical Approach in Global Software Developments: Is the Theory Relevant today?	Inductive	Literature Review/ Analysis	N	N	N

for sanctioning in Socio-Technical Systems. Gulden et al [187] outlined a research agenda on visualizations for socio-technical artifacts. Baxter et al [40] identified interdisciplinary research problems that address how to apply socio-technical approaches in a cost-effective way, and how to facilitate the integration of Socio-Technical Systems engineering with existing systems and Software Engineering approaches.

Project management

Although not strictly related to the Software Engineering domain, the papers represented in Table 2.10 provide two interesting contribution on project management issues. Harrison et al [193] affirmed that the Human Interaction Management, the Human Interaction Management System, and Goal-Oriented Organization Design provide the basis for a collaboration infrastructure that is conducive to the Learning Organization and that exemplifies good socio-technical design. While Crofts et al [114] discussed about theory use in socio-technical research with particular reference to Global Software Engineering.

Software Engineering economics

Two papers were also concerned with economics issues. In particular, Kim et al [255] support the concept of a social platform, the importance of core technology, and key properties of the ICT ecosystems. And Cevenini et al [88] elaborated on the articulate aspects of anonymization where a balancing between legal and technical aspects could possibly ensure the system efficiency while preserving the individual right to privacy. Those papers are represented in Table 2.11.

TABLE 2.11: Papers on Software Engineering economics

Article	Title	Methodological Philosophy	Research Method	Cataldo et al	Conway	Brooks
Kim et al [255]	A socio-technical analysis of software policy in Korea: Towards a central role for building ICT ecosystems	Inductive	Field study	N	N	N
Cevenini et a [88]	Privacy Through Anonymisation in Large-Scale Socio-Technical Systems: Multi-lingual Contact Centres Across the EU	Inductive	Literature Review/ Analysis	N	N	N

Table 2.12: Papers on Software configuration management

Article	Title	Methodological Philosophy	Research Method	Cataldo et al	Conway	Brooks
Bider et al [54]	Becoming Agile in a Non-disruptive Way-Is It Possible?	Mixed Methods	Case Study	N	N	N
Znamenskij et al [502]	Effect driven Evolution: Information Systems Architecture for Large Dynamic Organizations	Deductive	Conceptual analysis	N	N	N

Table 2.13: Papers on Software testing

Article	Title	Methodological Philosophy	Research Method	Cataldo et al	Conway	Brooks
Martin et al [299]	Cooperative work in software testing	Inductive	Ethnography	N	N	N

Software configuration management

Papers of Table 2.12 deal with tracking and controlling tasks in software. Bider et al [54] developed a non-disruptive method of transition to Agile, while using a knowledge transformation perspective to identify the main features of an Agile mindset and how it differs from the one of traditional methodologies. While Znamenskij et al [502] advanced a new approach for evolutionary information system engineering to provide high-quality support for large and complex Socio-Technical Systems.

Software testing

One article is about testing, in Table 2.13. Martin et al [299] considered cooperative work crucial to get testing done.

General management

Finally, Drozdowski et al [140] were concerned that the Indian government has provided substantial economic liberalization to support their software industries, though continued reform is necessary to ensure that they remain a software superpower, in Table 2.14.

2.5.2 Publication Fora

Publication fora for papers using the notion of Socio–Technical Software Engineering are very diversified, with 70 identified venues. However, their significance is very skewed. Only 10 venues published more than one paper about STSE, and the International Conference on Software Engineering (ICSE) alone hosted 9 contributions. In total, the following venues: International Conference on Software Engineering; International Workshop on Cooperative and Human Aspects of Software Engineering; Information and Software Technology; Foundations of Software Engineering; Asia-Pacific Software Engineering Conference; IEEE Transactions on Software Engineering; International Conference on Human Aspects of Information Security, Privacy,

Table 2.14: Papers on General management

Article	Title	Methodological Philosophy	Research Method	Cataldo et al	Conway	Brooks
Drozdowski et al [140]	India's Rise as a Software Power: Governmental Policy Factors	Inductive	Discourse analysis	N	N	N

Table 2.15: Distribution of publication fora 1

Venue	Venue Type	#
International Conference on Software Engineering	Conference	9
International Workshop on Cooperative and Human Aspects of Software Engineering	Workshop	6
Information and Software Technology	Journal	4
Foundations of Software Engineering	Conference	3
Asia-Pacific Software Engineering Conference	Conference	2
IEEE Transactions on Software Engineering	Journal	2
International Conference on Human Aspects of Information Security, Privacy, and Trust	Conference	2
International Conference on Supporting Group Work	Conference	2
International Workshop on Social Software Engineering	Workshop	2
Requirements Engineering	Journal	2
ACM Inroads	Magazine	1
Agile conference	Conference	1
Americas Conference on Information Systems	Conference	1
Australasian Conference on Information Systems	Conference	1
Computing and Informatics	Journal	1
Confederated International Conferences On the Move to Meaningful Internet Systems	Conference	1
Conference of Open Innovations Association	Conference	1
Conference on Commerce and Enterprise Computing	Conference	1
Conference on Computer Supported Cooperative Work	Conference	1
Conference on Manufacturing Modelling, Management, and Control International Federation of Automatic Control	Conference	1
Conference on Software Engineering Education and Training	Conference	1
European Conference of Information Systems	Conference	1
European Conference on Software Architecture	Conference	1
Future of Software Engineering	Conference	1
Human-Computer Interaction	Conference	1
Information Systems Research	Journal	1
Interacting with Computers	Journal	1
International Conference on Advances in Computing and Communication Engineering	Conference	1
International Conference on Autonomous Agents and Multiagent Systems	Conference	1
International Conference on Business Information Systems	Conference	1
International Conference on Engineering of Complex Computer Systems	Conference	1
International Conference on Enterprise Information Systems	Conference	1
International Conference on Evaluation of Novel Software Approaches to Software Engineering	Conference	1

and Trust; International Conference on Supporting Group Work; International Workshop on Social Software Engineering; Requirements Engineering, published 34 articles, which represent 36% of total papers.

Table 2.15, and Table 2.16 provide a plastic representation of publication fora ordered per number of articles published and venue type.

2.5.3 Citations per publication types

The works by Conway, Cataldo et al, and Brooks, seem to be poorly recognized. Table 2.17 shows that only 19% of the analyzed papers refer to Conway, 17% to Cataldo et al, and just 12% to Brooks' book. Since three works referred explicitly to at least one of the reference articles, we included them. In particular, Herbsleb [202] cited Conway through his previous publication [204]. Sarma et al [411] acknowledge both Conway and Cataldo et al in other works. Similary did Valetto [463] with Cataldo et al.

In terms of publication type, we found 17 Journal papers, 56 Conference papers, 1 Magazine paper, and 20 Workshop papers.

The majority of the mapped articles did not refer to a specific theoretical Socio–Technical Software Engineering paradigm. Mostly, they referred to STSE as the combination of social and technical issues.

Table 2.16: Distribution of publication fora 2

Venue	Venue Type	#
International Conference on Global Software Engineering	Conference	1
International Conference on Human-Centred Software Engineering	Conference	1
International Conference on Internet Science	Conference	1
International Conference on Networking, Sensing and Control	Conference	1
International Conference on Open Source Systems	Conference	1
International Conference on Requirements Engineering	Conference	1
International Conference on Software and System Process	Conference	1
International Conference on Software Engineering and Knowledge Engineering	Conference	1
International Symposium on Empirical Software Engineering and Measurement	Conference	1
International Symposium on Open Collaboration	Conference	1
International Symposium on Software Reliability Engineering Workshops	Workshop	1
International Systems Conference	Conference	1
International Workshop on Advances and Applications of Problem Frames	Workshop	1
International Workshop on Empirical Requirements Engineering	Workshop	1
International Workshop on Mining Software Repositories	Workshop	1
International Workshop on Replication in Empirical Software Engineering Research	Workshop	1
International Workshop on Software Engineering for Large-Scale Multi-Agent Systems	Workshop	1
iStar	Workshop	1
Italian Workshop dagli Oggetti agli Agenti	Workshop	1
Journal of Ambient Intelligence and Smart Environments	Journal	1
Journal of Computer Information Systems	Journal	1
Journal of Internet Services and Applications	Journal	1
Latin American Computing Conference	Conference	1
Malaysian Conference in Software Engineering	Conference	1
MIS Quarterly	Journal	1
Nordic Conference on Human-Computer Interaction	Conference	1
Portland International Conference on Management of Engineering and Technology	Conference	1
Requirements Engineering @ Brazil	Workshop	1
Social Inclusion: Societal and Organizational Implications for Information Systems	Conference	1
Symposium on Computers and Communication	Conference	1
Telecommunications Policy	Journal	1
The Knowledge Engineering Review	Journal	1
Working Conference on Software Architecture	Conference	1
Workshop on Enterprise and Organizational Modeling and Simulation	Workshop	1
Workshop on Future of Software Engineering Research	Workshop	1
Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation	Workshop	1
World Congress on Services	Conference	1

Table 2.17: Acknowledgment of previous works per publication types

Type of publication	Conway	Cataldo et al	Brooks
Conference	11	9	7
Journal	3	3	2
Workshop	4	4	2
Total	18	16	11
Percentage	19%	17%	12%

This outcome suggest space for improvement in order to identify theoretical instances of STSE.

2.6 Discussion

The surveyed research indicates that the use of the notion of Socio–Technical Software Engineering is a diversified area of investigation, skewed on a few areas and venues.

Answering RQ_1 is quite straightforward, since about 90% of all articles are about Software Engineering management, Software design, Software construction, and Software requirements. Other areas are quite marginal. Especially management issues are the biggest concerns of scholars. Collaboration, cooperation, and work organization are key topics of STSE. Both analysis and solutions on this area focus to improve the gap between the way people work and interact to provide technology solutions. However, the way scholars discuss about this gap are quite broad and target on several topics and specific solutions. Indeed, in Section 2.5.1 the RQ_5 addresses a detailed discussion about each paper per area.

Publication fora are also very skewed on 10 venues. Although we found 70 fora, most of them published just one article about STSE. However, the most relevant venues for such type of contribution is Software Engineering's leading venue ICSE, with 9 papers. Another significant fora is the International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), which focus especially cooperation topics. Instead, the most important journal is Elsevier's Information and Software Technology (IST), with 4 articles. The reason seems quite clear, since IST focus on contributions concerning software management, to improve software development practices. So, RQ_3 can by answered by saying that there is no specific fora for STSE topics. However, cross–area Software Engineering venues, like ICSE appear to be the best suited publication target. Nevertheless, when targeting highly specific issues (like cooperation), more specific venues, like CHASE, are also suitable.

The lack of systematization is evident, after having pursued a bibliography review. Indeed, just 27 papers (29% of the total) recognized at least one of relevant past contributions, and just four recognize all three of them, suggesting that there is not a common view about the STSE paradigm. Since more than 70% of analyzed works did not acknowledge properly key works for articles' theorization, we answer RQ_4 negatively.

Most papers are case studies, concept implementation, conceptual analysis and literature reviews. This finding is fairly similar to that of Glass et al. [176]. So, 64% of the papers mainly tend to develop tools to support or assess collaboration among developers, find new forms of formalization, explore socio–technical issues in real world contexts and undergo critical reflections through literature reviews. Interestingly, although most of them use a deductive–quantitative approach, a significant 38% provided inductive–qualitative contributions. According to Glass's results, Computer Science is a traditionally deductive community [176]. A wider use of methodological philosophies, which enriches Computer Science contributions is a positive note. Researchers used for their investigation a broad use of methods, also mixing them. However, we have to say that very rarely, papers committed explicitly to one philosophy, or method. Mostly, the methodological section was omitted or given for granted. Nevertheless, with regard to RQ_2, we can affirm that both research philosophy and methods use are quite diversified, suggesting a methodological maturity of scholars in this area, although they need to improve methodological clarity.

2.7 Conclusions

The aim of this chapter was to systematize the broad and diversified literature around the use of the notion of Socio–Technical Software Engineering. This emerging paradigm is quite often used, and given often for granted. Since there is no unanimous and homogeneous use of STSE, a systematic literature review was not a suitable research approach. According to our scope, we launched a systematic mapping study, identifying 94 articles published over 20 years.

The picture from our research suggests that the Software Engineering community is seriously concerned to improve the way developers face collaboration to build new software. Moreover, there is also an increasing awareness towards social issues in Software Engineering. The type and topics of these contributions are quite broad and diversified. In addition, the debate about *what is* Socio–Technical Software Engineering is quite jeopardized. Relevant literature is mostly missing and poor theoretical support is provided by scholars in this domain. On one hand, researchers explore collaboration aspects to improve technical solutions, on the other hand, they are not concerned about theoretical aspects of collaboration in order to improve their solutions.

Although we surveyed only the use of the term STSE, and can not draw general conclusions about the the entire research dealing with both social and technical aspects of Software Engineering, findings are significant from several aspects. It provides a fair overview of the way the term STSE was used, from several dimensions (e.g., area, methodological), identifying paper's research goal. Furthermore, it offers a first insight about the use of STSE, highlighting the relation to previous literature. Finally, it can considered a good proxy for researchers dealing with socio-technical issues to find out the most suited publication venue and relevant literature.

All in all, this mapping study is a pragmatic way to get insight about what the community's debate around Socio-Technical Software Engineering.

Future research will focus on two directions. Firstly, theorization works should provide a homogeneous model for STSE. To do so, all provided frameworks of the papers have to be systematized, discussed and compared. Secondly, find new relations from other areas to support cross–fertilization. Indeed, STSE is not a strictly Software Engineering paradigm, several of its insights are mutated from social sciences. Therefore, a deeper understanding of these relations underpins STSE and enriches it.

Chapter 3

The Software Quality–Architecture–Process Model

3.1 Introduction

The quality and flexibility of Information Systems are among the most relevant sources of competitive advantage [372, 464]. While new digital 'platform' companies arise, disrupting traditional business models and entire industries, established companies are striving to compete with these new entrants [360]. Incumbent organizations developed their information systems over years with long-lasting IT plans with mainframe-like systems [251]. Considering the level of pervasiveness of software, which is increasing at a path we have never experienced before, the digitalization of services and products are the new challenges of incumbents. While consumers are becoming more keen to use technology for their daily applications, businesses are rethinking the values they offer to customers and the corresponding business models for their competitive differentiation [50]. Thus, ISQ assurance is the grounding asset to meet customers' expectations. Indeed, the ISQ performance enables greater systems' reliability and flexibility, improving users' satisfaction and technology acceptance [489].

Information Systems research dealt widely with the notion of quality, intended as the sum of the high-level constructs of Information Quality, Systems Quality, and Service Quality [125]. The Software Engineering (SE) and Management Information Systems (MIS) communities provide complementary perspectives, addressing IT usage in organizations. The point is that they emphasize on different activities and methods for controlling and improving ISQ. In fact, according to [466], the borderline between the quality of a software system and that of an information system is rather subtle, since software normally refers to programs whereas an information system is the organizational context in which software is used. Although several frameworks have been developed to evaluate ISQ [320, 364], we did not find recent papers with a focus on the IT-related characteristics. We gathered several concerns in these years about *low information systems quality* of an ever–increasing number of financial applications. However we did not find a comprehensive picture of the problem in literature. This chapter addresses this contingent research gap.

As a matter of fact, the motivation of this study comes from the industrial practice. Several domain experts raised concerns about critical issues, e.g., the growing complexity of the information systems, unjustified applicative layers and middleware stratification, difficult reverse engineering of their legacy software, and costs explosion. We started to explore effective techniques able to tackle the problems raised by the customer community. In doing so, we realized that a comprehensive model, to

address informants' concerns related to the quality of their information systems were still missing. Consequently, we started this journey to approach this research gap. This leads to major concerns about "what is inside" any financial information system.

Thus, we are interested to find suitable answers to the following research questions (RQs):

- RQ_1: What are the major IT-related concerns of the financial sector?
- RQ_2: Are these concerns shared among the community of experts of the IT financial sector?

In this chapter we do not highlight technical problems and solutions *as such*. Our aim is to provide a valuable model to analyze and understand this compelling issue for the financial community. We identified the most relevant IT quality factors, investigating the concerns of several stakeholders i.e., banks and outsourcing companies, system integrators, software vendors, and consultants. Our focus is industry specific. We have chosen the financial sector for the following reasons. Firstly, a longitudinal study would have been too generic, whereas we wanted to grasp in depth details of the phenomenon. Secondly, the financial sector is a traditional business, with highly complex legacy systems, which is facing a radical transformation due to market and regulation drivers. Finally, due to its industry structure, it is a quite homogeneous sector. In order to pursue the study we surveyed the opinions of two panels of experts. For the panels' composition, we used the well-profiled contacts of an established IT consulting firm. The panels included more than one hundred senior IT financial experts who are mostly in top managerial positions. At the end of the research process, we gathered results that we consider highly valuable and generalizable. We used an innovative research method, based on the epistemological paradigm of Mixed Methods research [112]. Finally, we report on the emergence of a meta model, which links software quality, architecture, and process. Our study may be helpful in understanding in depth and with high internal and external validity (due to the Mixed Methods approach) IT quality and problems related to systems evolution and maintenance. For this reason the first stage of our research is inductive, since we wanted to elicit the relevant items of concerns related to ISQ. We used a research methodology which merged inductive research (through Delphi) with a deductive one (survey-like), to provide a comprehensive analysis of the problem. In these terms, we integrated both forms of data collection within the same research. So, we embedded one form of data within another to analyze different types of research questions [112]. Finally, we mapped the induced factors into the ISO standards 25010, 42010, and 12207, which refer to the three categories of software quality, software architecture, and software process. We argue that the developed model is of actual use, since it provides an abstract and general reference for ISQ, grounded in empirical evidence.

The structure of this chapter is as follows. In Section 3.2 we pursue the literature review, highlighting the research gap. Moreover, we explain our motivation along with IT financial market similarities, to motivate our research journey and its generalization. Then, we present the research design and details of the Delphi–like study that we conducted, in Section 3.3. This is followed by a presentation of the results of our study in Section 3.4. In Section 3.5 we expose our findings, and present the SQuAP (Software Quality-Architecture-Process) model, as also implications for research and practice. The theoretical contribution of this study is discussed in Section 3.6. Finally, in Section 3.7 we conclude our chapter, discussing its limitations. Furthermore, we outline future research. Our main considerations are summarized after each section.

3.2 Literature Review

The definition of Information Systems is blurred and changed in time [210]. Information Systems referred to datalogical and infological systems [275], reporting and control systems [59], formal specified technical systems [455], inquiry systems [96], behavioral systems [129], socio-technical systems [326], and human-activity systems [89].

Generally speaking, "Information Systems research is to study the effective design, delivery, use, and impact of information technologies on organizations and society" [250, p. 3]. Accordingly, our research focused on quality aspects of information technologies, related to financial organizations, in the spirit of [466].

ISQ appeared to be from the first moment of our research journey a highly sensitive issue. This concern is also largely shared in literature. Typically, billions of dollars are spent in IT projects since their success is an important competitive advantage of any organization [180]. Scholars argue that IT quality is the most important success factor of information systems – "Software quality can determine the success or failure of a software product in today's competitive market" [458, p. 84]. The quality of the software developed for information systems is an important source of competitive advantages for contemporary companies [375]. An analysis of the Software Engineering literature highlights the importance of product quality "in use" for industries, which strategically exploit software functions. The available literature also largely supports the view that a critical success factor for most of the initiatives in a domain like the IT sector is the information available to guide and support the management of software quality efforts [181].

The importance of the three dimensions of software quality, software development process, and software architecture to manage and evolve information systems has been extensively reported in literature [377]. Accordingly, the three related communities proposed significant advances to structure, assess, and improve the respective domains.

Therefore, we will now analyze the most relevant literature of the Software Engineering and information systems research communities, highlighting the research gap in Subsection 3.2.5.

3.2.1 Information Systems Quality

In literature, information systems quality has been linked by DeLone & McLean to the concept of information systems success [124, 125]. It is composed of *system quality*, *information quality*, and *service quality*. System quality measures users' perception of adaptability, availability, reliability, response time, and usability [125]. Information quality relates to the content issue, and can be measured with the item completeness, ease of understanding, personalization, relevance, and security [125]. Finally, service quality measures the overall support delivered by the service provider with assurance, empathy, and responsiveness [125]. Subsequent literature explored and validated these constructs as key IS success factors [488, 153, 366].

However, these contributions strictly related to the individual perception and use of information systems. Their focus is on the *social* pillar, rather than the *technical* one [360].

In an epistemic view of complementary contribution of the two Software Engineering and Management Information Systems communities to address IT usage in organizations [466], also the technical pillar needs to be solid and studied. Consequently, in the next Subsections, we address such a research gap.

3.2.2 Software Quality

The most important reference for software quality is the ISO/IEC 25010:2011 standard on the quality of software products and their usage (in use quality). This standard is based on eight characteristics which qualify a software product and five characteristics which assess its quality in use.

The empirical validation of quality characteristics has been carried out in literature in several ways. Software quality assessment has been one of the first concerns of software-related literature [317]. A survey-based study of 75 end users and developers to assess the standard structure was conducted in 2004 with ambiguous results [239]. In 2007 a similar study lead to the same conclusions, which were helpful for the ISO/IEC 9126 standard revision [238].

The need to decrease software life cycle costs, at the same time enhancing software quality, is a very well-known issue, which leads primarily the research on this topic [64]. Since the quality issue impacts directly companies, one of the first comprehensive software quality assessment model (SQM) was developed by NEC [444]. They used the Goal–Question–Metric (GQM) structure to define the Factor (e.g., correctness), Criteria (e.g., traceability), and Metric.

Software Engineering literature had also growing expectation for integrated approaches to quality modeling. Scholars proposed to use meta models as a ground to develop a base quality model [472]. Notably, quantitative quality models have been developed taking into consideration several quality metrics, suggested by the standard itself [474], [473]. Nevertheless, several other software quality metrics tools have been developed [83], inspired by the ISO 25010 standard, introducing also new software quality related metrics, like SQUALE [322].

Management Information Systems literature gives a limited contribution to understand the software quality dimension. The quality construct relates usually to some definition of Product Quality [32], which we found is not very useful for assessment purposes.

3.2.3 Software Process

A structure for the software process life cycle has been defined by the industrial standard ISO/IEC 12207:2008, to outline the tasks required for developing and maintaining software [426]. Regardless from the development methodology chosen (i.e., Agile or Waterfall [377]), this standard includes all the relevant concepts of the life cycle. Therefore, it is a useful structured tool for software houses, to assess if they have undertaken all recommended actions or not. Nevertheless, the Software Engineering literature developed also other process assessment tools, e.g., Software Process Improvement and Capability Determination (SPICE) [136]. Derived from ISO 12207 concepts, SPICE has become a new standard i.e., ISO/IEC 15504. It was built on already existing software assessment methods, such as the Capability Maturity Model (CMM) by the Software Engineering Institute [356], TRILLIUM developed by the Bell laboratories in Canada [20], among the most known ones. Rather than depicting the concepts of a software process, CMM focuses on the maturity of the process itself. In other words, it provides a framework to assess a process through six capability levels, ranging from 0 (*ad hoc*) to 5 (optimizing). So, it is possible to evaluate an organization's capability to deliver software artifacts through an objective assessment. *Ad hoc* processes are occasional and chaotic, without any proof that any kind of process quality was guaranteed; on the other side of the maturity spectrum optimized processes are continuously improved through innovation, benchmarking, simplification, controlling, and change management [149].

However, the ISO 15505 is not a static framework. Several advancements have been proposed with the successor of the CMM, the Capability Maturity Model Integration (CMMI), which aimed to improve the usability of maturity models by integrating many different models into one framework [454]. Although SPICE is considered to be the leading reference for process assessment, several other frameworks have been proposed by scholars [461]. Still, the use of these other frameworks in industrial environments is doubtful [461].

Management Information Systems researchers do usually not focus directly on software process as a model, rather as a variable to understand other factors (e.g., Competitive Performance [338] or Project Performance [339]). High level constructs like process flexibility [353] and process predictability [73] are used to define software process, derived per analogy from the Software Engineering and manufacturing literature [339]. Nevertheless, there is a clear idea that the quality of software development processes are crucial for the performance of organizations and their competitive advantages [340]. Other scholars take as a reference the CMM model and try to build on that a higher model [123] to assess a project performance. Generally speaking, MIS literature sees the software development process as input for e.g., Project Performance rather as an outcome of different characteristics, like the Software Engineering literature does.

3.2.4 Software Architecture

Also the dimension of software architecture has a reference standard, ISO/IEC 42010:11. This is basically a *glossary* where the most relevant terms of software architectures are defined and put in context. Traditional Software Engineering literature is focused on the well–known suitable patterns for software design [424]. Hence, typical research in this domain is about how architectural patterns and guidelines impact software components and configurations [167]. A survey study analyzes in this perspective architectural patterns to identify potential risks and to verify which quality requirements have been addressed in the design [134]. With regard to software architecture evaluation, intended as a way to achieve quality attributes (i.e., maintainability and reliability in a system), some approaches have emerged, mainly ATAM proposed by the Software Engineering Institute [249, 105, 47, 44]. More recently an approach based on quality requirements has been introduced, in order to offer guidance on the choice of the most appropriate method for an evaluation [27, 316]. Similarly, also the practitioner's communities developed their frameworks, see for instance [4]. However, none of these techniques are related to the ISO 42010 standard. Apparently, the debate about the assessment of software architecture is less mature compared to the software quality and process domains.

From an Management Information Systems research perspective, several advances have been made in literature regarding Enterprise Architecture Management assessment [11, 487]. Such a research stream focuses on the the activities carried out in an organization to install, maintain, and develop the Enterprise architecture, to deal with its different architectural layers and foster a holistic and integrated view of the enterprise IT architecture [11]. This approach studies the interaction between the technological components of the information systems and the organization, to achieve common business objectives [241]. While this framework defined initially a quite conceptual perspective [69], recently empirical studies are gaining momentum [487]. High level constructs (i.e., Product Quality, Infrastructure Quality, Service Delivery Quality, Organizational Anchoring, Intention to Use, Organizational & Project Benefits) are empirically assessed [274] to depict the level of IT-business alignment [295].

3.2.5 Relationships Among Dimensions

With regard to the mutual relationships of the three dimensions of software quality, process, and architecture we found scarce empirical literature. Most papers refer to the *obviousness* of the interaction of these dimensions, providing little evidence for such statements, like "[...] it would be extremely unusual to find a high quality software system with a poor design" [303, p. 471].

Process and Quality

Software quality and process are among the most studied issues. The very idea itself of process maturity is closely related to quality, one of the most relevant issues studied by the Software Engineering Institute at Carnegie Mellon [221]. How we have seen before, such research efforts developed the Capability Maturity Model [356] and its evolutions [95]. This kind of interaction is also the most evident. In fact, recent literature lets emerge empirically this kind of relationship [405]. Similar studies noticed how this kind of relationship is magnified with a Continuous Development methodology [393]. In models like CMM, architecture is considered exogenous. It is given for granted, thus the model focuses on the development process to enhance quality. Moreover, it has also to be noted that the effort to structure software development in order to be in control of its quality went back to the 1960s by the US Department of Defense (DoD). Accordingly, SEI was started up at Carnegie Mellon by the US military in the 1980s, where the first version of CMM was published in 1988 [221]. The first architecture standard IEEE 1471, on the other hand, was issued just on August 1995. The reason might be that software architecture, as a discipline started to rise years later with respect to process and quality, which were considered more important.

Process and Architecture

The relationship between software process and architecture has been studied in several ways, mainly as a consequence of the process model. According to this view, the architecture supports a structural decomposition of the development cycle into tasks, and the decomposition continues until each defined task is performed by an individual or single management unit [221]. The idea is still that of an exogenous element. Similarly, iterative [61] and non-iterative [398] software process models consider the software design as an element of the process. Notably, with the Twin Peaks model [344] software process and architecture are on the same level, since one influenced the other. It is also the theoretical ground for Agile architecting [104]. Consequently, with the Agile movement, the differences between the development phases narrowed down [209]. So, software process and architecture were considered as one comprehensive issue [26]. However, recent studies show how this idea was poorly adopted by the Agile developers, since none of the architecting approaches has been widely used in combination with Agile practices [495].

Architecture and Quality

Regarding the relationship between software quality and architecture, some theoretical contributions were made to understand how an architecture is inspired and even driven by quality concerns [249]. A good reference for this relationship and its value in practice is Bass *et al.*[38]. A recent book highlights the importance of software quality analysis for software architecting [316]. Still, this remains a relationship actively in

evolution, especially in the context of the novel proposal of DevOps processes for cloud and microservices architectures [39].

Quality – Process – Architecture

At our best knowledge, no comprehensive model to study information systems as the triple interaction of software quality, process, and architecture has been developed in literature, identifying a clear research gap.

In the last decade, there has been a considerable effort, especially by the Management Information Systems research community, to study the phenomenon of the alignment of business and information system's architecture [8]. What emerged is the importance of such alignment for both business' competitiveness and technical efficiency. In fact, when it comes to integrate new solutions, modules or interfaces such alignment is of key importance. Several other scholars found similar results, suggesting the importance of standard governance defining key architecture roles, involving key stakeholders through liaison roles and direct communication, institutionalizing monitoring processes and centralizing IT key decisions [65].

Especially in the financial sector, architectural governance is a key issue for IT efficiency and flexibility [416]. Generally speaking, this finding is also largely shared beyond the financial sector [274]. The need for people from different backgrounds (mainly business and technical ones), to align the organization is the greatest insight of this research stream. This pattern has been firstly theorized by DeLone & McLean [124] and then revised in 2003 [125] by the same authors. A recent literature review shows how in over 90 studies this pattern has been empirically observed as successful [364].

In this regard, also special tools have been developed to monitor the alignment and co-evolution of business process and enterprise software systems to estimate the change propagation caused by a change request in business processes or software systems based on the software architecture and the process design [396], [395].

3.2.6 The IT financial market

Information systems effectiveness in general, and quality in particular, has always been considered by MIS literature a key strategic issue for the financial sector [310]. We recognize that key determinants of banking service quality, which includes ISQ have been proposed by literature [237]; however, a general framework for ISQ of this crucial sector is missing.

To improve reader's context understanding, we pinpoint the two most relevant homogenization drivers of this sector: market and regulation.

From a market perspective cost cutting needs are the same at least for EU and US banks due to low interest rates level set by the European Central Bank (ECB) and the Federal Reserve (FED) and insolvency issues in the mortgage market. This reduces sensibly bank's margins and profitability, leading to generalized internal cost cutting. The maintenance and evolution of information systems of financial organizations are influenced by such business goal by delivering with lower budget (and time, due to fast regulatory changes). Same custom applications and software packages are implemented in European banks because software vendors sell standardized industry solutions. COTS (Commercial Off-the-Shelf products) market is quite similar wordwide since it offers industry solutions to get benefit from the low marginal cost of software [116]. Typically, if one competitor offers one new application, also others are willing to follow in order to avoiding the lose crucial market shares. Finally, banks are

among the most globalized businesses, so same information systems are shared among more branch countries. For this reason they share the same problems.

From a regulatory perspective, similarities become even more clearer. Currently, the most important financial regulations are standardized within the European Union, e.g. MiFID, and the European financial Authority is coordinating the national authorities, so the regulation differences among countries are decreasing. IT concerns among banks are quite similar, since most regulation is provided by shared European financial regulators. More precisely, all the efforts in the direction of a European financial framework are pushing information systems of financial organizations towards new issues, like the Payment Service Directive (PSD 1) [2]. Since the goal is to integrate a Single Euro Payments Area (SEPA), major efforts are asked to European banks and their information systems to interoperate properly, leading also to new financial business models. In this regard, the Payment Service Directive 2 (PSD 2) is offering novel architectural challenges, since it will provide third parties, through bank's APIs, to establish new payment channels with the customer's account, breaking the traditional monopoly of the the bank over its own channels [3]. In other words, third parties will have access to a customer's account via APIs to connect a merchant and the bank directly. This will dramatically impact in the near future the financial information system architecture, leading to new payment service models [432]. On this regard, the European financial Authority (EBA) recently issued the *Regulatory Technical Standards on strong customer authentication and secure communication under PSD2* [467] for enhancing the security level of payment services across the European Union. These Technical Standards, as other regulations, compel banks of the Euro Payments Area to comply with uniform protocols and procedures of their information systems to support the *Free Movement of Capital* EU principle, according to articles 63 to 66 of the Treaty on the Functioning of the European Union (TFEU).

Moreover, since banks are *de facto* global businesses as their mission is to operate world-wide (e.g., global wire transfer), they have to comply with common standards. The Bank for International Settlements (BIS), whose purpose is the collaboration among central bank authorities is the most relevant organization which supports global financial standardization. BIS also supported the development of crucial industrial standards for the financial sector. Just to cite a few, we remember the ISO 19092-1 Financial Services - Biometrics - Part 1: Security framework, ISO/IEC 15944-8:2012 Information technology – Business operational view – Part 8: Identification of privacy protection requirements as external constraints on business transactions. Those standards have been developed to support modeling international requirements for identifying and providing privacy protection of personal information throughout any kind of information and communications technology (ICT) based business transaction.

3.2.7 Systems' scope

When we refer to financial information systems, we mean all systems which support financial operations in different market segments (e.g., retail, corporate, private, portfolio management) and different products/services (e.g., deposits, payments, loans), internal governance (e.g., risk management, communications), internal administration (e.g., accounting, balance sheet), internal support processes (e.g., passive cycle, asset management, real estate management, HR) and regulation requirements (e.g., anti-laundering communications).

This list of applications and services is huge. Typically, these information systems are composed by millions of lines of code, built along several years with different technologies. For instance, even today an important part of the applications are

developed in COBOL, CICS, or DB2 in IBM mainframe environments, due to their robustness for financial purposes.

Financial information systems are usually organized in different functional areas:

- Guidance systems: they centralize and aggregate all customers' and banking data, like clients' registry, contracts, products, conditions, credits, and guarantees.
- Channels: they run the front-end system for the management of ATMs, call center, phone banking and internet banking. They act as intermediaries between customers (retail, private and corporate) and the bank. Thus, channels allow typical banking operations e.g., execution of transactions. They take also care of sales force automation and customer relation management systems.
- Legacy systems: they manage various types of relations, contracts and operations, like deposits, securities back office and payment systems.
- Finance area systems: they manage the different bank's positions in securities and derivatives exposure.
- Control systems: they support management controlling instruments, like risk management and strategic marketing.
- Support systems: such systems support the passive cycle, assets, real estate, staff, HR and general purpose support applications.

3.3 Research Design

As researchers, we have our epistemological bias, which usually remains hidden or implicit, even if they deeply influence our research [428]. Since we approached this research, we decided to use *Pragmatism*. It derives from the work of Peirce, James, Mead, and Dewey, and arises out of actions, situations, and consequences rather than antecedent conditions (as in postpositivism) [90], [37]. It focuses on the research problem, rather than the method. This is the philosophical ground for the Mixed Methods approach.

These four scholars paved the epistemological ground of our approach. Although with different point of views, they complemented the pragmatic approach.

According to Peirce (1839-1914), human concepts are defined by their consequences [358]. So, people's independence of will to decide the actions to undertake is crucial in any experimentation. Ideas and concepts are the main drivers for truth.

On the other hand, James (1842-1909) highlights the pluralistic view which recognizes the multiplicity of human truth. Due to his psychology background, he argued that human thought may be revealed only in action.

The Logic of Controlled Inquiry was proposed by Dewey (1859-1952) [127]. He argued that reasoning by itself can not provide change. Only the combination of action and reasoning reorders the setting.

Finally, Mead (1862-1931) suggests that any human action is socially reflective [302]. Any human behavior that elicits responses from another individual constitutes a social act. The social consciousness is the reflection of ourselves, mirrored in the reactions of others.

The pragmatic view does not commit to any system of philosophy and reality. Like Mixed Methods, both quantitative and qualitative assumptions are used. This

is related to the view that the world has an absolute unity. Truth is what works at the time, it is not based in a duality between reality independent of the mind or within the mind [112]. Therefore, in Mixed Methods research, investigators use both quantitative and qualitative data because they aim at the best understanding of a research problem [112].

This chap reports the results of a Delphi-like study modeled on the Delphi methodology (qualitative) and survey (quantitative). The first research phase has been devoted to address RQ_1, to let emerge the most compelling IT quality concerns. In the second phase, the elicited concerns were validated by the expert community at large, to give an answer to RQ_2. In order to investigate the concerns we used a phenomenological approach [324]. Moreover, the survey is a semi-structured one so, beside closed questions based on an hybridized-Likert scale, experts are asked to express openly their opinions about any single item. Thus, the survey itself is mixed quantitative and qualitative.

Generally speaking, empirical Software Engineering is developing new research designs according to its research questions, which may cross the threshold of traditional borders of the discipline due to the pervasiveness of software [102], [99]. The Delphi method is becoming a popular tool in the Software Engineering discipline [263], even though it is still not well–known. On the other hands, Likert–based surveys are a popular quantitative research method in information systems [94]. We first present a brief discussion of the Delphi method, since it is the underpinning method of this research. Then we discuss how the Delphi panel was selected, and provide details of the Delphi process i.e., threats validation and the survey one. Finally, we discuss the outcomes.

3.3.1 The Delphi-like Method

The Delphi method has proven to be a popular tool in Software Engineering [263] and more in general in Information Systems research [74], [418], [196], [325], [418]. It allows to capitalize the experiences of the expert panel in identifying key issues of software developers, identifying the most important factors by continuous feedbacks. The objective is to create valuable information through a structured process of knowledge collection from a panel of experts with controlled opinion feedbacks [135]. The process consists of a series of rounds in which each expert communicates his opinion through a structured form i.e., questionnaire, structured interview, collected by the researcher. It is an inductive data–driven approach, ideal for explanatory studies for which little empirical evidence is available [188]. Using a step–wise methodology the research question is narrowed down, from a multitude of issues, to a bunch of a few consensus-based factors [417].

After gathering experts' concerns on the IT financial sector we consolidated it into 28 factors (RQ_1) with a Delphi style methodology. Then, we collected opinions and evaluations about the factors with a "target-panel" of 124 profiled experts (RQ_2). The administration of the whole process is represented in Figure 3.1.

3.3.2 Selection of Delphi Panelists

We are aware that the selection of panelists is the most critical aspect of Delphi, especially with respect to the validity threats: a lousy panel selection may compromise the whole qualitative research.

Thus, we mitigated this risk exploiting an established private dataset of an affirmed consultant firm; moreover, the use of a double panel using Mixed Methods enforces

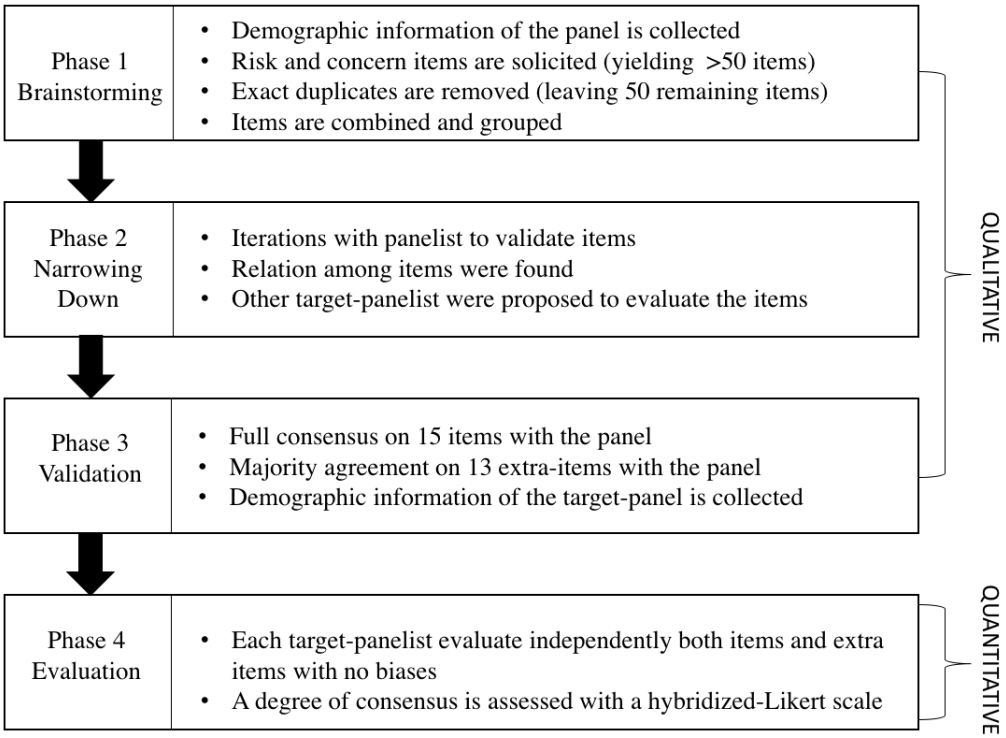


FIGURE 3.1: Delphi–like administration process

the validation of both the questions and the answers. However, it still remains *the most important yet most neglected aspect of the Delphi method* [347]. Methodological literature agrees upon the fact that the choice of experts is the single most difficult factor in panel selection [236].

Hence, our greatest efforts were devoted to the selection of the first and then the second panel. Since we intended to perform a vertical study (i.e., sector specific), we put a lot of care to the choice of experts invited to participate to the panels.

We started from a privileged position. In fact, we could use an expert pool of an established IT consulting firm, specialized in the financial sector, which works with all main financial groups. So, we were able to address personally highly qualified experts from the IT financial industry. This makes our panel highly reliable and representative.

The first panel was chosen using a stratified random sampling. Strata were defined upfront, to define the sample population. Companies and roles were chosen to have a fair representation of the IT financial sector.

The first panel did not mirror the population representation task regarding experience, since we addressed explicitly senior experts, in agreement with [236]. Also the target-panel was chosen with a stratified random sampling, but inside a larger pool.

We asked panelists to give their opinions about the sector composition (companies and roles) and also the name of other experts. Therefore, we adjusted the sample population representation along with our research journey, following a suggestion found in [112]. However, also for the target-panel we looked for more senior opinions, adapting the research method to our purposes [113].

In Table 3.1 we describe the demographic composition of the first panel and in Table 3.2 that of the target-panel. We profiled our panels in great detail, much more than traditional Delphi studies. The guarantee of anonymity we gave to all experts allowed a very open and truthful discussion.

		#	%
Company	Consultants	7	25%
	Bank	6	21%
	System Integrator	6	21%
	Outsourcing	5	18%
	SW Vendors	4	14%
	Total	28	100%
Experience	11–20 years	4	31%
	21–30 years	8	62%
	More than 30 years	1	8%
	Total	13	100%
Role	CEO/CIO	6	29%
	Chief Data Officer	5	24%
	Appl. Maint. Group Exp.	4	19%
	IT Architect	4	19%
	Maintenance Manager	2	10%
	Sales	0	0%
	Total	21	100%

TABLE 3.1: Panel composition

We took into consideration three dimensions: (i) years of experience, (ii) sector in which the experts worked, and (iii) relevant roles they served in their careers. Obviously, sector and role may be more than one per expert, since it is normal to change job during any career path. In detail, these are the dimensions and sub-dimensions surveyed:

Experience. For the Panel composition, we used only senior informants. Thus, we had three groups, composed by experts with more than 10 years and less 21 years, between 21 and 30 years and more than 30 years of experience. We divided the experts into five groups in the target–Panel: less than 5 years, between 5 and 10 years, between 10 and 20 years, between 20 and 30 years, and more than 30 years of experience. The greatest majority of our experts have more than 20 years of experience in the sector. Thus, we consider the opinions of our panel of high value.

Companies. We profiled also different company types, involved in the IT financial sector. Most experts declared some experience as interns within a *bank*. The companies second by relevance are *system integrators*, since a high degree of customization is needed for bank products. *Consultants* and *Outsourcing companies* are also very important actors for the IT financial sector, since most work is outsourced to external workers or companies. *Software vendors* deliver software packages products for banks. As stated before, software packages are commonly used by small/medium sized financial institutes to either address specific needs (e.g. loans, deposit or transaction accounts) or support common processes (e.g. accounts payable and receivables). According to our experience, the composition of both the panel and the target-panel represent a trustful representation and distribution of companies in the IT financial sector.

Role. (*CEO/CIO*) The Chief Executive Officer is the top manager of software integrator or vendor companies, while the Chief Information Officer is a board member, in charge of the information system of the bank. The *Maintenance Manager* is in

		#	%
Company	Bank	72	35%
	System Integrator	41	20%
	SW Vendors	34	17%
	Consultants	33	16%
	Outsourcing	23	11%
	Total	203	100%

Experience	0–5 years	8	6%
	6–10 years	1	1%
	11–20 years	23	19%
	21–30 years	72	58%
	More than 30 years	20	16%
	Total	124	100%

Role	Appl. Maint. Group Exp.	59	39%
	CEO/CIO	32	21%
	IT Architect	18	12%
	Chief Data Officer	18	12%
	Maintenance Manager	13	9%
	Sales	8	5%
	Other	2	1%
	Total	150	100%

TABLE 3.2: Target-Panel composition

charge of the development and maintenance of IT financial systems. The *Application Maintenance Group Expert* is responsible for complex groups of IT financial applications. The expert in charge of the IT financial architecture is the *IT Architect*. *Sales* is a senior manager of the sales department of a software vendor or system integrator. Finally, the *Chief Data Officer* is an expert in industry standards and methodologies (e.g., IEEE, ISO, ITIL, DAMA) and in charge of the bank's data governance. Also the distribution of roles within both the panel and target-panel is representative of the IT financial sector, at our best professional knowledge.

The methods we used to compose the two panels varied slightly. For example, we took into consideration for the target-panel the role of sales managers, not considered before. We had feedbacks that their role is also critical for the IT financial sector, so we adjusted it along the research process (following [112]). Senior experts were chosen for the first panel, with at least 10 years of experience within the sector. In the target-panel we wanted a wider and trustworthy representation of the sector, so we included also junior people in the target-panel. However, about two-thirds of the target-panel had more than 20 years of experience.

Most of our panelists and target-panelists experienced more than one position in more than one company. We did not find it meaningful to show just e.g., their last job, or the longest one. Experts expressed their opinion according to their general knowledge of the domain, which was gained through different professional experiences. This enriches the research, since experts were able to judge the factors from different, highly qualified, viewpoints within the same sector. This complies with our epistemological paradigm and research approach.

3.3.3 Data Collection and Analysis

The Delphi–like study took over a year for the four phases (see Figure 3.1), which are described next. According to the Mixed Methods approach, this research includes both qualitative and quantitative approaches. In order to develop the qualitative part the first three phases focused on identifying the relevant items. Then, we used a survey-based approach to validate the panel's items by a wider panel (target-panel).

The qualitative research started in October 2014 and lasted until April 2015. The survey took less time, from June 2015 up to November 2015.

Phase 1: Brainstorming. After collecting the demographic information from our pool, and composing the panel, we conducted a brainstorming round to elicit as many concerns as possible. This phase was helpful to broadly understand the problem and seek concerns. It was an unstructured process, where more than 50 items were solicited. Afterwards, exact duplicates were removed, leaving 50 items. Finally, items were combined and grouped.

Phase 2: Narrowing Down. In the second phase, further iterations involved the panel to validate the items. Relations among items and their grouping were discussed, again in an unstructured way. Experts were asked to discuss the grouping.Items were represented in sticky notes on a blackboard. This helped the discussion over the items and their relations. Afterwards, we wrote down the items list, in the terms discussed by the panel. Finally, the panelists discussed about the target-panel composition and also proposed other experts to invite.

Phase 3: Validation. The goal of the third phase was to validate the items. The panel considered more than 30 concerns but it came finally to an agreement about the 28 concerns we will discuss in this chapter. Since we were able to have have all our panelist within one room, there was no need to use statistical techniques to assess their level of agreement. Typical Delphi studies, where informants are distributed, ask panelists to rank concerns according to their importance and then aggregate them through e.g., Kendall's coefficient of concordance (W) to assess the degree of consensus among the panelists themselves [417]. We had a narrow group of highly qualified experts and managed them as a working group. The introduction of statistical methods for the validation of the final items were not grounded in our phenomenological approach. We wanted to grasp the essence of their experiences as described by the participants [324]. At the end of this phase a full consensus was reached on 15 items. The panelists did not always agree on the degree of such consensus (i.e., *strongly agree* or just *agree*). However, the baseline (i.e., *agree*) was always met for each item.

There were then 13 more items which did not reached full consensus but were strongly advocated by some panelists. Although both brainstorming and narrowing down phases involved a lot of personal confrontation, these 13 extra items were either advocated or opposed by at least one panelist. While some panelists supported one item, some others argued that for their respective industrial segment the proposed item was not relevant. Therefore, during the validation phase, there was a consolidated opinion about the first 15 items and mixed feelings about the 13 extra ones.

We were interested to evaluate also such 13 extra items. We included them in our survey since we wanted to evaluate if the panelists' opinions were shared by the financial community. Finally, some demographic information about the target-panel was collected.

Phase 4: Evaluation. The last phase concerned the quantitative inquiry approach, to evaluate the concerns by a larger stratified sample group (target-panel). We prepared an on-line survey with the concerns and made personal invitations to experts. Target-panelists could start the survey only after they inserted their personal

credentials, given by e-mail or phone. To have control over the research and the randomized stratified sampling, no answer was anonymous. We used an hybridized-Likert scale for the concerns' evaluation. The aim was to compute semantic differentials (i.e., "Strongly Agree", "Agree", "Disagree", "Strongly Disagree") to define the level of agreement, typical of Likert scales [283]. We hybridized the Likert scale with even bipolar values (negative and positive), symmetric to 0, without an average effect. To stress possible differences, we gave higher values to both extremes, also to avoid an average effect. So, we assigned the following values: "Strongly Agree"= 3, "Agree"= 1, "Disagree"= -1, "Strongly Disagree"= -3. The idea was to highlight different semantic values of the two extremes and suggest equidistance between the center point of the scale and the two extremes. To overcome the bias that can result from the order items were presented, we randomized the questions and did not tell to the participants about the difference between items and extra-items.

3.4 Results

The Delphi–like study resulted in a set of 15 concerns presented in Table 3.3 along with the target-panel evaluation. All concerns were shared, at least, by 70% of the target-panelists with a different intensity degree (measured with the hybridized-Likert scale). In Table 3.4 we represent the 13 extra-items. Interestingly, all these extra-items have a lower share degree than those in Table 3.3. The reason may be that the mixed feelings of the first panel was also shared among the target-panel. This generally confirms that a high consensus degree of a Delphi panel reflects also an eventual validation. However, we found the concerns which emerged from Table 3.4 of interest and show them separately.

To improve our theorization, we abstracted the explicit informants' concerns into corresponding factors. We are aware of the fact that dealing with 28 factors provides a huge amount of knowledge and is not really parsimonious. However, we only dealt with unique items. This means that for our panelist the 28 concerns had to be treated differently. Thus, to be consistent with our research design, we had to managed them separately. Finally, according to [156], in order to broaden the generalization, we pursued a mapping of the concerns into high-level constructs proposed by industrial ISO standards.

Each factor is summarized followed by a short discussion where panelists' and target-panelists' opinions are directly quoted. For the sake of readability, only in this section, we use the terms panelist, target-panelist, informants, and experts as synonyms.

Factor 1: Module interfaces complexity.

A financial information system is characterized by a high number of modules; if these are strongly coupled this increases the number of interfaces and their complexity.

One panelist mentioned the spaghetti like architecture stating that *"we are experiencing a growing complexity in delivering new projects due to past implementation of point-to-point architectures delivered in the last years"*. Integration costs and higher complexity compared to a green field makes the business case less effective. Even worse, another one said that *"sometimes the interfaces to be updated are so complex that an ad hoc middleware is required"*. The lack of knowledge seems to be another root cause since a third panelist said *"I believe that the lack of knowledge about application functionalities led to code duplication over the years"*.

Factor 2: Interfaces architectural complexity.

#	Concern	"Agree" and "Strongly Agree"	Average score
1	Software modules interfaces are characterized by a high level of complexity and represent an important part of each module in terms of number of objects and LOC.	96%	1,95
2	Interfaces among modules are developed in a stratified way in time. This brought to a complex architecture hardly manageable and not future-proof.	96%	2,47
3	Software quality for custom development is decreasing in the last years (especially Cobol / CICS / DB2).	77%	1,15
4	SW Maintenance & enhancements evolution costs and time of information systems are increasing due to the (i) stratification of software, (ii) poor documentation and (iii) low quality of the source code.	82%	1,58
5	Low software quality depends on increasing pressure for enhancements evolutions in shorter time with lower budget.	79%	1,57
6	Low software quality depends on a poor level of functional and technical analysis and detail.	79%	1,00
7	System analysis is hindered by an inadequate documentation and database.	87%	1,70
8	It is difficult to build and maintain effective documentation because of low budget and time shortening.	84%	1,46
9	New software packages have more functionalities than in the past but their increased complexity leads to difficult evolution management.	83%	1,58
10	Software packages are poorly documented for effective maintenance and evolution.	82%	1,30
11	Software packages documentation main gap is due to a poor description of the data managed by the system (i.e., not only the record layout, but also the fiscal and logical data model, the data dictionary.).	77%	1,09
12	"Application Maintenance" contracts do not improve the software documentation.	77%	1,15
13	International software applications are of higher quality but are not *per se* more maintainable.	76%	1,29
14	Domestic software applications are characterized by more functionalities and lower quality without any significant impact on software maintainability.	75%	0,92
15	Software quality can not be reliably measured through tools and methodologies	70%	0,93

Table 3.3: Concerns and results of the Delphi-like study

#	Concerns	"Agree" and "Strongly Agree"	Average score
16	Lower software quality is related to decreasing skills of IT professionals.	48%	0,04
17	The use of external developers increases the difficulty to assess developers' skills.	66%	0,71
18	Software Engineering methodologies and tool to assess software's quality are not reliable.	70%	0,93
19	It is hard to agree with software vendors on Software Engineering methodology to enhance quality and to assess it.	66%	0,57
20	Lower IT professionalism depends on the poor use of Software Engineering methodologies due to shrinking IT budget and time.	72%	0,94
21	Lower IT professionalism depends on decreasing professional rates.	58%	0,62
22	Web technologies are developed with fewer Software Engineering rigor, also misinterpreting the Agile paradigm.	70%	0,85
23	Low software quality depends on unclear user requirements.	72%	0,86
24	Unclear user requirements depend on poor business & IT elicitation processes.	63%	0,75
25	Unclear user requirements depend on different "jargon" of IT & business departments.	58%	0,59
26	Poor data analysis (in terms of data modeling and data structure) influences directly the overall functional analysis.	70%	0,93
27	Fine-granular functional analysis is hindered by poor data modeling and data structuring.	73%	0,98
28	There are no effective software documentation methodologies and tools.	61%	0,36

Table 3.4: Extra-concerns and results of the Delphi-like study

This second factor is a direct consequence of the first one. Module interfaces complexity led to a typical anti-pattern [79]. According to a panelist, *"stratified software interfaces affect old developed applications; only using Service Oriented Architectures we took advantage of the strong benefits related to an integrated architecture"*. Legacy layered software is a remarkable problem as stated by another panelist *"there is a well–known problem related to platform software, which was developed years ago and never replaced. Only a tactical update with a short run perspective"* was carried out. Continuous update of legacy layered software seems an attractive and affordable way, but long-term sustainability is questioned. Most concerns were related to the maintainability of such architecture. No refactoring solution was seriously taken into consideration, due to cost. However, this short-term view did not decrease costs because it is very likely that the system will stop working properly in a medium period and the replacement cost could be very high. Moreover, this leads also to unexpected problems which usually rise with such complex systems. A third panelist said *"we decided to replace the client data module and its interfaces, since we reached a point where we were unable to manage the evolution required by the business"*. Due the high degree of coupling of these modules (Factor 1), proper reverse engineering is required.

The next factors will show how the lack of documentation hinders reverse engineering. All these elements make it difficult to improve the bank's information system.

Factor 3: Custom software quality.

Apparently, the quality of custom software applications is decreasing. Moreover, several modules were developed with old programming languages like COBOL, which are still widely adopted in the financial industry, while there is a lack of junior experts because such languages are not included in the current formal IT education programs.

A panelist explained that *"the number of developers able to develop in COBOL is rapidly decreasing due to retirement. New developers are not skilled enough with COBOL and other mainframe languages because they are focused on the newer languages like Java"*.

The interaction among old and new coding paradigms is another point raised by the panelists. One said that *"software quality is getting worse because we developed using a stack paradigm, adding new software adapting layers on old software. Unfortunately, this paradigm prevents the use of new technologies"*. Furthermore, the decrease of quality *"is perceived also in all other used programming languages"* according to other panelists. This may also depend on the actual training of developers. According to one expert *"several developers we hired did not go through a formal computer science education"*. This may be due to the high demand on the job market of software developers who however get low salaries. Skilled developers have usually many job offers and tend to choose the most profitable one.

Factor 4: Increase of maintenance costs.

Some factors have a direct impact on maintenance costs. The overall architectural complexity, the decreasing software quality and incomplete documentation are the most important drivers of high maintenance costs and time. As declared by a panelist, *"during the last six years our software modules have been impacted by a deep reengineering project and the most heavy effort was related to building new documentation"*.

Another reason of the growing costs seems to be linked to skills: a panelist pointed out poor competences as primary cause of frequent module rebuilds that cause an increase in the application complexity. This is related to the use of different technologies *"it often happens that instead of updating the software, new applications are built on it. The coexistence of different technologies is an important cause of high maintenance costs"*. According to one panelist *"most costs are related to continuous*

regulatory changes requested i.e. by the ECB". This is, apparently, another element of stratification and architectural complexity.

Factor 5: Quality vs. Time & Budget.

The whole panel agreed unanimously that there is a direct relationship among quality and time and budget. One expert stated that *"a relevant cause of poor software quality is related to time constraints, these aspects have impact on quality"*. More time and budget to develop and evolve software properly would increase quality and decrease maintenance costs. In fact, *"an already well-written code could be evolved with low budget, while a stratified software which has been poorly written makes updating difficult, increasing costs and time"*. This is a chain-effect, one said that *"poor software quality is related also to software stratification due to low investments and the obsolescence of the information system. Poor implementation of Software Engineering methodologies due to low budget magnifies this crucial issue day after day"*. Another element which emerged is the relationship between the organization structure of banks and its impact on the information system. One panelist stated that *"it is of high relevance the organization structure of the customer to define the proper information system"*. Apparently, this element impacts on the quality of the system itself.

Factor 6: Quality vs. System analysis.

Even though the design phase is perceived as the most important up-front activity, it is poorly implemented. One panelist stated that *"it is necessary to invest in this phase, to get benefits within the whole life-cycle"*. However, *"it is poorly carried out, due to shrinking budget and time"*, said another informant. The problem may depend on the fact that often the role of IT departments is not perceived as highly critical by top management. So, *"there is not an adequate collaboration between business functions and the IT departments"*. Therefore, system analysis activity is often skipped because it is hardly tangible. Stakeholders are not willing to invest in some activities where they do not see an immediate return. This leads to long term problems, as identified before. Another element is the that the formalization of the business requirements and a complete and effective vision of the information system is difficult. In this regard, one panelist affirmed that *"customers usually give poor and not coordinated requirements, leading to silos-like solutions instead of a fully integrated development"*.

Factor 7: System analysis vs. Documentation.

Inadequate documentation impacts on the system analysis and so on software quality. One expert stated that *"documentation is inadequate to the scope, being or too technical or too business-like with poor information abut the system"*. Moreover, a wrong interpretation of Agile methodologies results into poor documentation, in fact *"along with the stratification problem the misleading interpretation of Agile led to a poor and ineffective documentation"*. It is important to underline that the lack of data models' documentation and metadata definition *"leads to an inadequate and dangerous system analysis"*.

Factor 8: Documentation vs. Time & Budget.

Time and budget constraints have a direct impact on software documentation. Due to low budget for new developments and urgency for new applications, documentation is the first element which is skipped. A panelist said that is impossible to keep documentation aligned with software both for the frequency of software update and the lack of methodologies used to develop and manage software. According to another panelist, *"constraints on project costs and time causes poor or no documentation. In fact, the documentation is usually delivered only if you have enough budget"*. Due to budget limitation, often banks prefer to skip documentation if they are offered a discount on the application cost. Also time plays an important role. Often, there

is no time for documentation or, even worse, it is perceived as a waste of time. A panelist explained that *"budget constraints clearly impact on documentation, since we are not able to justify the budget required to keep documentation aligned. The only affordable way is to insert control points on the software development process (SDLC) to define the least amount of information necessary for maintenance"*. In this regard, documentation tools appear to play an important role. Another panelist declared that *"we can limit the problems that we have in software documentation thanks to the adoption of new generation tools (thanks i.e. to metadata) and the Agile approach"*.

Factor 9: New packages functionalities vs. Complexity.

For the reasons analyzed before, the demand for more functionalities rose in the last years, along with their complexity. Moreover, software vendors do not always apply industry standards. One panelist explained that *"new software packages require methodologies and standards that only few big vendors can adopt. In this situation when we want to customize those packages we must rely on those big vendors but with a low degree of control and the danger of lock in"*. Some experts also said that *"recent packages are too complex and have lower quality than in the past"*. This factor explains the relationship between the market trend i.e., more functionalities with architectural complexity (factor 2) and dependency on vendors (factors 10-12).

Factor 10: Packages vs. Documentation.

The lack of documentation for software packages is perceived as a *"commercial strategy of suppliers to lock-in customers"*. This appears natural, since *"the development is often given to software houses"*. For one expert *"documentation is always lacking"* and it is not uncommon to *"buy packages without any kind of technical and functional documentation"*. Another reason may be the fact that *"suppliers tend to hide technical documentation as IPR protection strategy, delivering only the functional documentation"*. However, as another expert said, *"this problem needs to be tackled with a good Service Level Agreement"*, according to an expert.

Factor 11: Packages documentation vs. System analysis.

The lack of documentation in packages impacts directly on the logical data model and quality controls. As stated by a panelist *"information about data is one of the biggest problems, as well as the role that data plays on business lines"*. Moreover, *"process logic is needed to achieve a correct level of documentation. Just data are not enough"*. Also this factor suffers from low budget and scarce time.

Factor 12: *Application & Maintenance* contracts vs. Documentation.

Application & Maintenance (AM) contracts are set to outsource the development and maintenance, to decrease internal costs. Typically, they do not provide an adequate documentation. Therefore, when the supplier is changed, system evolution becomes rather difficult. Lock-in situations are very common since *"suppliers want to defend their know-how to maintain their competitive advantages"*. Like factor 10, *"a good Service Level Agreement is key to overcome problems"*. However, what happens is that SLA are not respected properly, to get higher discounts on services.

Factor 13: International applications vs. Quality & Maintainability.

According to the panel, there is a difference between domestic and international software products, which is partially a concern. Apparently, international applications are more maintainable but have less functionalities. The reason seems to be that *"international applications are less flexible than domestic ones because they implement simpler functions and not because they are written or designed better"*. One panelist stated that for a well–known, specific software application *"a low level of customization usually means lower license and maintenance costs"*.

Factor 14: Domestic applications vs. Quality & Maintainability.

Regarding domestic applications, they appear to have more functionalities but incur in higher maintenance costs. For one panelist, the reason seems to be that *"Domestic applications are really rich of functionalities due regulatory requirements defined by the financial Authority"*. One expert expressed a specific concern, stating that *"software applications should comply with international standards. Before the acquisition each customer has to test this compliance. In reality, this never happens, and if it were the case most would fail. The only exception are applications delivered to NASA, US Air Force and so on, but at which costs?"*.

Factor 15: Measurement of software quality.

Losing control over the system quality is a concern. First of all, *"poor use of Software Engineering methodologies are the first killer of quality"*, according to one panelist. According to another expert *"even though methodologies are well–known and some tools are available, they are not implemented within the software development process"*. The discussion about tools was quite interesting. *"Tools do exist but are extremely expensive and not suited for small banks"* stated one panelist. For another informant *"a tool suited for us does not exist on the market place, so we built one ourselves"*. Finally, an expert very frankly explained that *"tools and methodologies are well–known, however neither customers nor suppliers use them since none is willing to pay for them"*.

Factor 16: Lower developers' expertise and professionalism.

This was one of the most debated factors. Experts have quite a different opinion on such a topic. The confrontation on this issue gave some extremely interesting insights. The problem is quite broad and complex. The general view is that the software development landscape changed dramatically along the years. So, developers could not be blamed for poor quality work. Again, *short time to market* expectations and *shrinking IT budget* is generally considered the main reason for low software quality. One panelist was able to give us a comprehensive view on the topic. According to this person, reasons should be found in:

- decreasing developers' daily rates and an extensive use of junior professionals;
- low quality and superficial functional analysis given to developers;
- compression of the development time following business needs (e.g., regulatory constraints, C-level decision, strategy);
- weak methodology and programming skills.

Our experts complained about low IT budget and use of junior professionals for highly complex applications. *"Software quality decreases because developers' daily rates decreased in the last years"* stated one informant. This opinion is largely shared among our panel. In fact, many others informants shared similar statements, like *"the IT supply ecosystem changed dramatically in the last years. The pressure on unit prices led to the proliferation of a purely tactical development. So, long term-view maintenance & evolution (with a higher unit price) has been penalized, increasing future system's cost"*. Or *"shrinking time to market of software projects contribute to the low level of software quality. Moreover, the decrease of daily developer's rates forces vendors and consultants to hire people which accept very low rate. Typically, they are not the best developers on the market"*. *"In my opinion, the contraction of IT budget costs and lower salaries payed to developers is the main cause of the decline of software quality, driving to an overall increase in the total cost of ownership of the information system"*, stated another. Since *"shrinking IT budget leads to lower developers' daily*

rates, senior figures are less involved in crucial projects". Not surprisingly, one expert complained that *"software vendors and consultants use for our projects only junior professionals with no supervision, so, the result respects this aspects. We really have to control their work step by step"*. According to another informant, *"more than a result of a decline in individual professionalism, I would say that it is a drop in quality of delivery of increasingly smaller teams to cost cutting"*. Finally, one said that *"I did not notice a decrease of individual professionalism. Rather I observe poor deliveries by shrinking teams due to lower IT budget"*.

Moreover, our experts found that low software quality is due to poor (or missing) Software Engineering methodologies. *"In my experience, technical skills (i.e., knowledge of the programming language) is not lacking. Poor Software Engineering skills are the problem. Developers write directly the code, without planning and test it. So, testing is carried out by the customer after the release"*. Another informant said that *"I observe a shrinking quality in such environments where stringent time to market requirements are set without an organizing proper development process, including testing, according to state of the art best practice (e.g., DevOps)"*. Moreover, *"poor management of requirements in the demand phase leads to poor functional analysis and software quality"*, according to another panelist. Also, *"the fast deploy (with low testing) of new functionalities impacts negatively on the entire software life cycle"*. Probably, *"the complex information system architecture forces developers to sub-optimal solutions"*. Finally, *"more that lacking professionalism, what is missing are proper integration methodologies for system integration required by our complex information system"*.

However, some are also complaining for lacking of specific skills. *"On older architectures (i.e., mainframes) developers are not enough skilled"*. Or *"lacking legacy system management skills as also a poor vision of end-to-end multiplatform development are the cause of low quality"*.

Complexity, of both regulation and application's environment impacts negatively on software quality, according to other panelists. *"We have to consider that information system's complexity raised exponentially in the last years. Thus, the level of professionalism has to be considered also on this regard"*. Concerning regulation, *"the most relevant issue is the stringent financial compliance, which changes rapidly. So, the information system needs to be updated with a really tight time to market"*.

Also IT-business alignment is considered to be a reason of low software quality. *"In my opinion, it is a side effect of complaining with business goals, losing the architectural and strategic view on the information system, more than people individual skills"*.

Factor 17: Contracting & Skills.

Our informants had different opinions and held a variety of positions concerning contracting and assessment. For some, outsourcing is the main cause of poor software quality. *"According to my opinion this is the key problem. Outsourcing policies of the last years are the reason of poor software"*. Similarly, another person stated that *"the depletion of internal personnel skill, in favor of outsourcing will hurt long term sustainability. I always suggest to outsource only non-critical functionalities"*.

Other are quite in favor of outsourcing projects, since they are easily assessable. *"Contracts with good KPIs which ensure objectivity of delivery's evaluation is a viable and crucial solution. Such kind of artifacts are more manageable and of better quality than internally developed software"*. Or *"with a well defined sourcing model and metrics projects are fairly manageable"*. For another informant *"there are methodologies which enables a good control on suppliers"*. And *"to ensure proper control on information system management there is the need for a good IT governance; through*

which each process related to the evolution and maintenance of the software is managed and controlled" stated also on panelist. Generally speaking *"for Time & Material contracts, previous skills assessment is very important for project's success. Whereas, in traditional contracts, the focus is shifts on the quality of the deliverable and to its process"*.

Other experts generally supported outsourcing, with some criticality. *"A good initial setup is crucial for a good project management. However, I saw a more sloppy management in the last years, so that most project's aspects are disregarded"*. Or *"recurring to vendors and consultants leads to more complex project management but ensures the right skills and experience to get the job done. However, internal personnel is pivotal to control vendor's artifact quality"*.

Finally, some others expressed a more neutral opinion on the factor. *"There is no such right or wrong solution. Outsourcing development, evolution and maintenance needs to be carefully evaluated case by case"*.

Factor 18: Lacking tools & Methodologies.

We gathered different opinions for this factor. As usual, the broadness of the domain shows various positions. Some fully support the factor, stating that *"since we didn't find an adequate tool on the market, we developed it by ourselves. We are experiencing enormous benefits from it"*. Others, complain (again) with tight schedules *"I agree in those contexts with "forced" tight time to market deployments"*. And others already started the path of full methodology and tool implementation. *"Our bank is already implementing methodologies (i.e., DevOps) to improve software quality through a testing phase of new applications"*.

However, the relation between tools and Software Engineering methodology is quite broadly stressed. When the process method is poor also the best tool will fail. *"Now effective tools are available on the market. However, they require a strong methodological and organizational support. Software quality measurement by itself is useless if not supported by a Software Engineering culture and knowledge"*. Or, *"before lacking tools, I would argue that it's the poor Software Engineering methodology which "kills" software quality"*. Moreover, *"there are adequate tools and methodologies, the problem is that they are not applied within the software life cycle"* said another panelist. Training is also an issue. One informant said that his organization don't have skilled employees to use highly complex tools. So, this is a key reason why they are not implemented. *"There may be adequate tools but using them is quite a complex task. Yo need to be very skilled to use them"*.

Another interesting aspect is the trade off between software quality assessment and costs. *"There are methodologies but fully implementing them is quite expensive"*. Another stated that *"I think there are enough tools and methodologies, the problem is that software quality assessment is rather expensive, so none is willing to pay for it. Thus, they are simply not used"*. Especially small banks complained about licensing costs. *"For our small bank, existing tools are too expensive"*. And *"there are enough tools but they are very expensive"*.

Finally, vendors complain about the market maturity and ability to value both tools and methodologies as viable solution for software quality. *"There are tools and methodologies but it's the market which lacks to recognize them"*.

Factor 19: Establishment of internal and external development processes.

The establishment of internal and external development processes appears not to be an issue *per se*. *"It's possible to codify them contractually"* affirmed one panelist. Another affirmed that *"I think that Software Engineering methodologies and tools are mature enough to guarantee a good software quality for both internal and external*

developments". Furthermore, one highlighted how the process definition needs some iteration to be efficient. *"Processes need to evolve according to needs and experience. However, you need proper custom tools to verify the compliance. In that way, the developers become accountable for their work"*. Rather the problem is on monitoring and assessment. *"You have just to impose a set of guidelines to outsources, but then you have to be monitor them by internal employees. I think here is the problem, since the IT does not have an internal audit structure"*. Or, *"I think that besides asking for process guidelines, you have also to assess them"*. Some experts noted that customers may not be enough skilled to negotiate standards with a big software house. *"I question myself whenever software quality guidelines need to be negotiated, especially with highly certified software houses"*.

According to some, on the other hand, it is very difficult to establish the process externally. *"For outsourced custom software, this is rather difficult to assess, since any software house uses its internal standards"*. At least, you have to differentiate. *"You have to differentiate internal and external development. In the first case you have to manage it properly accordingly to business goals and costs. In the second case you can not impose process decision but you may pretend a certain quality standard"*.

Internal skills are quite important for this task. *"The most relevant aspect is that internal employees do not properly review outsourced developed software due to lack of time and skill"*. Organizations which puts value on such skills reported quite positive experiences, *"it depends on the business culture. If the organization retained internally IT functionalities, it is easily assessable, with very good outputs for system's quality; otherwise it is not"*. Also the use of state of the art tools is suggested by one expert in this regard. *"Innovative Software Engineering tools should be more exploited within organizations. There are already automatic testing tools which are very helpful"*.

Typically, IT cost-benefit trade off is a constant issue. In fact, *"it depends on how much you want to invest for good software quality (trade off between costs and benefits)"*, said one informant. Schedule is also here an issue, *"often the time to market is detrimental for continuous testing of IT projects"*.

Finally, one informant shared his experience with suppliers. *"The point is not the process definition, which is quite consolidated in best practices, but the assessment and accountability of suppliers. According to my experience there are three patterns:*

- *suppliers who do not follow the agreed methodology;*
- *suppliers who already use an adequate methodology/best practice, negotiate it with the customer according to his needs;*
- *suppliers who already use an adequate methodology/best practice, but do not find an appropriate agreement with the customer.*

The best solution is always an agreed process and its monitoring with adequate tools".

Factor 20: Developer's professionalism vs. Skills.

The aim of this factor is to understand the relationship between decreasing developer's professionalism with the use of Software Engineering methodologies and best practice due to time and budget constraints. Some experts argued that *"decreasing professionalism does not depend on budget, rather on education"*. Other saw the problem as domain specific, *"this may be true in other domains, not in digital–web ones"*.

Other experts related this factor to process issues. For example, one informant said that *"most of the time is devoted to the requirement understanding, thus time dedicated to the development shrinks"*. Another expert stated that *"since software is*

intangible, it is very hard to manage properly its purchasing from vendors. On the other side, vendors still struggle to find a good trade off between quality (intended as use of standards and best practices) and cost".

However, education is considered a central issue by most experts. *"Junior developers are not experienced with concurrency and integration since they are trained with stand-alone platforms".* Similar statements emerged quite frequently. *"I would also add that there is a larger use of low skilled developers".* Or, *"I strongly believe that this is a central issue. However, I also noticed the use of developers with no technical background".*

Probably, again, we may explain this phenomena with shrinking budgets. *"Sourcing policies had a great impact on that".* And *"often, due to strict time-to-market schedule, quality is sacrificed".* Since there are not adequate resources to invest in education and to train developers with no formal education in IT, just because they are willing to accept lower wages is a sound interpretation of this factor. *"I agree that time and budget are key issues to understand this tendency"*, affirmed another panelist.

Factor 21: Developer's professionalism vs. Rates.

The aim of this factor is to understand the relationship between decreasing developer's professionalism with the decrease of professional rates. Apparently, experts expressed quite a variegated picture on the topic. *"Professionalism in general is not decreasing. However, choosing cheaper suppliers leads to some compromises".* On the contrary, another stated that *"I agree in general but I disagree in particular, since we have very skillful experts in our organization".* Another, shared that *"today customers are willing to pay more for more quality".*

With regard to system integrators, *"as long as I can remember, system integrators learn the job by doing it, while we never argued about senior's professional rates".* Training appears to be a qualified issue also for others, *"My company invests a lot in developer's training, regardless of their daily rate. We believe that this is the reason because we sell a lot of software products and system integration services".* The issue appears to be more complex than the problem statement. In fact, an expert said that *"you can not relate professional rates to training. Internal policies are too complex to understand the phenomenon. In a market economy, where a company wants to increase its customer, value creation through its employee is key to its survival. Otherwise, it would be substituted by other companies".*

Finally, an informant suggested that it is not about individual professionalism, rather team composition. *"I do not think this is just a problem of training, rather of sizing and team management due to low budget. Project management, Business Analysis e Design (System and Module Design) are penalized in favor of purely development tasks".*

Factor 22: Web technologies vs. Methodologies.

With this factor, panelist expressed their opinions concerning the use of web technologies with development paradigms. There is the idea that the adoption of such technologies lead to less rigorous approach to Software Engineering. In this regard, Agile is considered a scapegoat of such sloppiness. *"Agile software development is largely misunderstood. Documentation is automatically extracted by metadata but the design is completely lacking".* Or, *"behind the Agile buzzword people hide methodological shortcuts".* Similarly, *"this problem rises when methodologies are not understood or applied, like Agile ones".* Others made more punctual observation, stating that *"the use of Agile works very good for prototyping. However, it is a time bomb for the design of patterns or architectures".* There is apparently a scarce culture about Agile. *"In my experience, I only saw Waterfall methodologies. This because the customer rarely is*

able to express clearly requirements", said another expert. This could be a reason why is not seen as a methodology, rather a shortcut to Waterfall. Culture is apparently an important (and lacking) issue.

Others noticed that *"I do not think it depends on technologies, rather on low attention about quality"*. And, *"Web technologies lead to high subjectivity in the development. Developers tend to use less Software Engineering methodologies within the development phase"*. Moreover, *"some new technologies may change something (like Big Data) but not the way in general to develop software"*. With respect to the use of tools one said that *"this aspect relates also to wizard development tools, where personalizations are managed without a methodology"*. And to development environments *"I think that the development on open environment leads to more flexibility with respect to host ones. However, this leads to less rigor, so companies have to manage it properly'*.

Finally, a clear recommendation by one expert. *"The customer has to choose the methodology. Otherwise, the supplier will provide software developed with different standards and methodologies"*.

Factor 23: Quality vs. Requirements.

This factor analyzes the relationship between low software quality and clarity of user requirements as provided by the customer. The idea is that poorly defined requirements lead to misunderstandings with software developers, which will deliver lower quality software. The functional quality will be low if the requirements' clarity is low.

"I would say that it is quite obvious". Or, *"It is very hard to have high software quality if requirements are not clear"*. For some informant, it is a key issue of professionalism. *"It is correct, but IT professionalism means to me to transform business objectives in viable requirements"*.

Other panelists expressed different viewpoints. *"To me it seems much more like an alibi"*. Or shift the responsibility. *"The IT should be accountable to understand the right requirements"*.

With regard to the user requirements elicitation from our experts emerged interesting viewpoints. *"I do not think that user requirements are generally not clear enough, rather it is the process to capture and formalize them which has been reduced to a simple wish list"*. And, with reference to specific professional key figures one informant said that *"this is particularly true in the financial sector, where key figures like Demand e Service Manager, which are able to translate business language into technical requirements, are lacking. Usually, managers talk directly to developers with frequent incomprehensions"*. Also the quality of interaction matters. *"It depends on the interaction between customer and developer. A good developer (with more experience) will interpret better businesses' needs, however, he won't be cheap"*. As also, *"developers should be more proactive and ask if they do not correctly understand one requirement"*. More generally, *"this is a very important cause, since the clarity of the demand cycle is very important for a robust development phase"*.

Finally, an important element which emerged is the difference between functional and non-functional dimensions of software quality. *"It is partially true. If the requirement is badly or partially defined, it does not mean that software is* per se *of low quality. It will just work as it was (partially) defined"*. Moreover, *"I partially agree. Even though functional requirements are of low quality (due to misunderstanding), it does not mean that also non-functional ones are also of low quality"*. So, if generally experts affirmed that there is relationship between requirements clarity and functional requirements quality, this may not influence the non-functional dimension.

Factor 24: Requirements vs. Methodologies.

This factor analyzes the relationship between badly defined user requirements provided by the customer and the available Software Engineering methodologies to elicit them. The idea is to see if poor methodologies to elicit business goals and needs lead to the definition of unclear software requirements. One panelist expressed this concern. *"Requirements are usually elicited orally. Organizations are rarely structured to develop properly business needs into software requirements according to Software Engineering methodologies and best practices".*

The relation between the business and the IT appears to be a critical issue to explain this point. *"Poor requirements depend on the fact that often the customer does not know what he wants!"* Thus, startups with less defined organizational routines [335] seems to be more flexible and have less problem on requirements elicitation. *"New organizations reduced the distance between business and IT, so, requirements are more easily elicited".*

Although elicitation methodologies are considered important, the collaboration with business is perceived are pivotal. *"A robust process of Demand Management is alone not the answer to bad requirements elicitation. Rather, the business has to express clearly its goals and expectations".* Moreover, *"processes and methodologies are important to manage the demand properly and to turn businessesF' needs into software requirements. However, these needs should be expressed with enough clarity".*

Customers skills are also considered crucial for high functional requirements quality. *"Poor customer culture and skills lead to badly defined user requirements. There is the need to educate and support the business to define them better".* And, *"according to my opinion the problem lies in a low professionalism at the business side, which ask for more specialized consultancy".* The cause may also be poor prototyping. *"I agree. I think it depends also on the difficulty to provide a viable prototype to show how the future application will work".*

Interestingly, one informant expressed his concerns to consider the topic as just procedural. *"In my experience bad requirements are due to several factors related to doing teamwork. It is much better to focus on collaboration and work through goal oriented target rather than use highly structured protocols and methodologies which enhances barriers among people".* By the way, contemporary paradigms, especially Agile and DevOps ones are build according to this believe.

Generally speaking we noticed how our informants let emerged Humphrey's law, according to which requirements are not known by the customer until he will use them.

Factor 25: Requirements vs. Technical jargon.

Communication among different professional figures is usually not trivial. This factor explores the impact of the (mis-)use of technical jargon of different departments for requirement elicitation.

Although this may be true for many, since *"IT experts do not know business processes, they know about technology"* and *I think that there is the need of a "translation" phase"*, it appears to be very much related to the single business culture. *"The use of jargon is related to the business culture".* So, organizations which are aware of it tend to solve the problem upfront. *"When business and IT teams are co-located requirements are elicited effectively. Traditional silos approaches do not work".* Or, *"it is an old issue, nowadays business and IT speak the "same language", especially in our organization".* Moreover, *"I would agree if collaboration among IT and business is poor. With regard to my personal experience interaction and collaboration works very well, so we developed a common "dialect" within the company".*

One panelist suggested that excessive specialization may also be an important issue. *"I think that especially in the financial industry, people with a cross-cutting*

view on both business and IT processes are lacking. This may depend on excessive vertical specialization, so that there are no people around with the broad picture. To have such picture you have to put around a table many different figures to define the requirements, which is quite difficult. This is the reason why it is so difficult to elicit requirements".

Other explained that prototyping and preliminary analysis is of crucial importance for good functional requirement definition. *"I think that poor prototyping is a major issue of this topic. They enhances incredibly requirements specifications".* And *"if there is a good common preliminary analysis, elicitation is not a big issue".*

Finally, new professional figures are merging. *"It is true, for this reason new figures like the CDO (Chief Digital Officer) are emerging".* So, linking functions are quite important. *"A good demand manager is the "trait d'union" between both departments. This figure has to be strengthen using best practice tool, like business glossaries, metadata and impact analysis".*

Factor 26: Data analysis vs. Functional analysis.

This factor explores whenever poor data analysis influences functional analysis and consequently the system integrity. *"The "functional centric" view is quite misleading, since it relegates the importance of data. Data give the "static view" of existing functionalities, which is very important for the functional analysis. One software product may have all possible functionalities required by the user but lacking of fundamental data its deployment and system integration becomes impossible".* Moreover, *"data analysis skills are generally lacking and poorly used in functional analysis".* Apparently, this issue is present market wide. *"In my opinion, in the market there is a lacking perception about the importance of data analysis as preliminary phase of functional analysis".*

However, other experts disagree. *"Saying that poor analysis are due to poor data understanding is a quite generic (it is obvious that data processing is the main IT goal) and old issue".* Other issues are also relevant to understand the factor. *"Also knowing what different data means is important".* And *"it is not only an issue of poor technical skills".* Furthermore, *"personally, I saw poorer knowledge of banks' operation processes".*

Apparently, there was a shift after 2010 which give interesting insights. *"It is true for applications developed before 2010. Data governance is now more relevant and data analysis is performed before the functional one. So I see a clear discontinuity with the past".*

Factor 27: Functional analysis vs. Data modeling.

Our panelists generally stated that difficulties in functional analysis lies in bad data modeling and identification of data sources. *"In our bank, we need a data mapping, otherwise any further analysis would be useless".* The reason may be that for many *"the knowledge of data is the real starting point".*

The deep knowledge of data sources is considered a main driver for software quality, since it enhances functional requirements.

Factor 28: Documentation standards and tools.

In this last factor, the issue of lacking software documentation standards and tools emerged. Experts expressed that poor documentation hinders software maintainability and increases evolution costs. *"I think that the real problem is the time/effort trade off. If developers are pushed just to write some code, without documenting it properly, it becomes a problem in the medium run. Maintenance and evolution are critical since, due to developers' turnover, it is hard to have control over the software".* And, *"following development and architectural standards, enables those who did not write the software to manage it properly afterwards".*

However, other panelists affirmed that both standards and tools already exist, they just need to be implemented. *"There are already dedicated documentation tools, also open source ones, like "Alfresco", you just have to use them"*. And, *"there are tools but developers have to use them. From alone they do not work"*. Furthermore, *"there are very effective standards, but often they are not used"*. Or, *"there are valuable tools and the market need to recognize them. Software development is very similar to an assembly line of mechanical products. Tools and methodologies increase quality and reduce cost"*. One company also developed for its own purposes a documentation tool. *"Recently, we developed an "ad hoc" documentation tool which is very effective"*. However, an expert disagreed on this point. *"In my company, we already defined documentation standards. However, we still did not find an adequate tool to increase documentation efficiency"*.

Also here training appears to be a crucial point. *"Often standards are defined but no training is planned for developers"*. So also, *"the real problem is that minimal development standards are not followed"*.

Finally, the role of the IT department is stressed by one informant. He used the word "dignity" in the sense that such technical decision may not lie in another departments hand or even be the results of top-down decisions. *"I would argue that it is the IT department which should have the "dignity" to impose standards, not waiting decisions from the business side"*.

3.5 Discussion

Our research questions aimed to identify (RQ_1) and validate (RQ_2) the main quality related concerns in the IT financial sector. What emerged is a strong relationship among software quality, software architecture, and software process. Cost cutting had a direct impact on architectural aspects (both on *ex-ante* planning and *ex-post* analysis/reverse engineering), documentation and data modeling. Traditional quality models appear to be insufficient to explain these outcomes. Moreover, none of our informants affirmed to use any comprehensive quality model for their systems, reporting to struggle with their IT infrastructure.

Indeed, legacy systems are the most important ones for the day-to-day business operations [251]. The old-fashioned COBOL language is predominant for such systems since it is very efficient to process millions of batch transactions per day, which is the core activity of any financial organization. Such systems are truly business critical. They are reliable and have been running for decades, have been well tested in time and run virtually with no errors.

Unfortunately, we report the tendency to stratify core systems, in order to implement rapidly new functionalities due to budget constraints. Architectural integrity has been challenged by two opposed drivers: new requirements and scarce resources. Testing and documentation are often skipped. Reverse engineering becomes a highly complex and expensive activity.

Furthermore, the coexistence of codes written in different times and different programming paradigms is perceived as a barrier to software evolution and maintenance. The use of languages like COBOL impacts negatively the code understandability, as also the possibility to be translated in a more modern language. Moreover, none of the organizations we interviewed admitted to have a complete overview and map of the source code running on their information systems. Although we have to say that a considerable portion of information systems of financial organizations are provided

by vendors with COTS products under licenses, code ownership is fragmented and not centralized.

This raises also another issue: the total cost of ownership [122]. The levels of application complexity, system stratification and fragmented ownership increase tremendously the total cost of ownership of information systems of financial organizations. CIOs do often prefer incremental working solutions, rather than a comprehensive approach to such issue. Informants suggested that this behavior is caused by a fast turnover of C-levels and a short-term view. They do not have real incentives to tackle this problem, rather they focus on short-term, quarterly goals. This pattern has also been observed in literature [157].

Generally speaking, information systems of financial institutions appear to be characterized by a highly complex and stratified architecture, with a sinking quality. Moreover, with respect to the past, experts' opinions let emerge gloomy scenarios, due to an increase of functionalities driven both by market and regulations requirements and inadequate budget provisions.

3.5.1 Theoretical Coding

Although we did not commit to Grounded Theory [174], we used theoretical coding to map the items to the relevant literature [441]. According to our *pragmatic* Mixed Methods approach, we were able to build a valuable model from empirical evidence. In Grounded Theory theoretical codes are formalized after open and axial coding, since they are the highest level constructs [173]. Indeed, theoretical codes are flexible and do not explain just one theoretical construct, rather they link several constructs. Glasser explained how "they are not mutually exclusive, they overlap considerably... [and] one family can spawn another" [174, p. 73]. The use of theoretical coding provides a sharp analytical edge in the results. It helps to provide clarity, coherence and theoretical relevance of data.

Since 28 items are not parsimonious for a model development, we clustered the factors into high-level categories, following [156]. So, we did not just provide the description of the elicited items, we mapped them into some relevant ISO standards. Using ISO standards instead of literature has several advantages in our case.

Information systems of financial organizations are supposed to comply with such standards. We dealt with concepts which were very well–known to our informants, enabling a homogeneous and coherent model. Another advantage of dealing with standards is that they are *de facto* second-order theories, built on grounded pre-existing ones and shared among scholar's and practitioner's communities. Three categories emerged from our study: software quality, software process, and software architecture. The 28 factors we identified were mapped in the corresponding standards. With regard to software quality, we mapped our factors within the Product Quality Model of the standard ISO/IEC 25010:2011, as shown in Table 3.5. To map the factors regarding software process the ISO/IEC 12207:2008 standard for the Software life cycle processes was used, shown in Table 3.6. Finally, to see the most relevant elements related to software architecture, we performed our mapping over the ISO/IEC/IEEE 42010:2011 standard in Table 3.7.

To map our factors to sub-characteristic we used a Delphi approach derived from software cost estimation [62]. We elaborated autonomously the categorization. After that phase, we met personally, and took for granted the unanimous cases. The other cases, where we were in disagreement, were discussed until consensus was reached. Since the factors which came out from the panel where rather cross-cutting along

different quality characteristics, we assigned some factors on more than one characteristic. We choose to be not restrictive in our categorization since the aim was to see if the standards are sufficiently comprehensive for the identified factors. The final factor mapping, as described in Tables 3.5, 3.6 and 3.7, is the unanimous outcome of the iterations.As a result, multidimensional and overlapping concepts provided by experts were referenced to relevant theory, developing, *de facto*, a model which is intended to give a comprehensive picture of ISQ by abstracting and connecting from more detailed individual ISO models contained within it (i.e., a meta-model [308]).

3.5.2 A Model for Information Systems Quality

A *meta dimension* of quality emerged from empirical evidence. We specifically propose the prefix *meta-*, intended to define another subject that analyzes the original one but at a more abstract, and higher level [1]. Indeed, factors may be explained through a cross-cutting analysis of the three standards, since they are not related just to one category (i.e., quality, architectural, or process). So, having a comprehensive idea about ISQ means to manage a meta–view on these three categories. Basically, each category influences the others and has an impact on them.

For instance, business goals have a direct impact on the information system. For example, IT cost cutting by a bank executive board to invest in other departments or to be more profitable the next quarter is a clear business goal. Short term-view decisions or even not-taken decisions are business goals which influence all three categories of quality, architecture, and process. As a result of our research, non-IT executives are willing to pursue short-term goals, according to our experts, increasing the total cost of ownership. We did not collect non-IT experts' opinions in our study, so we can not detail this phenomenon. However, literature tried to explain this common pattern [157]. Like the panelists remarked, this led to low maintainability & evolution capabilities as also to some architectural anti-patterns. Several quality aspects are influenced by such business goals, like quality, maintenance & evolution, processes, and architecture. In other words, the proposed ISQ model is able to provide a comprehensive view about the key quality characteristics of an information system. Furthermore, it helps to trace the impact of decisions on one category into the others.

Therefore, we do not induce (intended as the outcome of a qualitative research) another quality dimension of IT systems. We induce, instead, a new *meta model* for information systems quality composed by software quality, software architecture, and software process.

Our contribution is of qualitative nature: we inductively report emerging factors of IT quality into the three categories of the ISQ model which have not been observed before in a real-world context. The proposed SQuAP (Software Quality, Architecture and Process) meta-model is represented in Figure 3.2 and it links to the ISO/IEC 25010 standard regarding software quality, ISO/IEC 42010 about architecture description, and ISO/IEC 12207 regarding software life-cycle processes.

3.6 Theoretical and Practical Contribution

From a research perspective, this work makes a theoretical contribution by providing a new model of ISQ based on the three categories of software quality, software architecture, and software process. Also, the explanation of the links between these categories, into ISO standards' sub-categories and experts' IT quality factors has been provided in Tables 3.5, 3.6, and 3.7. Additionally, it relates an organization's behavior and its impact on ISQ, through the relationships among the categories.

Characteristics	Sub-characteristics	Factors
Functional suitability	Functional completeness	6, 7, 12, 23, 24
	Functional correctness	6, 7, 12, 15, 24, 25, 26
	Functional appropriateness	3, 6, 7, 27
Performance efficiency	Time behavior	5
	Resource utilization	5
	Capacity	
Compatibility	Co-existence	5
	Interoperability	5
Usability	Appropriateness recognizability	7
	Learnability	8
	Operability	3, 9
	User error protection	
	User interface aesthetics	
	Accessibility	
Reliability	Maturity	8, 15, 18, 19, 20, 21, 22
	Availability	
	Fault tolerance	7
	Recoverability	
Security	Confidentiality	
	Integrity	2, 7, 15
	Non-repudiation	15
	Accountability	2, 7, 15
	Authenticity	
Maintainability	Modularity	1, 2, 10, 18
	Reusability	4, 8, 9, 10, 13, 14, 18, 28
	Analysability	4, 5, 6, 8, 9, 10, 11, 12, 18
	Modifiability	2, 4, 5, 8, 10, 11, 12, 18
	Testability	4, 5, 8, 10, 18
Portability	Adaptability	4, 8, 9, 10, 13, 14, 28
	Installability	10, 13, 14
	Replaceability	4, 6, 8, 9, 10, 11, 12, 13, 14

Table 3.5: Factor mapping according to ISO/IEC 25010 – System and software quality models

Characteristics	Sub-characteristics	Factors
Primary	Maintenance	1, 24, 5, 9, 10, 12, 13, 14
	Operation	
	Development	2, 5, 18, 22, 23, 24, 25, 26, 27
	Supply	13, 14, 16 17, 19, 20, 21
	Acquisition	3, 13, 14, 17, 19
Supporting	Documentation	4, 7, 8, 10, 11, 12, 28
	Configuration Management	9
	Quality Assurance	1, 2, 3, 6, 12, 13, 14, 15, 18, 19
	Verification	3, 9, 15
	Validation	15
	Joint Review	
	Audit	15
	Problem Resolution	9
Organizational	Management	5, 8, 12, 20, 21, 25
	Infrastructure	17
	Improvement	12, 17
	Training	16, 17, 20

Table 3.6: Factor mapping according to ISO/IEC 12207 – Software Life-cycle Processes

Characteristics	Sub-characteristics	Factors
Objective	Model Kind	
	Architecture Model	1, 2, 18, 26
	Architecture View	1, 6, 15, 18, 23, 26, 27
	System-of-Interest	1, 11, 12
	Architecture Rational	1, 8
	Correspondence	4, 14, 22
	Correspondence Rules	4, 9, 13, 22, 24, 28
	Architecture Framework	9, 13, 26, 27
Subjective	Concern	3, 5, 12, 14, 15, 16, 17, 18, 19, 20, 21, 23, 28
	Stakeholder	1, 3, 6, 15, 12, 16, 24, 25
	Architecture Viewpoint	10, 23
	Environment	11, 21, 22
	Architecture Description	1, 7, 8, 10, 18, 28

Table 3.7: Factor mapping according to ISO/IEC 42010 – Architecture description

FIGURE 3.2: SQuAP (Software Quality-Architecture-Process) meta-model

Literature highlights that software quality aspects and architecture are tightly coupled [377]. Conscious or unconscious faulty decisions on an architectural level, lead to low system-wide quality attributes, in particular poor maintainability and evolution capabilities [287]. We were able to validate this literature insight also in our research. We observed a tendency to stratify applications, rather than to evolve the existing ones. This is performed for several reasons due to poor documentation, testing, and reverse engineering capabilities.

As for practice, our model provides a management tool to map business decisions into IT categories, to trace their impact on the system at large. SQuAP thus becomes a tool that can structure strategic decisions. This may lead to a general rethinking of IT-business alignment. Typically, the failure of IT to deliver value, poor understanding of IT by business executives, lack of a clear vision with respect to IT, different views of business executives and technology specialists, lack of a shared vision in relation to IT, poor technical skills of CIOs, and criticism of legacy IT are the reasons of such a structural misalignment [267]. Mutual understanding of CIOs with other C–levels is considered in literature as a main driver to IT-business alignment [48]. Not surprisingly the relationship between IT–business alignment and competitive advantages is a well–known topic in research, since it leverages business capabilities through automation and business intelligence [374].

SQuAP addresses how to deal both the managerial and the technical perspectives, and their IT-business alignment, and in particular the total cost of ownership of information systems of financial organizations. It is intended to bridge the understanding of IT to non–IT C–levels. Indeed, a wider use of standards (i.e., IEEE/ISO/IEC) is an accountable way to move in this direction. In this regard, a model like SQuAP, based on industrial standards, is useful to visualize and assess the relevant assets of information systems.

3.7 Conclusions and Limitations

This study used a qualitative Delphi-like methodology to identify IT quality concerns regarding information systems of financial institutions. The concerns were validated through a survey of a large target-panel composed of carefully selected domain experts. The picture which came out of our research is a consequence of the short-term vision of C-levels. This short-sightedness impacts the strategies of maintenance and evolution of information systems of financial organizations. All concerns made explicit or implicit reference to software quality, process, or architecture issues. Thus, we mapped them within the ISO 25010, 42010, and 12207 standards. A new *meta model* for ISQ emerged from our inductive research approach regarding the IT financial sector. Such model explains the tight relationships between software quality, process, and architecture. We have followed an inductive, theory-building approach. The focus is not on the single standard, rather on the interactions among them, proposing a cross-cutting view. This model provides valuable insights to map and assess ISQ.

Limitations were our concerns during this study, since we used a new inquiry methodology through Mixed Methods. We use both qualitative and quantitative validity paradigms. To analyze the qualitative dimension, we adopt: credibility, transferability, dependability, and confirmability, following [185]. For the quantitative dimension we use traditional statistical conclusion, internal, construct, and external validity, as suggested by [491]. Even though validity dimensions of qualitative and quantitative researches are different, we aggregated them but discussed them separately due

to epistemological reasons. The research was homogeneous and both qualitative and quantitative dimensions gained trustworthiness one from another.

Credibility & Internal. Factors identified are all credible. We identified them through a Delphi-like process which lasted about one year. Panelists were sector experts, with a daily exposure to the researched concerns. The whole panel had the chance to discuss, refine, and group the elicited items, through different phases. None of the experts argued that any of the concerns should be excluded. Extra-items, which did not reach full consensus among the Panel, were isolated and treated properly. Moreover, they are presented separately from the first 15 items. However, target-panelist were asked to validate all 28 items through survey randomized items. Random stratified assignment of the research subjects, were designed to maximize internal validity. Representativeness of the sample is high for two reasons. Experts were chosen among a highly qualified pool of an established IT consulting firm of the financial sector. Strata were first assumed by our experience and than integrated into the panel. Moreover, the guarantee of anonymity given to both panelists and target-panelists allowed a unbiased and frank discussion. This increased the credibility of the study, since experts were able to answer and express openly their knowledge.

Transferability & External. The Mixed Methods approach aims to explore and build new theory (induction) and also to validate it (deduction). The degree to which the results of qualitative research are transferable to other settings may be interesting. Admittedly, we focused on one sector, to enhance internal validity, however our findings are still fairly generalizable. The financial industry is one of the most standardized sectors worldwide due to market and regulations similarities. All those points were discussed in Subsection 3.2.6. Therefore, it is reasonable to believe that most concerns are largely shared among the entire industry, for the reasons before described. Furthermore, our findings may also be generalizable for other industries with similar characteristics. This study focuses on the most important factors which threaten the IT financial sector. Threats to external validity are not considered very harmful since we consider a standardized industry.

Dependability & Construct. Experts were carefully chosen through a stratified randomized sampling, using the highly refined pool of an established IT consulting firm, specialized in the financial sector. We described the three dimensions (experience, company, and role) and sub-dimensions, as also the stratification of the samples. At the end of the qualitative research, a target-panel of 124 experts carefully selected through stratified randomized sampling were asked to evaluate the panel's outcome. The outcome was a substantial consensus on all concerns with a degree of agreement above 70% for all first 15 concerns, with high average scores. However, even though the first 15 concerns were highly shared among target-panelists, this does not mean that these are the most relevant for them. Thus, following our research approach we added also 13 extra-items not fully shared by the panel. Interestingly, such extra-items have a lower agreement degree among the target-panel. However, this approach provided a full *pragmatic* picture of our research journey.

We remark that the concerns were elaborated by a high level panel of experts in a three-phase round. The aim of the quantitative part was to verify the construct. Hence, we fully comply with our Mixed Methods approach.

Confirmability & Statistical conclusion. All items were discussed within the first panel, through different phases, which led to a continuous check. After the whole qualitative research process, concerns were evaluated by a large target-panel composed of 124 experts. We computed our results with MS Excel using representative sample sizes to increase statistical power. Measures and treatment implementation are considered reliable.

It was concluded that the threats would not to be regarded as critical. As we know, there is a constant trade-off between internal and external validity [491], and our sampling strategy took this into account. With Mixed Methods, we aim to get useful insights for theory building (external validity and transferability), while we also try to validate it within the same study (internal validity and credibility).

Our research will continue in several directions. Mixed Methods research is needed to validate and extend this model to other industries. Several software quality metrics, as also managerial and organizational evaluation techniques can be exploited. We are exploring the support of semantic technologies, especially ontologies, to provide a formal knowledge representation layer on top of which different methods and technologies can be validated and developed. Finally, empirical validation through case study research may broaden the use of the model as a central reference for ISQ.

Chapter 4

The SQuAP Ontology

4.1 Introduction

Industrial standards are widely used in the Software Engineering practice: they are built on pre-existing literature and provide a common ground to scholars and practitioners to analyze, develop, and assess software systems. As far as software quality is concerned, the reference standard is the ISO/IEC 25010:2011 (ISO quality from now on), which defines the quality of software products and their usage (i.e., in-use quality). The ISO quality standard introduces eight *characteristics* that qualify a software product, and five characteristics that assess its quality in use. A characteristic is a parameter for measuring the quality of a software system-related aspect, e.g. reliability, usability, performance efficiency. The quantitative value associated with a characteristic is measured by means of metrics that are dependent on the context of a specific software project, and defined in established literature.

The ISO quality standard only focuses on the resulting software product without explicitly accounting for the *process* that was followed or the implemented *architecture*. However, there is wide agreement [377] about the importance of the impact of three combined dimensions: software quality, software development process, and software architecture, on the successful management and evolution of information systems. In this respect, the industrial standard ISO/IEC 12207:2008 defines a structure for the software process life cycle, and outlines the tasks required for developing and maintaining software [426]. Regardless of the chosen methodology (i.e., Agile or Waterfall ones [377]), this standard identifies the relevant concepts of the life cycle and provides a useful tool for software developers to assess if they have undertaken all recommended actions or not. Each lifecycle concept can be evaluated according to its maturity level by means of established metrics, e.g. the Capability Maturity Model Integration (CMMI) [454]. As for the architectural dimension, the ISO/IEC 42010:2011 standard provides a glossary for the relevant objects of a software architecture. With regard to software architecture evaluation, intended as a way to achieve quality attributes (i.e., maintainability and reliability in a system), some approaches have emerged, the most prominent being ATAM, proposed by the Software Engineering Institute [249, 105, 47, 44]. Typical research in this domain is about how architectural patterns and guidelines impact software components and configurations [167]. A survey study [134] analyzes architectural patterns in order to identify potential risks, and to verify the quality requirements that have been addressed in the architectural design of a system.

The mutual relations among the three dimensions and their impact on the quality of software systems has been poorly addressed in literature, but recent empirical studies [405] in the domain of Software Banking Systems pointed out the importance of those relations. The study involved 13 top managers of the IT Banking sector in a first phase, and 124 additional domain experts in a second validation phase. The result is a model named SQuAP that describes these relations in terms of *quality factors*.

According to [181] the information available to guide and support the management of software quality efforts is a critical success factor for domains such as IT. Considering the broad coverage of its empirical provenance, a formalisation of SQuAP may serve as a reference resource and practical tool for scholars and practitioners of Software Engineering, to assess the quality of a software system, to drive its development in order to meet a certain quality level, as well as to teach Software Engineering.

The SQuAP model builds on the concept of *quality factor*: a n-ary relation between software quality characteristics that cover the three dimensions of software product, process, and architecture, based on the three reference standards ISO/IEC 25010:2011, ISO/IEC 12207:2008, and ISO/IEC 42010:2011 respectively. A SQuAP quality factor can be described as a complex quality characteristic (or *parameter*) that provides a three-dimensional view for assessing software quality. The model identifies twenty-eight quality factors.

Our contribution consists in a resource named SQuAP-Ont, an ontology that formally represents the concept of quality factor by reusing existing ontology design patterns (e.g., Description and Situation [378, 163]), instantiates all factors identified so far, and axiomatises them in order to infer measurable factors based on the characteristics available at hand. In addition, the ontology has been annotated with OPLa[1] (Ontology Design Pattern representation language) to increase its reusability. SQuAP-Ont is publicly available online[2] with accompanying documentation that describes the factors, under a CC-BY-4.0 license.

In the rest of the chapter, after discussing relevant related work (Section 4.2), we provide additional details about the SQuAP model by presenting two sample factors in Section 4.3; we describe the SQuAP-Ont ontology: its main concepts and axioms, the adopted design methodology and the reused ontology design patterns (Section 4.4); we provide examples of how to use it in Section 4.5; and discuss the resource potential impact (Section 4.6) before concluding and identifying future developments, in Section 4.7.

4.2 Related Works

The use of ontologies in the Software Engineering domain is very common [81, 499, 240]. The ISO standards referenced in Section 4.1 have been the subject of several ontological studies. For example, useful guidelines for their ontological representation are proposed by [199, 179].

An ontology-based approach to express software processes at the conceptual level was proposed in [288] and implemented in e.g. [431] for CMMI [95]. Software quality attributes have been modeled in [247], while an ontology for representing software evaluation is proposed in [85]. A formalisation of the ISO 42010, describing software architecture elements, is developed in [150] and in [270]. While [19] argues that different architecture domains can be integrated and analyzed through the use of ontologies.

Most [288, 431, 95, 150] of the aforementioned works focus on a strict representation of standards in terms of ontologies. Other scholars [247, 85] provide only preliminary ontological solutions for modelling quality characteristics or software evaluation and, to the best of our knowledge, they overlook the reuse of ontology design patterns. In contrast, our work focuses on the relation between the different ISO standards (system quality, software development process, and software architecture)

[1] http://ontologydesignpatterns.org/opla/
[2] https://w3id.org/squap/

for supporting the assessment of software system quality, with the added value of following a rigorous pattern-based ontology design approach.

Also at a higher level, an ontology to harmonize different hierarchical process levels through multiple models such as CMMI, ISO 90003, ITIL, SWEBOK, COBIT was presented in [352].

Ontologies referred to software quality focus primarily on quality attributes [247]. One quality evaluation, based on the ISO 25010 standard, is enhanced by taking into consideration several object-oriented metrics [323]. Similarly, [85] reuse current quality standards and models to present an ontology for representing software evaluations.

The ISO 42010 standard regarding software architecture already a formalization of architecture framework within the ontology of the standard [150]. An ontological representation of architectural elements has also been expanded by [270]. With particular reference to the architecture rationale, some visualization and comparison techniques with semantic web technologies have been proposed in literature [292]. Moreover, scholars showed that different architecture domains can be integrated and analyzed through the use of ontologies [19]

Finally, the Semantic Web community proposed also guidelines regarding the representation ISO standards of Software Engineering with ontologies [199, 179]. These two papers use a domain ontology, proposing the creation of a single underpinning abstract domain ontology, from existing ISO/IEC standards. According to the authors, an adoption of a single ontology will permit the re-engineering of existing International Standards as refinements from this domain ontology so that these variously focused standards may inter-operate.

4.3 Relational quality factors: the SQuAP Model

The motivation for developing SQuAP is based on the understanding, in both Software Engineering and information systems communities, that assessing software quality for contemporary information systems requires to take into consideration the relations between different dimensional perspectives, namely: software quality, process and architecture [377]. Accordingly, we conducted an empirical study in the banking sector [405]: In a first phase, based on the Delphi method [117], we involved 13 top managers of this sector to express their greatest software quality concerns. The result was a set of distinct 28 quality factors emerging from the elicited concerns, after a consensus–based negotiation phase (part of the Delphi method). In a second phase, we involved 124 domain experts that validated the 28 factors with a high level of agreement. Each factor has been then linked to a number of characteristics or elements defined in the three different standards i.e., ISO 25010, ISO 42010, and ISO 12207, for software quality, architecture, and process respectively. We followed the theoretical coding approach [441] to map the factors to the ISO standards [3].

The selection criteria and demographics of the experts involved in both phases, the inclusion and exclusion criteria, the agreement figures as well as the industry coverage are explained in Chapter 3. In the same chapter, we provide further details about the methodological approach and an exhaustive description of the 28 factors. A list of them is also published on the resource website[4] along with a short textual description, and by tables depicting their mappings to ISO standards.

[3] Standards are *de facto* second-order theories, built on grounded pre-existing ones and shared among scholar's and practitioner's communities.

[4] https://w3id.org/squap/documentation/factors.html

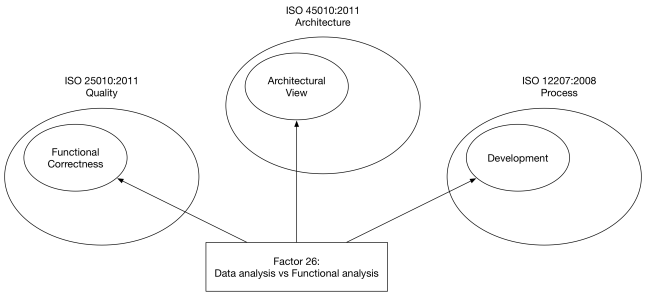


FIGURE 4.1: Factor 26: Data analysis vs. Functional analysis. This factor is defined as a relation between three quality characteristics of a software project: Functional Correctness (ISO 25010), Architectural View (ISO 45010), and Development (ISO 12207).

The 28 SQuAP factors have been rigorously although informally defined in our previous work [405]. These definitions are the main input for the development of SQuAP-Ont, hence it is relevant to report here at least one them. The definition of a factor consists of a set of quotes from the experts involved in the study followed by an analysis (based on theoretical coding) on what are the main characteristics and elements from the standards that emerged as components of the factor. We choose randomly one factor (26) to explain the underlying logic of the factor mapping.

Factor 26: Data analysis vs. Functional analysis. This factor explores whenever poor data analysis influences functional analysis and so, system integrity. *"The "functional centric" view is quite misleading, since it relegates the importance of data. Data give the "static view" of existing functionalities, which is very important for the functional analysis. One software product may have all possible functionalities required by the user but lacking of fundamental data its deployment and system integration becomes impossible"*, said one surveyed expert. Moreover, *"data analysis skills are generally lacking and poorly used in functional analysis"*, affirmed another one. Apparently, this issue is present market wide. *"In my opinion, in the market there is a lacking perception about the importance of data analysis as preliminary phase of functional analysis"*. However, other experts disagree. *"Saying that poor analysis are due to poor data understanding is a quite generic (it is obvious that data processing is the main IT goal) and old issue"*. Other issues are also relevant to understand the factor. *"Also knowing what different data means is important"*. And *"it is not only an issue of poor technical skills"*. Furthermore, *"personally, I saw poorer knowledge of bank's operation processes"*. Apparently, there was a shift after 2010 which give interesting insights. *"It is true for applications developed before 2010. Data governance is now more relevant and data analysis is performed before the functional one. So I see a clear discontinuity with the past"*.

Theoretical coding analysis: experts stressed the importance of data and functional analysis, impacting on the dimensions of **Functional Correctness** (software quality), **Development** (software process), and **Architecture View** (software architecture). In fact, they refer to the functional suitability of applications through correctness. This impacts the development process, which is supported by such analysis. The architecture view addresses the concern of a suitable system held by the system's stakeholders. Figure 4.1 shows a graphical representation of Factor 26.

4.4 SQuAP-Ont: an OWL formalisation of SQuAP

The SQuAP Ontology (SQuAP-Ont) is designed by reusing ontology design patterns (ODPs) [165] according to an extension of the eXtreme Design methodology [379, 58]. This extension mainly focuses on providing ontology engineers with clear strategies for ontology re-use. According to the guidelines provided by [379], we adopt the *indirect re-use*: ODPs are re-used as templates. At the same time, the ontology guarantees interoperability by keeping the appropriate alignments with the external ODPs, and provides extensions that satisfy more specific requirements. The ontology addresses a set of *competency questions* [184], listed in Table 4.1, identified by analysing the SQuAP model (cf. Section 4.3) and by discussing with domain experts.

TABLE 4.1: Competency questions used for modelling SQuAP-Ont.

ID	Competency question
CQ1	What are the quality characteristics of a software system at software, process, and architectural level?
CQ2	What are the factors, the assessment of which, is affected by a certain quality characteristic?
CQ3	What are the quality characteristics that affect the assessment of a certain factor?
CQ4	What is the unit of measure (i.e. metric) associated with a certain quality characteristic?
CQ5	What is the value computed for assessing a certain quality characteristic?

In addition, it has been noted that in order to assess a factor at least one of its affecting quality characteristics must have been measured.

4.4.1 Ontology description

Figure 5.6 shows a diagram of SQuAP-Ont. We use the namespace `https://w3id.org/squap/`. SQuAP-Ont re-uses as templates the following ontology design patterns [378]: Description and Situation (D&S)[5] [164], and Parameter Region[6].

The D&S pattern allows to represent the conceptualisation of a n-ary relation (i.e. description) and its occurrences (i.e. situations) in the same domain of discourse. For example, it is used for representing the description of a plan (D) and its actual executions (S), the model of a disease (D, e.g. its symptoms) and the actual occurrence of it in a patient (S), etc. SQuAP-Ont reuses this pattern for modeling quality factors with the class `:SoftwareQualityFactor`, a subclass of `:Description`. The actual occurrences of quality factors assessed in a specific software project are modelled with the class `:FactorOccurrence`, a subclass of `:Situation`. Both `:Description` and `:Situation` are core elements of the D&S pattern. According to the D&S pattern a `:Description` defines a set of `:Concept`s. In the context of SQuAP-Ont we say that a `:SoftwareQualityFactor` uses a set of `:SoftwareQualityCharacteristic`. This relation is modelled by the property `:usesQualityCharacteristic`. We model three types of `:SoftwareQualityCharacteristic`: `:SoftwareQuality`, the class `:ArchitecturalAlignment`), and the class `:ProcessMaturity`. They classify the characteristics associated with the three different ISO standards and their respective perspectives, i.e. software quality, architecture, and process. In a similar way, a set of entities are in

[5] http://ontologydesignpatterns.org/cp/owl/descriptionandsituation.owl
[6] http://ontologydesignpatterns.org/cp/owl/parameterregion.owl

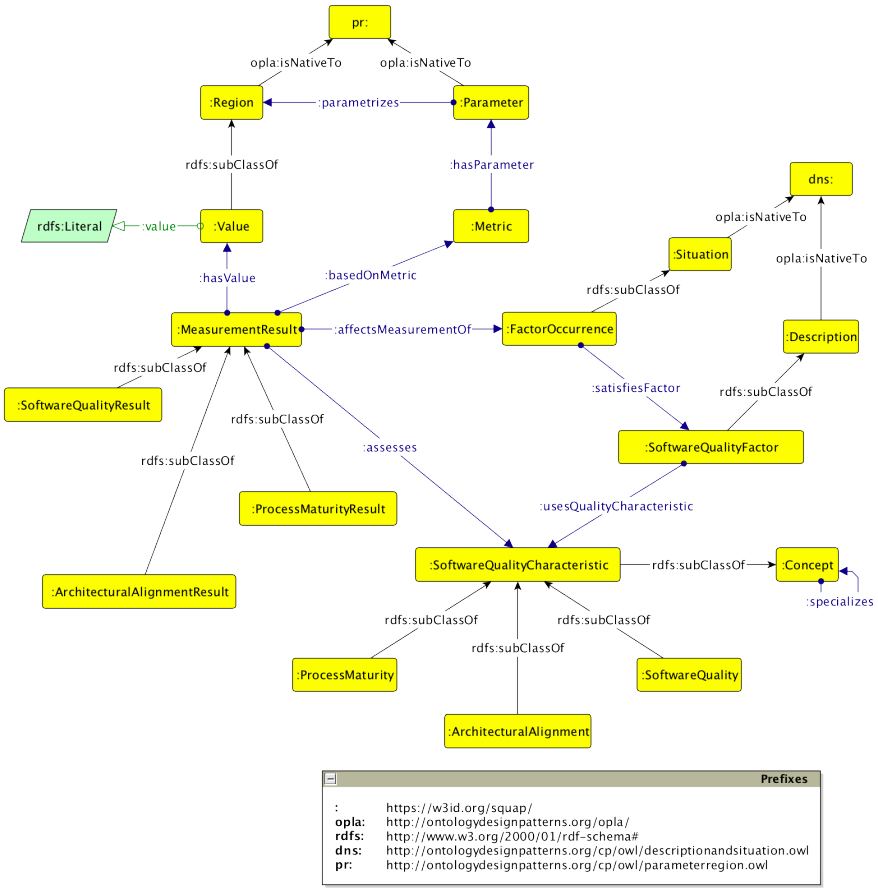


FIGURE 4.2: Core classes of SQuAP-Ont.

the setting provided by a `:Situation`. In the context of SQuAP-Ont we say that a set of `:MeasurementResult` affects the assessment of a `:FactorOccurrence`. We model three types of `:MeasurementResult` with the classes `:SoftwareQualityMeasurement-Result`, `:ArchitecturalAlignmentResult`, and `:ProcessMaturityResult` which are instantiated with result measurements computed for assessing the quality characteristics of a specific software system. A `:MeasurementResult` has a `:Value` and a reference `:Metric`. For example, we may want to represent that the *Reliability* of a software system is associated with a specific degree value according to a certain metric. This part of the model reuses the Parameter Region ontology design pattern as template.

In D&S each entity that is in the setting of a `:Situation` is classified by a `:Concept`. In the context of SQuAP-Ont we specialised this relation by saying that a `:MeasurementResult` assesses a `:SoftwareQualityCharacteristic`. Based on the `:MeasurementResult`s that compose a `:FactorOccurrence` it may satisfy one or more `:SoftwareQualityFactor`s (cf. modelled by the property `:satisfiesFactor`.

`:SoftwareQualityFactor`s are represented in SQuAP both as individuals and classes, by exploiting OWL punning. In the first case, the factors are represented as instances of the class `:SoftwareQualityFactor`. All factors identified by the SQuAP model are instantiated in the ontology. In the second case, the factors are represented as subclasses of `:SoftwareQuality-Factor`. The benefit from modeling factors with punning is the possibility to use both DL axioms or rules (e.g., SPARQL CONSTRUCT) to infer new knowledge. SQuAP-Ont models the three types of `:Software-QualityCharacteristic`s in a similar way: a set of individuals extracted from the SQuAP model, according to the three reference ISO standards (cf. Section 4.3), are

included in the ontology. They are also modelled as classes so as to make the ontology extensible with possible specific axioms). Furthermore, `:SoftwareQualityCharacteristic`s are organised hierarchically by means of the object property `:specializes`.

SQuAP-Ont is annotated with the OPLa ontology [211] for explicitly indicating the reused patterns. We use the property `opla:reusesPatternAsTemplate` to link SQuAP-Ont to the two patterns we adopted as template i.e. D&S and Parameter Region. Similarly, we use the property `opla:isNativeTo` to indicate that certain classes and properties of SQuAP-Ont are core elements of specific ontology patterns. This annotations enable the automatic identification of the patterns reused by SQuAP-Ont, e.g. with SPARQL queries, hence facilitating the correct reuse of the ontology.

Finally, SQuAP is aligned, by means of an external file, to DOLCE+DnS UltraLight [7]. Tables 4.2 and 4.3 report the alignments axioms between the classes and the properties of the two ontologies, respectively.

TABLE 4.2: Alignments between the classes of SQuAP-Ont and DOLCE UltraLight.

SQuAP class	Align. axiom	DOLCE class
:Region	owl:equivalentClass	dul:Region
:Value	owl:subClassOf	dul:Amount
:Parameter	owl:equivalentClass	dul:Parameter
:Concept	owl:equivalentClass	dul:Concept
:Situation	owl:equivalentClass	dul:Situation

TABLE 4.3: Alignments between the properties of SQuAP-Ont and DOLCE UltraLight.

SQuAP prop.	Align. axiom	DOLCE prop.
:classifies	owl:equivalentProperty	dul:classifies
:isClassifiedBy	owl:equivalentProperty	dul:isClassifiedBy
:usesConcept	owl:equivalentProperty	dul:usesConcept
:isConceptUsedIn	owl:equivalentProperty	dul:isConceptUsedIn
:satisfies	owl:equivalentProperty	dul:satisfies
:isSatisfied	owl:equivalentProperty	dul:isSatisfied
:specializes	owl:equivalentProperty	dul:specializes
:isSpecializedBy	owl:equivalentProperty	dul:isSpecializedBy
:isSettingFor	owl:equivalentProperty	dul:isSettingFor
:value	owl:subPropertyOf	dul:hasRegionDataValue

4.4.2 Formalisation

The following is the formalisation of SQuAP-Ont described in Section 4.4.1. The formalisation is expressed in Description Logics. For brevity, we use the terms `SwQualityChar` for `SowftwareQualityCharacteristic`, `ArchAlign` for `Architectural-Alignment`, `ProcMat` for `ProcessMaturity`, `SwQuality` for `SoftwareQuality`, `SwQualityFactor` for `SoftwareQualityFactor`, `MeasureRes` for `MeasurementResult`, `ProcMatRes` for `ProcessMaturityResult`, and `MeasureQualityRes` for `Measurement-QualityResult`.

[7] http://www.ontologydesignpatterns.org/ont/dul/DUL.owl

$$
\begin{aligned}
Value &\sqsubseteq Region \sqcap = 1 value.Literal \\
Concept &\not\equiv Description \sqcup Situation \\
SwQualityChar &\sqsubseteq Concept \\
SwQualityChar &\equiv ArchAlign \\
&\sqcup\ ProcMat \\
&\sqcup\ SwQuality \\
ArchAlign &\sqsubseteq SwQualityChar \\
ArchAlign &\not\equiv ProcMat \sqcup SwQuality \\
ProcMat &\sqsubseteq SwQualityChar \\
ProcMat &\not\equiv ArchAlign \sqcup SwQuality \\
SwQuality &\sqsubseteq SwQualityChar \\
SwQuality &\not\equiv ArchAlign \sqcup ProcMat \\
Description &\not\equiv Concept \sqcup Situation \\
SwQualityFactor &\sqsubseteq Description \\
&\sqcap\ \forall uses.SwQualityChar \\
&\sqcap\ \exists uses.SwQualityChar \\
MeasureRes &\sqsubseteq \forall assess.SwQualityChar \\
&\sqcap\ = 1 hasValue.Value \\
&\sqcap\ = 1 hasMetric.Metric \\
ArcAlignmentRes &\sqsubseteq MeasureRes \\
ProcMatRes &\sqsubseteq MeasureRes \\
MeasureQualityRes &\sqsubseteq MeasureRes \\
FactorOccurrence &\sqsubseteq Situation \\
&\sqcap\ \exists isAffectedBy.MeasureRes \\
&\sqcap\ \exists satisfies.SwQualityFactor \\
uses \circ specializes &\sqsubseteq usesConcept
\end{aligned}
$$

4.4.3 Implementation details

The namespace https://w3id.org/squap/ identifies the ontology and enables permanent identifiers to be used for referring to concepts and properties of the ontology. Additionally, we setup a content negotiation mechanism that allows a client to request the ontology either (i) as HTML (e.g. when accessing the ontology via a browser) or (ii) as one of the possible serialisations allowed (i.e. RDF/XML, Turtle, N-triples). The alignments with DOLCE+DnS UltraLight (DUL) are published in a separate OWL file[8], which imports both SQuAP-Ont and DUL. This allows one to use either SQuAP-Ont alone or its version aligned with and dependent on DUL.

The resource, including the core ontology, the alignments, and the usage examples, is under version control on a GitHub repository[9].

SQuAP-Ont is published according to the Creative Commons Attribution 4.0 International (CC-BY-4.0) license[10]. The license information is included in the ontology by using the VOID vocabulary [11].

[8] https://w3id.org/squap/squap-dul.owl
[9] https://github.com/anuzzolese/squap
[10] https://creativecommons.org/licenses/by/4.0/
[11] http://vocab.deri.ie/void

4.5 How to use SQuAP-Ont

Flexibility is among the most relevant characteristic of this ontology. Although the higher levels of this ontology, regarding the standard and the factors mapping are fixed, its measurement model can be adapted to the most suited scenario. In particular, the proposed way to evaluate information systems characteristics is just an explanatory example, which can be adapted to for any kind of assessment purposes.

As a usage example of the SQuAP ontology, we show a real world example consisting in the evaluation of a banking application by means of the Goal-Question-Metric (GQM) approach [36]. GQM defines a measurement model on three levels i.e. Conceptual level (Goal), Operational level (Question), Quantitative level (Metric). This method offers a hierarchical assessment framework, where goals are typically defined and stable in time, and metrics may be adapted according to new measurement advances. So, we stress the fact that this chapter does not focus on the measurement model, rather on the knowledge representation of SQuAP for assessment and benchmarking purposes. So, we provide the following synthetic RDF data about the assessment. The data are expressed as RDF serialised in TURTLE[12].

```
@prefix : <https://w3id.org/squap/examples/gqm/> .
@prefix arc:
  <https://w3id.org/squap/ArchitecturalAlignment/> .
@prefix sw:
  <https://w3id.org/squap/SoftwareQuality/> .
@prefix prc:
  <https://w3id.org/squap/ProcessMaturity/> .
@prefix squap: <https://w3id.org/squap/> .

:compatibility-result a
  squap:SoftwareQualityMeasurementResult ;
  squap:assesses sw:Compatibility ;
  squap:basedOnMetric :sonarqube-sw-quality ;
  squap:hasValue :sonarqube-value-b .

:correspondenceresult
  a squap:ArchitecturalAlignmentMeasurementResult ;
  squap:assesses arc:Correspondence ;
  squap:basedOnMetric :likert-scale-1-7 ;
  squap:hasValue :likert-value-7 .

:documentation-result
  a squap:ProcessMaturityMeasurementResult ;
  squap:assesses prc:Documentation ;
  squap:basedOnMetric :likert-based-prc-maturity ;
  squap:hasValue :likert-value-6 .


:sonarqube-sw-quality a squap:Metric ;
  squap:hasParameter :sonarqube-params .

:sonarqube-params a squap:Parameter ;
  parametrizes :sonarqube-value-a ,
    :sonarqube-value-b ,
    :sonarqube-value-c .

likert-based-prc-maturity a squap:Metric ;
  squap:hasParameter :likert-scale-1-7 .

:likert-scale-1-7 a squap:Parameter ;
  parametrizes
    :likert-value-1 , :likert-value-2 ,
    :likert-value-3 , :likert-value-4 ,
    :likert-value-5 , :likert-value-6 ,
    :likert-value-7 .
```

[12]The RDF is available at https://w3id.org/squap/examples/gqm

```
:sonarqube-value-b a squap:Value ;
  squap:grade "B" .

:likert-value-7 a squap:Value ;
  squap:value 7 .

:likert-value-6 a squap:Value ;
  squap:value 6 .
```

The example describes a banking system associated with three assessments about the dimensions of software quality, architectural alignment, and process maturity. The specific measurement results are: `:compatibility-result`, which assesses the characteristic `sw-quality:Compatibility` (software quality), `:correspondenceresult`, which assesses the characteristic `arc-alignment:Correspondence` (architectural alignment), and `:documentation-result`, which assesses the characteristic `prc-maturity:Documentation` (process maturity). Those measurement results are associated with a value (e.g. `:likert-value-7`, which identifies the value 7 of a Likert scale) and a metric (e.g. `:likert-scale-1-7`, which identifies a Likert scale ranging from 1 to 7). Each value is reported with a literal representation and its associated with a metric. It is possible to use the axioms defined in SQuAP-Ont in order to gather all the factors that can be enabled by the available measured quality characteristics (e.g. `sw-quality:Compatibility`). This can be done, for example, by executing a Protégé DL query, the result of which is shown in Figure 4.3.

In this example, different standards' items represent the Goals, which are measured with one or several (also concurrent) software quality metrics. In order to do so, we followed literature recommendations [474, 83]. The result is the sum of different evaluators, which represent a measurement of the three standards.

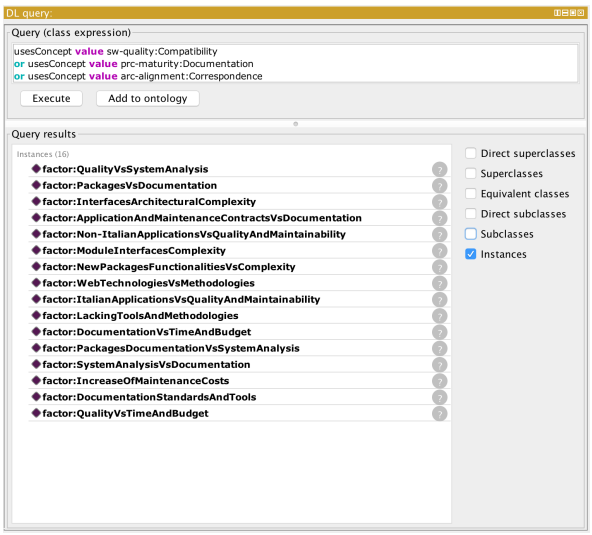


FIGURE 4.3: Execution of a DL query on the RDF sample.

Alternatively, it is possible to define productive rules in order to materialise the factors that are enabled by the available measured quality characteristics. The following SPARQL CONSTRUCT is a possible productive rule for our example.

```
PREFIX squap: <https://w3id.org/squap/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
CONSTRUCT {
  ?measurementResult
    :affectsMeasurementOf ?factorOccurrence .
  ?factorOccurrence
    a squap:FactorOccurrence;
```

```
    squap:satisfiesFactor ?factor
}
WHERE{
  ?factor
    squap::usesQualityCharacteristic ?char;
    rdfs:label ?factorLabel .
  ?measurementResult
    squap:assesses ?char
  BIND(IRI(
    CONCAT("https://w3id.org/squap/example/gqm/",
      ?factorLabel))
    AS ?factorOccurrence)
}
```

We remark that factors and quality characteristics are defined in SQuAP-Ont both as classes and individuals by means of OWL punning. Hence, one can decide to use DL reasoning or rules defined in any other formalism depending by the specific case, e.g. SPARQL CONSTRUCT.

4.6 Potential impact

In the last decade there has been a considerable effort, especially by the Management Information Systems research community, to study the phenomenon of the alignment of business and information systems [8]. What emerged is the importance of such alignment for both business' competitiveness and technical efficiency. In fact, when it comes to integrate new solutions, modules, or interfaces, such alignment is of key importance. Several other scholars found similar results, suggesting the importance of standard governance defining key architecture roles, involving key stakeholders through liaison roles and direct communication, institutionalizing monitoring processes and centralizing IT key decisions [65]. Especially in the financial sector, architectural governance is a key issue for IT efficiency and flexibility [416]. Generally speaking, this finding is also largely shared beyond the financial sector [274]. The need for people from different backgrounds (mainly business and technical ones) to align the organization is the greatest insight of this research stream.

To tackle the issue of information systems quality from an empirical perspective we started in 2014 to survey banking application maintenance group experts, Chief Executive Officers, Chief Information Officers, IT architects, technical sales accounts, Chief Data Officers, and maintenance managers [405]. This ongoing project is pursued with a leading consultancy firm, according to which we were able to cover with our representative sample the IT banking sector. Consequently, the need for a knowledge representation of different measurement models is perceived as contingent, and requested by the IT banking community in this broad project on information systems' quality. One crucial insight that emerged from the factors is the difficulty to assess their applications, also due to the diversity and complexity of measurement models. Indeed, the three standards measure three different dimensions. Quality measures the software as a product; Process as a process; and Architecture the alignment to a taxonomy. Accordingly, metrics and predictors reflect these differences. Therefore, the development of this ontology is a direct request from practitioners.

Since this research journey started from an industry's need, an ontology, intended as the knowledge representation of different measurement models is of pivotal importance, and a first tool to systematize the assessment of banking information systems' quality. Thus, this ontology will be used for consultancy purposes, to implement the SQuAP quality model. Moreover, it is also useful to trace changes of quality in time, and suggest specific improvements. So, this ontology is the knowledge layer over which

this quality model is built. Consultancy firms (like that ones we are working with) expressed their interest in a knowledge representation tool which can be displayed to customers in the assessment phase, to tailor their consultancy efforts. However, also bank's IT departments will use it for similar purposes. They can also tailor made and modify this ontology and the underlying metrics suggested by the literature, according to their specific needs.

For this reason we used a CC-BY-4.0 license, open to commercial use. Our industrial partners consider the use and reuse of this ontology as a great value for the practitioners' community.

4.7 Conclusion and future development

In this chapter we have described SQuAP-Ont, an ontology to assess information systems of the banking sector. SQuAP-Ont a) guides its users through the (ongoing) assessment phases suggested by Software Engineering literature; b) helps identifying critical quality flaws within applications; and c) extends and integrates existing work on software ISO ontology terms, diagram visualizations and ontology revisions. SQuAP-Ont has been developed for commercial use, within an industrial project on quality of banking information systems. Nevertheless, like all ontologies it is an evolving effort, and we are open to suggestions proposed by the broad researchers and practitioners community. In fact, we have already addressed several issues raised in previous studies, and according to industry's expectations.

Our future work aims to facilitate enrichment and refine the ontology continuously along with standards and literature recommendation changes. Another important aspect is to validate and monitor the application of SQuAP in domains and software projects different from the Banking System context. This may lead to the identification of additional factors as well as to the enrichment and refinement of the ones already identified.

Chapter 5

Knowledge Engineering for Socio–Technical Software Engineering

5.1 Introduction

The most critical phase in system design is the one related to the full analysis and understanding of "User Requirements". Difficulties arise especially where the ambiguity on the functions to implement is a continuous challenge. Due to the volatility of the user needs, changing of scenarios, and the intrinsic complexity of software products, requirement engineering benefits from an Agile approach in terms of (i) attainment with user's contingent needs, (ii) velocity, and (iii) cost reduction.

Formal methods for requirement engineering have primarily been conceived to drive efficiently the link between customers and developers [142]. They focus on reducing the management risk connected with the initial software production phase. The results achieved by these strategies are controversial and not always cost effective [294]. The diffusion of the use of Agile practices in the software production process is putting the human factor as the key asset to capture and understand the user needs [9].

We experienced an extensive use of methodologies to identify the "unexpressed dimension" of the user requirements and to surface the "implicit" knowledge of users within a real case study of an Italian governmental Agency.

The underlying principle is the methodological formalization of the non-linear human thinking into requirements in the form of agile "User Stories". Such an approach was successfully implemented within a mission critical organization to develop critical software applications. User stories are sentences written in natural language and have a very simple structure. The vocabulary used to write a user story depends on which user describes her need, thus is some sense it depends on the mental model the user has of her needs [456]. Capturing the essence of the users mental models [370] and overcoming the intrinsic ambiguity of the natural language are the two main goals of our study. Multiple dimensions to build a dynamic representation of requirements are the core innovative aspect of this work.

We give a problem definition of how to structure the description of user stories following Agile principles. The lessons we learned and some considerations about the importance of ontology based solutions for Knowledge Based Systems (KBS) in this context are discussed. The proposed approach is useful not only for requirement engineering but also to structure a highly interoperable knowledge representation architecture which enables a fast and flexible use in mission critical contexts.

This chapter is organized as follows. In Section 5.2 we review the most critical aspects of the use of KBS in mission critical systems; we also recall the basics of the *iAgile* process. Section 5.3 shows how we manage requirements using an ontology. The use of KBS technologies by the sponsoring organization is explained in Section 5.4. Finally, we draw our conclusions, summing up our findings in Section 5.5.

5.2 Complex software systems specification

In mission critical domains, the velocity of release delivery is often considered as one of the most valuable assets. A release will usually be a partial version of the final product, but the important issue is that it already works usefully for its users. An on-field command view of a military operation (i.e., user view of a Command & Control system) typically is: "*I want the right information at the right time, disseminated and displayed in the right way, so that Commanders can do the right things at the right time in the right way*" [41].

Important functionalities may be developed or refined in the first few sprints, due to the continuous interaction between users and developers. The primary objective of this constant dialogue within the development team is the rise of the implicit and unexpressed knowledge, which will be translated by developers into software artifacts.

One typical example in mission critical contexts is the "situational awareness". It may be described as: "*The processes that concern the knowledge and understanding of the environment that are critical to those who need to make decisions within the complex mission space*" [41].

Such a sentence contains a huge quantity of implicit knowledge. For example, the interpretation of "*those who need to make decisions*" has to be clarified. More generally, in a typical agile user story words like "situational awareness" would be written as "*as the one who needs to make decisions, I want to achieve the knowledge and understanding of the environment that are critical to accomplish my mission*". This statement is, of course, still overloaded with implicit knowledge.

In our case study, this issue was overcome through a careful composition of the team including domain experts. Continuos face to face and on-line interactions allowed to minimize information asymmetry [**Akerlof1995**] and align the different mental models [370]. The main shared target was to deliver effective software to end users in a fast way.

To understand better the main use cases, consider that a military C2 Information System (IS) for mission critical purposes is essentially built on the exercise of authority and direction by a properly designated commander over assigned forces in the accomplishment of the mission [446].

In order to deliver this capability several integrations have to be taken into account, i.e., hardware, software, personnel, facilities, and procedures/routines. Moreover, such a system is supposed to coordinate and implement processes, like information collection, personal and forces management, intelligence, logistics, communication, etc. These functions need to be displayed properly, in order to effectively support command and control actions [433].

The IS we are reporting on has been based upon the development of mission specific services, called Functional Area Services (FAS), which represent sequences of end-to-end activities and events, to execute System of Systems (SoS) capabilities. These mission oriented services are set up as a framework for developing users' needs for new systems. Furthermore, mission services are characterized by geographical or climate variables, as cultural, social and operative variables, which represent functional areas

or special organization issues. Mission services of the C2 system have been developed according to the NATO–ISAF CONOPS (Concept Of Operations), as required by management of the governmental agency we cooperate with:

- Battle Space Management (BSM)
- Joint Intelligence, Surveillance, Reconnaissance (JISR)
- Targeting Joint Fires (TJF)
- Military Engineering - Counter Improvised Explosive Devices (ME-CIED)
- Medical Evacuation (MEDEVAC)
- Freedom of Movement (FM)
- Force protection (FP)
- Service Management (SM)

Thus a C2 system is made of a set of functional areas which in turn respond to a number of user stories.

5.2.1 Evolution of a Mission Critical Information System through Agile

The mission critical information system we have studied is a Command & Control system which was capable to support on-field missions according to the NATO–ISAF's framework. The initial idea was to develop a Network Centric Warfare system (NCW) [15]. This system supports many of the operational functions defined in the contest of the NCW, according to the requirement documentation. The system has been employed in many exercises and operations and went through several tests. Today the system is serving mission critical purposes in NATO–ISAF operations e.g., the Afghanistan Mission Network.

However, several difficulties and limitations arose. The acquisitions were done according to Waterfall procedures, started in the early 2000s and went on until recently. The obsolescence of the components and related functionalities, along with the maintenance and follow-up costs connected to the Waterfall software life cycle are a big issue. Several problems are related to the impossibility to develop quickly new functionalities required by on-field personnel in a fast-changing mission critical scenario e.g., a modern asymmetric warfare. This led the use of agile software development paradigms which are supposed to overcome this crucial constraints.

Therefore, since 2014 a new "Main Command and Control System" (Main C2) to support the former system (Tactical Command and Control System or Tactical C2) has been developed. It was urgent to support the evolution of the Command and Control system, assuring a higher customer satisfaction in a volatile requirement situation. Moreover, due to budget cuts, the new system had to perform better with less resources. Costs related to both development and maintenance had to shrink rapidly.

Functional Area Services (FAS) are web-based services with a client–server architecture. Any FAS software component can be separately modified to respond to specific mission needs, as defined by users. The Main C2 has been validated in NATO exercise for the first time at CWIX 2015[1], with positive results. Core services are build

[1] www.act.nato.int/cwix

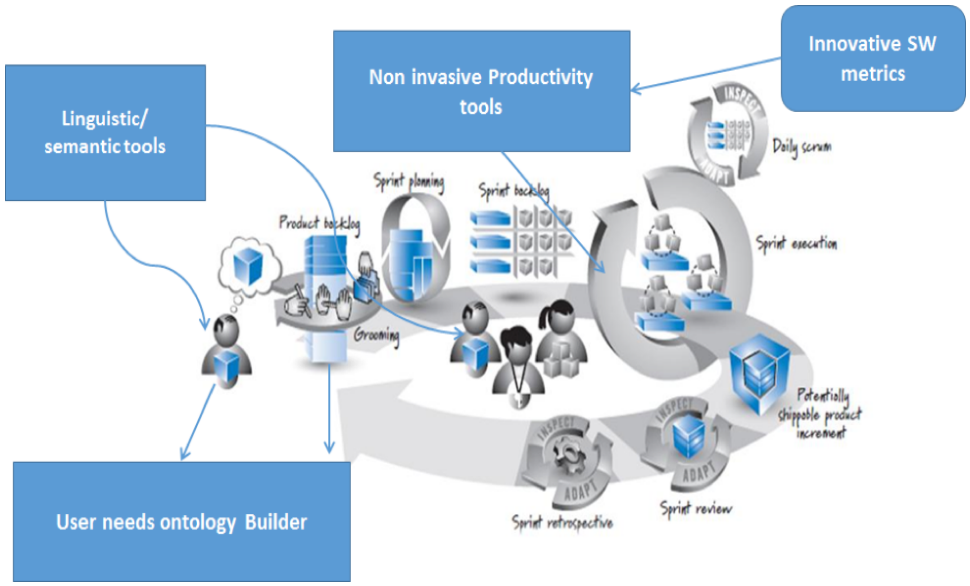


FIGURE 5.1: Sprint representation, inspired by [399]

to maximize interoperability with all the relevant NATO software packages available and COTS product. Therefore, Main C2 is both flexible to implement rapidly user needs, with high interoperability of already existing systems, like the Tactical C2.

To develop it, a new methodology was implemented, applying the principles of the "Agile Manifesto" [9] to both increase the customer satisfaction and reduce software cost. After the Agency's top management decided to go Agile, there was some discussion about the method to use. There was the need to exploit Agile's values and capability but within a mission-critical environment.

Scrum was found as the most suited, since it allows a high degree of accountability [421]. This methodology is very successful in commercial environments and the most widespread Agile methodology [465]. Moreover, it was the methodology which was the best known within the Agency. Therefore, other methodologies were not really taken into consideration, even though they might have given similar results.

The teams are mixed: they include developers from the defense industry and governmental officials, based at the Agency's Headquarter in Rome. The initial production phase was extremely successful and even the start up "Sprint" (production cycle of five weeks) was able to deliver valuable software [110].

What happened was that the expectation of the Agency's stakeholder grew rapidly. From 2014 to 2016, the methodology was refined, to respond to mission & security critical needs of the operations domain. Thus, an ad hoc Scrum-like methodology was developed with the name of *iAgile*, and tested for the development of the main C2 system [307].

This methodology, depicted in Fig. 5.1, has been developed for critical applications, where requirements change already during the first specification and after delivering the first release. The adaptation of Scrum for the special needs of C2 software systems has also been proposed in [194].

A well known approach to analyzing ephemeral requirements consists of formalizing and prototyping the requirement specification using a suitable language, like for instance Prolog [437]. The Humphrey's Requirements Uncertainty Principle remins us that, for a new software system, the requirement (the user story) will not be completely known until after the users have used it [446]. Thus, within iAgile, Ziv's Uncertainty

Principle in Software Engineering is applied, considering that uncertainty is inherent and inevitable in software development processes and product.

The incremental development approach enables easily any change of requirements even in the later development iterations. In our case study, due to the close interaction between the "requirement chain" i.e., from the customer to the development team, FAS were delivered with a high degree of the customer satisfaction.

The Scrum methodology developed within the Agency fully supports the change of requirements according to contingent mission needs. The traditional command chain was adapted to the development needs. Both structured and horizontal characteristics of Scrum are particular effective in a critical environment. These two characteristics are embedded in the model.

Mission critical organizations need to comply with a vertical organizational chain, to empower different stakeholders to their duties. In the field where we had this experience, a hierarchy enforces clear responsibility and accountability within the command chain. So, the customer becomes the accountable official for the mission needed requirement. However, to develop different mission critical requirements, it is crucial to have a straightforward and direct communication and collaboration with final users, according to the Agile Manifesto. Therefore, in the methodology, some user representative becomes part of the development team, allowing a better understanding of the needs and a faster development of the feature.

One of the key strengths of the methodology is its flexibility. The process is defined only at a high level, to be adapted in any theater of operation. It defines values, principles, and practices focused on close collaboration, knowledge sharing, fast feedback, and tasks automation.

The main stakehlder is the Product Owner (PO), who gives to the developing team the first input which is a product's vision. It is a high level definition to address the task that will be refined during the development cycle through the Backlog Grooming.

The Backlog Grooming is a key activity which lasts over the whole development process. It focuses the problem definition, refining redundant and obsolete requirements. Moreover, it prioritizes requirements according to contingent mission needs. The acceptance criteria and the scenario definition are set by the PO in the user stories.

The developing team used by the Agency is composed as follows (such team composition is an adaptation of standard Scrum roles within a mission critical context).

- The *Product Owner* is the governmental official in charge of a specific mission critical function which has to be developed. He provides the general vision of the final functionalities to the whole team i.e., what the system has to do. It may be that PO delegates its role to another official of his team. In this case, the PO becomes a team of people that has to decided about the systems functionalities and discuss them within the development team. Ideally, the PO team has to be representative of the final user, thus it should be made also of real users.

  This crucial role is pivotal for the positive outcome of the sprint. *De facto*, the shortening of the "requirement chain" through the involvement of end users and the constant feedbacks of the PO during the sprint is a key success factor.

  In our case study the stakeholders were initially barely aware about the development process. Due to a constant involvement within the iterations, the stakeholders became aware of the development methodology and aligned their expectations increasing their satisfaction. Through this involvement, there is an alignment of both interests and expectations that raises the quality of the

final artifact. So, the final product may not be fancy but down to earth with a high degree of immediate usability by a final user. Therefore, the degree of user involvement is of highest importance since it has a direct impact on the development itself and a ground for building a sense of ownership of the final product which is essential for the acceptation of the final product.

- The *Scrum Master* (SM) is a domain expert and is supposed to lead the development team and the Product Backlog management. The SM shapes the process according to mission's needs, leading continuous improvement like in any Agile team. He has to shield the development team from any external interferences as also to remove any hinder which may occur. What typically happens in mission critical organization is that information is shared only through very structured processes. So, there could be a loss of productivity, due the waste of time to obtain relevant information for the development process. The SM knows how to gain such information and is in charge of sharing it when needed, with no waste of time from the development side.

  According to the critical domain, he is accountable for the team's output. So, he is a facilitator but he takes the control of the team, considering also the different backgrounds of the members. Both PO and SM collaborate closely to refine requirements and get early feedbacks. Furthermore, his role is to build and sustain effective communications with customer's key staff involved in the development. Finally, he is in charge of the overall progress and take responsibility for the methodology used within the development cycles. So, he may do some corrections within the team to deliver the expected output.

- The *Development Team* composed by both military and civil contractors is in charge of the effective development. The team members are collectively responsible for the development of the whole product. Within the team there are no specialized figures (e.g., architects, developers, testers), and it is the team that organizes itself internally and takes responsibility over the entire product.

  The self organization empowers the team for the execution of the Sprint Backlog, i.e., the developed Product Backlog within the sprint, based on the prioritization by the PO. The team members are lead by the SM who is mainly a problem solver and interfaces with the organization which needed the mission critical product.

  The number of team members is between three and five highly skilled software developers. The absence of a specialization is due the fact that any member is supposed to have a good knowledge about the system developed with a clear vision on the final artifact. Finally, they are also involved in the testing phase, which is carried out by an independent audit commission.

- The *Coach* is an employee of the civilian main contractor and is in charge of the management of contractual issues. Since the typical contractual form for developing contractors is body rental, the Coach facilitates organizational issues which may occur during the development cycles. Her role is to smoothen problem which may rise, to get the team oriented to the development of the artifact.

After each sprint a deployable release of the system is delivered. In order to assure security standards of mission critical applications extra testing is pursued. This activity is carried out before the deployment within the mission critical network. So, before deployment three steps are carried out as follows:

1. The development team runs a first test in the development environment and then in a specific testing environment (stage), having the same characteristics of the deployment environment.

2. Afterwards, testing activities are performed by Agency's personnel involved in test bed based activities, in limited environments to validate the actual effectiveness of the developed systems in training and real operating scenarios (Integrated Test Bed Environment).

3. Finally, testing activities on the field performed to verify the compliance of the developed systems to the national and international standards and gather operational feedback to improve the system's performance and usability.

Only after the positive check of these three steps the functionality is deployed. At the end and beginning of a new Sprint, all interested stakeholders discuss about positive and negative aspects, to improve the next iteration. Therefore, it is an incremental process, which changes with the operational scenario. It is not a frozen methodology, but it evolves along with Agency's needs.

Finally, a quite important outcome of this approach is the cost reduction in all the system's lifecycle. A first assessment of the product cost per "line of code equivalent" with respect to other comparable internally-produced software showed a cost reduction by 50%. To consider those costs we computed a comparable software by dimension (LOC) and functional area (command and control). We considered all relative cost of personnel, documentation and maintenance costs and fix cost for office's utilities. The assessment after two years showed more significant cost reduction.

Generally speaking, we know from past experiences that, on average, cost per ELOC in similar C2 domains is about 145 dollars; with regard to ground operation the cost is about 90 dollars [383]. This study, in particular, was carried out for Waterfall in a procedural context. Based on Reifer's study, we carried out our evaluation regarding iAgile cost. It was quite surprising to realize that the software we measured had an average cost of 10 dollars per ELOC.

This was possible cutting maintenance and documentation costs, which represent the most relevant part of software development costs [377]. The cost reduction came mainly from the minor rework due to requirement misunderstanding (project risk reduction) connected to the short delivery cycle and to the integration of subject matter experts into the agile teams (asymmetric pair programming typical of iAgile). Moreover, the reduction of non-developing personnel played also an important role.

Since project management responsibilities were in charge of the Agency, the use of internal personnel reduced the cost of hiring industry's senior figures. Also the increase of teams' effectiveness from sprint to sprint led to cost cuts. Due to the incremental domain knowledge acquisition gained through domain experts and user's feedbacks developers were able to produce artifacts which were attained to customer's expectation, decreasing sensibly rework.

5.3 Requirements engineering, management & tracking

Agile software methodologies like Scrum put the development team at the center of the development process removing the major part of the procedural steps of the legacy methods and the connected "milestone evidence" mainly consisting of documents and CASE artifacts [46]. Agility is supposed to increase the production effectiveness and, at the same time, to improve the quality of the product.

However, in order to go Agile, a Waterfall-like static requirement documentation can not be replaced simply with a product backlog. The old-fashioned Waterfall frozen requirement document is no longer effective to capture the user needs in quickly changing mission critical environments. Replacing structured and consolidated text with volatile lists of simple sentences may result, in the case of complex systems, in a sensible loss of knowledge. Traceability of how the solutions are found and both the user and the developer growth may become "implicit and unexpressed knowledge" which are key elements within a high quality software development process.

Several studies suggest to overcome requirements misunderstanding as soon as possible, in order to improve the project results and to decrease development and maintenance costs within its life cycle [63]. This is one of the reason why the Agency started to develop some mission critical software in an Agile way, in order to "shorten the requirement chain", fostering software quality and cost reduction.

The ambiguity concerning the functions to implement is an everyday challenge. Due to the volatility of the user needs, changing of scenarios, and the intrinsic complexity of software products, a dynamic requirement engineering worked very well in an Agile environment [168]. However, the most challenging task is to identify the "tacit dimension" of the user requirements and to surface the "implicit" knowledge of users [335].

In most agile approaches requirements are given in the form of "User Stories", which are short sentences in natural language usually describing some value to be computed in some scenario in favor of some typical class of users. Such formalization drives non-linear human thinking in a standardized form where users have to explain how they imagine the system. This approach has been implemented for mission critical applications. Capturing users requirements and overcome the intrinsic ambiguity of the natural language are two of the main goals of this effort. Fully refined requirement specification documents are no longer meaningful; instead they should incorporate some guidelines to help the developers to effectively measure the quality of the features so that these can be improved. The result is a novel proposal based on an evolution of the "Scrum type" Product Backlog, here represented:

- *User Story.* A structured sentence which summarizes the functionality. Example:

  ```
  As <role>
  I want to <functionality description>
  in order to <goal to pursue>.
  ```

- *Business Value.* Describes the business value of the desired functionality.

- *User Story Elaboration.* It is an extended user story and it details how the functionality has to be implemented.

- *Acceptance Criteria.* Non functional requirements are given, necessary to accept the functionality (e.g., security, compliance to standards, interoperability issues). Moreover, also functional requirements have to be verified, to accept the developed software. Tests are typically focused on these functionalities.

- *Definition of Done.* It is when the story can be considered fully implemented. The Definition of Done includes the Acceptance Critera and anything that the PO believes is necessary that the team does on the code before it can be released.

- *Expected Output.* It is a list of expected outputs from the functionality, once implemented.

Software development methodologies should be inspired by their organization's needs and not by programming concepts. Well aware of Conway's principle [107] it is the mission need that shapes the information system. Not the structure of the organization, which in our case is highly hierarchic and in its communication flows reflects the Waterfall paradigm. Due to the constant iteration between the users' community, through the Product Owner, and the development team, required applications attains users' expectations. Our experience has shown the effectiveness to overcome the limitations of existing alternatives of a Waterfall like requirement engineering, which is ineffective for complex user requirements, especially in the mission critical domain.

If continuous interaction, typical of Agile, is crucial to overcome structural information asymmetry, which is present in any human interaction [12], experience showed that it is not enough. Any software project, especially Agile, involves different people, with different backgrounds and experiences. In other terms, we all have our "mental models" [234], which are the source of this information asymmetry. Mental models are psychological representations of real, hypothetical, or imaginary situations, identified by Kenneth Craik [111]. They are mental constructs of the world around us. A mental model is a representation of the world around us and shapes our behavior and approach to problem solving. Like a pattern, once we experimented that the solution works, we tend to replicate it. It helps us to not restart from zero any time we have to face a problem. Thus, it is a simplification. So, it is a mind construct of "small-scale models" of the reality, to anticipate events, to reason, and to underlie explanation [111].

To give an example of the difference between the semantic meaning of a nominal identical concept (i.e., difference in mental models) let us consider the notion of "battlespace geometry". Starting from a user story, a PO may write: "as a commander I want to be able to represent the forward line of my sector on a map to see the deployed materials".

The user has in mind a "line" whose geometrical elements (waypoints, start, finish and type) are decided by a superior command post that is given to him as part of a formatted order packet which he expects to appear on the map by a single click of his mouse and to be updated as the tactical situation changes. The developer's first comprehension will be "drawing" tools able to produce a line by simply calling some graphic libraries. The focus is on how to implement it writing the least possible quantity of new code. This is just an example but it qualifies the differences between the two worlds very well.

For the user the image on the video is just a representation of the real world, for instance a real portion of land where the operation is taking place. Instead, for the developer the same object is the result of a series of interactions showing a map on a video where he has to provide a design tool. As trivial as they may seem these differences are at the root of the software requirement specification problem that in the past has been tackled by freezing the requirement text and operating a certain number of translations into formal representations without reconciliation of the two different mental models.

Some concepts developed in conceptual semantics explain how the representation of the world expressed in natural language is the result of a mediation between the speaker's perception of the world and the current setup of his own mind (i.e., mental models). This poses the question on what we really do communicate about requirements when we use natural language. In [224] this problem is studied, and a solution based on feature maps is proposed.

In our case what emerged is a common ontology used by both users and developers. We found out that working on the ontology in the initial production process

(i.e., Product Backlog) improved the effectiveness of the Agile approach. In fact, the development of a Command and Control ontology, useful as knowledge representation tool as described in the next section, is also effective to merge different mental models and to support requirements traceability [462].

5.4 Use of KBS and OBS within iAgile

The use of Ontology-Based Systems (OBS) for managing requirements and user stories when applying Agile methods has been explored many times, but it is still an unresolved issue [271, 297]. Some literature suggests that ontology driven development should be the norm, both in general and specifically in the Agile arena [259].

The use of OBS is of paramount importance in a mission critical context. We have experienced it in peace-keeping operations, where rapid information flows coming from different actors (e.g. military, NGOs, citizens, press.) have to be processed. The different needs, contexts, and objectives of these actors are often reflected into a wide range of viewpoints and assumptions, producing different, overlapping and/or mismatched concepts and structures that essentially concern the same subject matter.

Different "organizational routines" [**Nelson2009**] lead to different communication standards, along with "tacit knowledge" [367]. Thus, there is the need to organize the different "mental models" [370] around the development process. Ontologies are a powerful tool to overcome this lack of a shared understanding, providing an unifying framework for the different viewpoints and mental models that coexist in vast and heterogeneous contexts.

As described in [462], this shared understanding provides many practical benefits, such as serving as the basis for human communication (reducing conceptual and terminological confusion), improving the interoperability among systems with different modeling methods, paradigms, languages and software tools, and supporting the main system engineering principles (such as reusability, reliability, requirements identification, system specification, etc.). The adoption of ontologies as a core components of software architectures [98] in conjunction with Agile methodology development principles has proven its effectiveness in changing and variable contexts.

An overall idea of the main elements of the development process is depicted in Figure 5.2. Both KBS and OBS are build on users' mental models. This means that requirements and the ontology represent user's view and needs. So, the user stories collected are the core elements. Following Agile methodologies principles, such artifacts are fundamentals to distill both information about the system to develop, and knowledge on the domain in which such system is expected to operate. User stories have been used to extract the requirements of the C2 system, and to develop an ontology for representing the main concepts of the mission critical domain.

So, the backlog grooming (i.e., the refinement of the user stories) becomes the instant where users stories split. Requirements are defined and the ontology is developed. This split is not straightforward. Considering that ontology's entities definition is very helpful to define better user's expectations, requirements documents are developed separately from the ontology. While requirements are developed manually by the development team, the ontology is developed by Protégé[2] [260]. As shown in Figure 5.3, developers already exposed to semantic technologies use this standard tool to develop the domain specific ontology.

[2] http://protege.stanford.edu

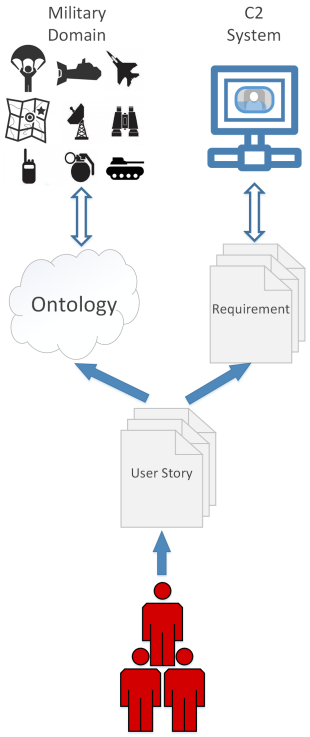


FIGURE 5.2: The ontology of the application domain and the system requirements are derived from user stories

5.4.1 An Ontology-based Architecture for C2 Systems

One of the main challenges in the mission critical domain is the ability of managing in a precise and accurate way the complexity, variability and heterogeneity of information. In particular, the ability of integrating different sources of information, extracting the most relevant elements and putting them into the context is of paramount importance for supporting the tasks of control and decision making. In our approach, ontologies and related technologies are the main tools for facing both the methodological and technological aspects of such context.

Similarly to other scenarios, also in the the mission critical domain people, organizations, and information systems must communicate effectively. However, the various needs, contexts and objectives are often reflected into a wide range of viewpoints and assumptions, producing different, overlapping and/or mismatched concepts and structures that essentially concerns the same subject matter. Ontologies are often used to overcome this lack of a shared understanding, providing an unifying framework for the different viewpoints that coexist in vast and complex C2 systems.

As described in [462], this shared understanding provides many practical benefits, such as serving as the basis for human communication (reducing conceptual and terminological confusion), improving the interoperability among systems with different modeling methods, paradigms, languages and software tools, and supporting the main system engineering principles (such as reusability, reliability, requirements identification, system specification).

The central role played by ontologies is summarized by Fig.5.4, which depicts the overall architecture of the C2 system. The ontology we have developed describes the data model contained in a Knowledge Base (KB), which contains all the information needed by the C2 system. The KB is populated by ad hoc software components (i.e. adapters), that extract information from all the different sources (e.g. legacy

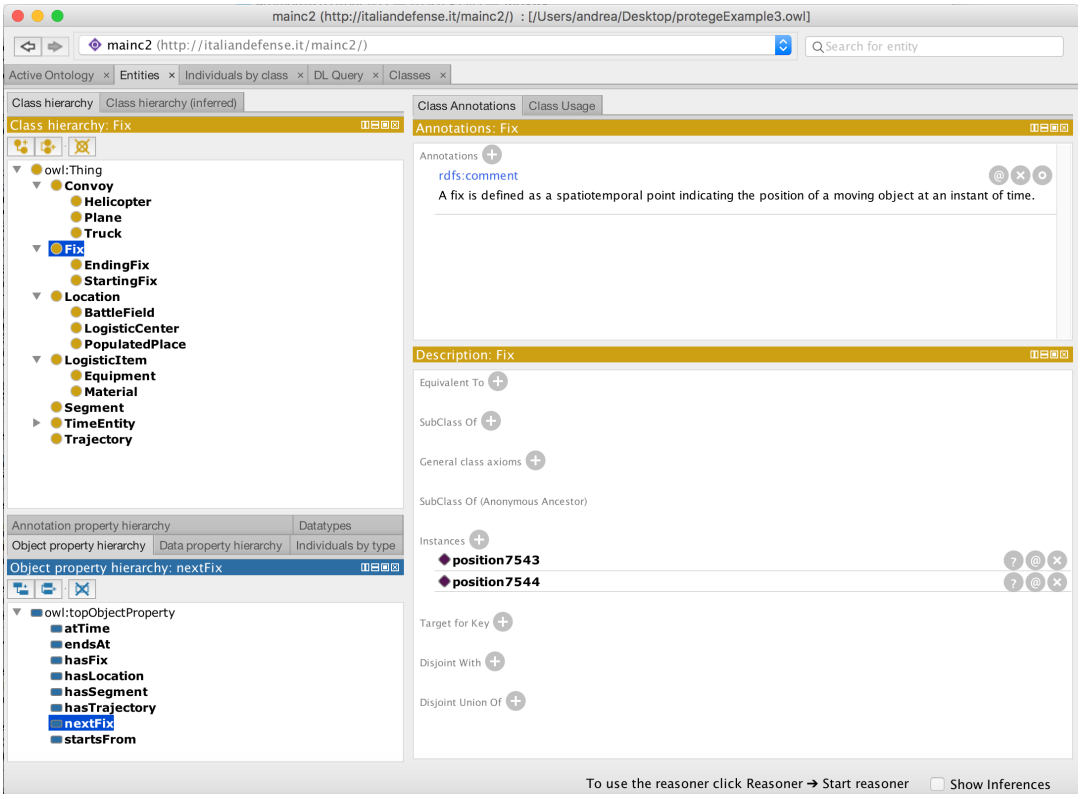


FIGURE 5.3: A snapshot of the C2 ontology during its development with Protege. The class and property hierarchies are shown on the left, while other contextual information (e.g. annotations, instances and relevant properties) are shown on the right.

tactile systems, news or ONG CMS, etc.), and convert it in semantic statements that conforms to the ontology. The frequency of the KB populating processes varies from sources to sources. Where possible, triggering mechanisms have been used to identify modifications in the sources and activating the data extraction. Otherwise, adapters performs the data extraction at fixed (and configurable) intervals. Of course, high data quality is a major issue here [97].

The KB contains provenance information defining:

- the origin of the data;
- the agent (usually a software component) responsible for the data extraction;
- other useful metadata (e.g. time information, type - such as insertion, deletion, update - of the KB modification, etc.).

The main advantages of having such information are the ability to discern among different authority levels, the capability of performing comparisons and advanced filters, just to cite a few. Moreover, from a technical point of view, provenance information are of paramount importance to have full control over the KB state during the whole system lifecycle (from the inception and development phases to the final operating period).

The Main C2 system is depicted at the upper vertex of the star architecture. It implements all the functionalities required by the users and described in the collected user stories. All the data required to expose such functionalities to the users are

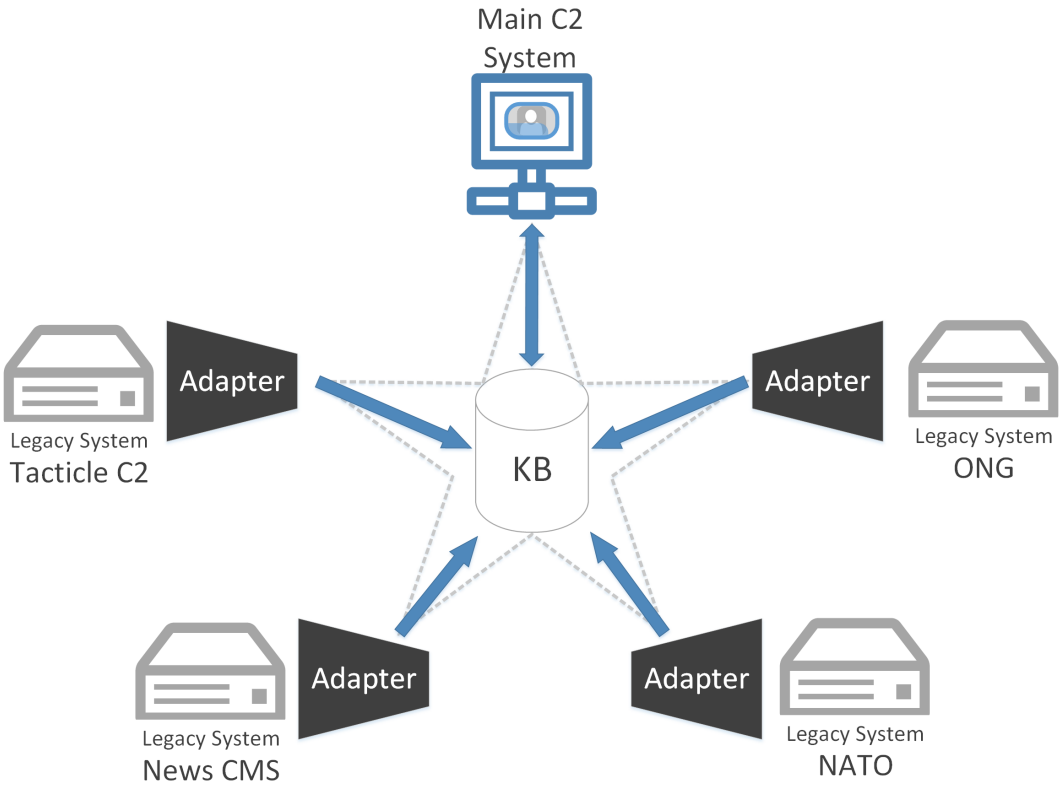


FIGURE 5.4: The C2 system is modeled around the star architecture pattern. The domain ontology is the center of such architecture, and is used to integrate different resources and systems.

retrieved from the KB by means of standard semantic queries. The same mechanism is used for adding new information in the KB (e.g. for keeping track of the output of the system users' analysis and decision processes).

5.4.2 Developing domain ontologies from user stories with iAgile

Ontologies play a crucial role in the development of the framework. The primary objective is to develop an ontology that is capable of modeling the complexity of mission critical domains. The starting point and primary source of information for this task is the set of 600 user stories collected from the system final users and domain experts. User stories have been grouped in small buckets (having between 5 and 10 user stories each). The process started by considering a first bucket, that has been used to develop an initial model. Finally, the ontology has been developed iteratively, by adding a new user story bucket to the set of already considered ones at each cycle.

At the end of each cycle, a small dataset (*test dataset*) is created according to the current ontology and considering the user stories under examination. Such a dataset is used to perform a *quality check* on the current ontology. In practice, tests are a series of queries that are derived by analyzing the functionalities described in the user stories. A query test must be executed on the test dataset to check the ontology validity after the modifications performed in the last cycle. In case of a positive result, a new user story bucket is considered and another development cycle is performed. Otherwise, the ontology is refactored and modified until the quality check is satisfied.

In order to clarify the ontology development process, we present in Figure 5.5 a bucket composed by eight real user stories collected for the C2 system.

US 2825 (2656)
As <system user>I want <to figure out if the track relative to an object (equipment) represented on the map has been updated, or if the position has just been changed.><Such a status would be highlighted by a green border around the icon on the object. This function should be enabled or disabled by the user.>

US 2828(2659)
As <commander of the logistics>I want <to view a summary, in both tabular and graphical format (e.g. histogram), of the efficiency logistic items.><The total of logistics items and the level of efficiency should be displayed in percentage of the total.>

US 2829(2660)
As <commander of the logistics>I want <to be able to view on a geographical map the geographical distribution of identified logistical items (e.g. equipment, materials, etc.).><The map shall display the concentrations of materials at the logistic centers, represented by a specific colored icon associated with the type of material. The color should reflect the overall efficiency of the logistic item. For each logistic center, it should be possible to view the amount items, with an histogram graph showing both total items along with the percentage of efficiency.

US 2822(2653)
As <system user>I want <that the system can receive and display information about convoy (e.g. trucks, helicopters, planes, etc.) - for example Moving Convoy, Halted Convoy, etc.>

US 2821(2652)
As <system user>I want <that the system can store information about convoy (e.g. Moving Convoy, Halted Convoy, etc ...).>

US 2820(2651)
As <FAS web user>I want <to be able to view on a geographical map the geographical distribution of identified logistical items.><The map shall display the concentrations of materials at the logistic centers, represented by a specific colored icon associated with the type of material. The color should reflect the overall efficiency of the logistic item. For each logistic center, it should be possible to view the amount items, with an histogram graph showing both total items along with the percentage of efficiency.>

US 2819(2645)
As <FAS web user>I want <to be able to view on a geographical map the geographical distribution of identified logistical items.><The map shall display the concentrations of materials at the logistic centers, represented by a specific colored icon associated with the type of material. The color should reflect the overall efficiency of the logistic item. For each logistic center, it should be possible to view the amount items, with an histogram graph showing both total items along with the percentage of efficiency.>

US 2711(2296)
As <system user>I want <to display on the map the trajectory and the last positions of a specific object. This functionality should be enabled or disabled by the user.>

FIGURE 5.5: User stories collected with iAgile are used to develop the system domain ontology

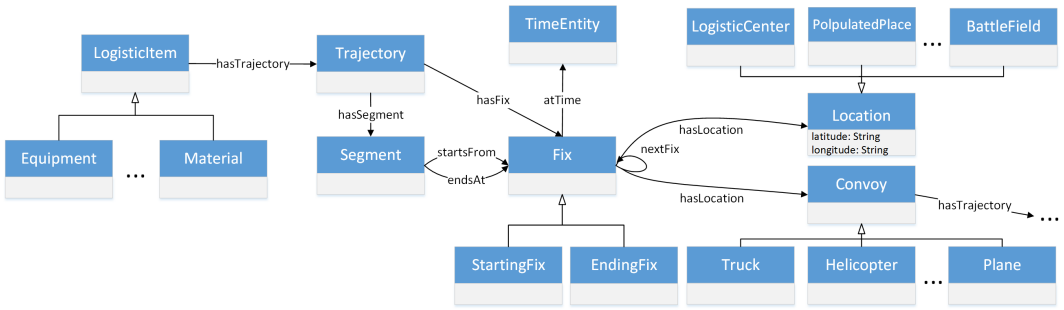


FIGURE 5.6: A fragment of the developed domain ontology. Three general concepts are represented (i.e. logistic item, location and convoy), and the trajectory pattern has been used to model positional information

Each user story is identified by a unique identifier. User stories have a common fixed structure, where users' roles, objectives and user story specifications are easily identifiable. During a further step of analysis, the concepts that are the main candidates for becoming classes in the domain ontology have been underlined.

An excerpt of the ontology developed starting from the eight user stories is depicted in Figure 5.6. Three general concepts are modeled: logistic item, location and convoy. Well known ontology engineering principles and best practices described in [178] have been used throughout the whole process. As shown in the ontology fragment, for example, the trajectory pattern [218] has been used to model objects positions and movements. This is an elegant solution for attaching trajectories composed of segment with a geographical or physical extents to any object of the domain.

The process of ontologically modeling the domain has many practical benefits. For example, we successfully converted requirements and constraints to the data model in semantic assertions. Such restrictions can be automatically checked by using popular semantic tools (e.g. reasoners). For instance, we can imagine to add an assertions that states the segments of a trajectory should not be in overlap with respect of both the time and space dimensions. In another words, we can impose that an object should not have two different positions at the same time. All the cases that do not respect such limitations in the data are automatically identified as inconsistencies stated by the semantic reasoners.

5.5 Conclusions

Knowledge Based Systems and Ontology Based Systems are key elements for the development of software-intensive mission critical systems. We reported about a real case study concerning a mission critical system developed for an Italian governmental Agency. Volatile requirements and fast changing operational scenarios led to the choice of a new development process model, transitioning from Waterfall to Agile. However, Agile is not a *panacea per se* but needs to be adapted for complex mission critical purposes. We customized Scrum into *iAgile*, developing a flexible but structured paradigm. Ad hoc steps were designed to comply with both velocity and security. Moreover, we found that such methodology led to an important saving of development and maintenance costs.

The role of the KBS we have used is double: to disambiguate requirements and to build an ontology for interoperability and knowledge representation. The ontology was designed using semantic tools during the requirement specification. Moreover, at the

same time, the user story elaboration is carried out for the functionality development. This process allowed to align the different mental models of users and developers. Therefore, after the first formalization of entities, it supported the next Sprint backlogs with a high relevance to users' expectations.

The big advantage of the use of an ontology derives from the interoperability with other legacy systems. In a real-world operational scenario, the mission critical information system is fed by information of different provenance. As shown, also non-governmental actors may deliver useful elements for an up to date situational awareness. So, from different sources it is possible gather data in a flexible and incremental way improving the information completeness, necessary to take mission critical decisions. Moreover, through the relationship between ontologies of other governmental or intergovernmental agencies both interoperability or replacement of such system is highly simplified. Since the Agency is supposed to offer security services in multilateral and multinational operations, interoperability is of strategic importance. Therefore, the presented approach is an important driver for a smooth and effective system deployment in a mission critical environment.

Future research will go in several directions.

A comprehensive approach to OBS, based on the acquired experience, will be implemented within the Agency with the aid of knowledge-based tools. Moreover, a Machine Learning approach in which requirements are automatically processed to assist the continuous development with Scrum [403] has also to be developed. Also, the use of concurrent development methodologies to support velocity and reliability has to be improved [401], along with a flexible system's architecture. Especially, the reliability in terms of systems "antifragility" of mission critical applications needs further investigation [402], [406]. Although we are aware of the relationship between the software quality dimension and its architecture [405], still efforts need to be pursue to figure out how Agency's business goals (e.g., velocity, cost reduction) impact on the system. Finally, also issues related to software reuse have to be explored [102, 99], since the cloning practices, also in critical systems, is quite common.

Chapter 6

Agile Contracting

6.1 Introduction

In several commercial domains Agile development has been effective for building new software systems or to evolve an existing one rapidly, decreasing development costs. The role of IT, intended as value-focused support for the design and implementation of digital technologies, disrupted new product and service developments [371, 372]. While consumers are becoming more keen to use technology for their daily applications, businesses are rethinking about customers value and the relative business models for their competitive differentiation [50]. Therefore, entire industries restructured their business processes to deliver new capabilities and goals to the new business model [345]. As software becomes more essential to the world's day-to-day activities, the community is calling to move Agile software development beyond a "craft–based approach to become a true engineering discipline" [226].

In this scenario, a business area of particular relevance for the size and the number of opportunities are mission & security critical systems, especially for defense applications [101]. These applications are usually very expensive and are managed with specific care from public administrations which bid contracts to external software houses. Apparently, a Waterfall software development process model responds to some fundamental needs in such organizations, like (i) a clear definition of the costs, (ii) early requirement definition, (iii) predefined schedule, and (iv) tracing liability if something goes wrong.

However, when costs rise exponentially during maintenance due to poor software quality of the deliverables or the loose requirement implementation, Waterfall shows all its limits. Moreover, velocity is a crucial factor for such organizations. Military operations need do be deployed within a very short time range on very different scenarios [**MFRCR16**, 100]. Information systems have to evolve accordingly. Waterfall, is not a suitable paradigm, since it is not enough flexible, expensive, and it fails frequently [434].

Although there are industry–scale Agile development techniques for the development of critical systems, like SAFe [284, 143], contractual aspects are typically overlooked. With regard to contractual issues, the adVANTAGE framework is a potential model for commercial organizations [67]. When a project is executed following a contract compliant with the adVANTAGE framework, a priority list of requirements in the form of user stories - namely a backlog - is managed jointly with the customer. Such a backlog is addressed in a sequence of Sprints. Each Sprint is a project phase of fixed duration, usually from two to four weeks, and realizes the software satisfying some of the top requirements found in the backlog. The development services are billed according to the quality of the software delivered at the end of each Sprint. If the contractor fails to realize the user stories agreed for a certain Sprint within the available jointly agreed target budget, the additional costs will be charged at reduced

daily rates. This approach or one similar is often followed in contracts between private companies.

However, the lack of concrete proposals, their experimentations, and discussions in technical and scientific forums of Agile contractual models suitable also for the public sector is one of the reasons why top managers are not keen to this development method. Accordingly, this work is an attempt to overcome such limitation.

We propose a foundational approach connecting the theory of how contracts should be organized with Agile practices, using the Italian context as a reference, and also identifying the key issues of Agile contracting which need further development.

The chapter has the following structure. Related work are presented in Section 6.2. In Section 6.3 Law & Economics of contract theory are briefly explained to understand the underling logic of software contracts. This interdisciplinary approach is crucial to understand the economics of contracts, i.e., alignment of interests, which is the most tricky part of Agile contracting. In Section 6.4 we deepen the Italian case, defining the key elements of the procurement law. After gaining a short understanding about the basic legal boundaries for Agile public contracting, we illustrate two approaches (Section 6.5); the first one (6.5.1) is based on Function Point Analysis; the second (6.5.2) is based on Scrum Sprints. A case study detailing how Agile contracting has been concretely implemented in the Italian Defence Acquisition Process is described in Section 6.6. Finally, in Section 6.7 we sum up our main proposal and envision some further work.

6.2 Related Work

The problem of Agile contracting is old. Basically, the main difference with respect to contracts for Waterfall developments, that are based on measuring and compensating effort spent during the process, is that fixed price seems more adequate for agile developments. The Agile fixed price is a contractual model which includes an initial phase after which budget, delivery dates, and the way of defining the scope of the system being built is agreed upon.

For instance in 2006 Alistair Cockburn published online an intriguing discussion of some typical Agile contracts[1]. His page lists several possibilities, like the following ones:

- fixed price, fixed scope, fixed time;
- fixed price, fixed time, negotiable scope;
- paying for effort as it gets spent. If the requirements are volatile and there is mutual trust among producer and consumer this is the best situation;
- max price with payment on incremental acceptance: it works with stable requirements;
- incremental delivery with payment on incremental acceptance
- price for each unit delivered, for instance a fixed fee for function point;
- base fee for each unit delivered, plus a low fee per hour, in order to incentive developers to early delivery.

[1] http://alistair.cockburn.us/Agile+contracts, retrieved on September 19, 2017

Similar contractual cases are also discussed in the work of Pilios [365]. Further aspects of an Agile contracts are risk share (customers and developers compensate the additional expenses for unexpected changes equally amongst themselves) or the option of either party leaving the contract at any stage (exit points).

Indeed, Agile methods tackle those issues, trying to align the interests of the development team and the customer. Our interest here is for Agile applied to mission critical systems sponsored by public institutions, with specific application to Italian public institutions.

In some earlier works we have reviewed the enactment of Agile software development methods within mission & security critical organizations, especially military ones [**MFRCR16**, 100, 401, 403]. Moreover, in the last years the debate around the use of Agile contracting also for commercial uses became a trending topic [23, 349, 66, 67]. In [23] the authors criticize traditional contracts for software development, which increase the risk of failure because requirements are frozen due to the sequential work flow, leading to a low quality of design, and causing a poor return on investment. The book [349] suggests fixed price or maximum price contracts for Agile developments, in contracts structured as follows. The contract has to describe to what extent, in percentage, the costs incurred by the supplier will be charged to the customer when the maximum price range is exceeded. A period of n Sprints is agreed upon as the test phase of cooperation. The final milestone is a checkpoint whereby the customer and supplier can enter into the real development of the project or maybe exit in a controlled manner. Another, more recent book [67] expanded a contractual model called adVANTAGE for Agile Developments, already sketched in [66]. This model puts specific focus on the willingness of the contractor to take some of the (apparent) risks of development that come with Agile practices. Still, it is not suited for public administrations, since it does not consider constitutional specificities, outlined in Section 6.4.

Recently the US government has devoted a lot of attention to the problem of Agile contracts. The Software Engineering Institute (SEI) has released in the last five years several reports concerning Agile for producing software products in particular for the military [278, 350, 277, 336, 337, 493]. It has also published some guidelines for Agile contracts for software acquired by the US DoD [492, 279]. These guidelines compare traditional developments with Agile developments for critical military systems. The major recommendation consists of post-award documenting contractor's performance throughout each Sprint and release e.g., using metrics like SQALE [286] for measuring technical debt in terms of effort, or bug defect rates, length of throughput time compared to contractor estimates, speed of time to value.

Nevertheless, despite these efforts, the problem of understanding how to lay down contracts for Agile development for mission critical products is still at its infancy.

6.3 The Law & Economics of Agile contracts

Contracts are agreements between two parties, with different interests, written down to fix such interests, alongside with some results compensation. Generally speaking, for a free-market economy, the ability of two parties to enter into voluntary agreements, namely contracts, is the key element for the market equilibrium [206]. Contract law and law enforcement procedures are fundamental for the efficiency of any economic system. Thus, contract law has to be intended as a set of rules for exchanging individual claims to entitlements (i.e., interests). In this way, it enforces the extent to which society gains from this agreement due to an efficient economic system.

When one party is unsure about the other party's behavior, contracts may mitigate this asymmetry. In our case, contracts are helpful when advance commitment enhances the value of an artifact by enabling reliance by the beneficiary [373].

From a Law & Economics viewpoint, there are several issues regarding the importance of contracting [206].

- *Coordination.* The most common reason to engage in a contract is to coordinate independent actions in a situation of multiple equilibria. The most straightforward example is the well known Prisoner's dilemma. Without coordination, two parties with different and independent interests will choose the scenario where both are worse off (i.e., both confess their crime and accuse the other party, in order to reduce the imprisonment time as a benefit). With coordination, on the contrary, both would get the better payoff, not admitting the crime, gambling the law system, escaping from a long imprisonment time. If the parties are well coordinated by a contract, they will get both the best trade-off, not going to jail at all. A contract to play this efficient equilibrium guarantees a positive outcome. This is also known as Nash equilibrium, where modern contracting theories get most of their inspiration. The coordination scenarios based on contracts are excellent models to understand institutions [330].

- *Exchange implementation.* Especially in situations of hidden informations (i.e., information asymmetry occurring when one party has an information which the other party does not have), contracts may mitigate such asymmetry [12]. To avoid adverse selection, which impedes market efficiency, contracts may provide warranties, to assuring the high quality of the product. This is very typical in software, where the vendors know the details of the product, while the customer is totally unaware of the code (usually obfuscated, if it is a licensed product) but only aware about its functionalities told by the vendor. Thus, alongside with software, there is usually a warranty about the product. In this way the customer potential downsize (bugs) will be fixed by the vendor and no special code awareness is needed before buying the software.

However, there are also some major drawbacks of contracting [206]. The most important from our point of view are:

- *Ex post: specification cost.* Writing down all possible contingencies which could arise within the future contractual relationship is extremely expensive. Potential contingencies of contractual obligations are usually very broad. Therefore, contracts are often left open and incomplete. In such cases there are two main scenarios. It could happen that the contract just fails to provide information for contingencies, since nothing was agreed upfront. In this case, parties have to decide what happens after a contingency. In the second case the contract could cover a broad number of contingencies but not fine-tuning them. In such way, parties still have to decide what to do, since contingencies are not defined precisely enough. Anyway, in both scenarios, contracts fail to assure the commitment of the parties.

- *Ex ante: dynamic inconsistency.* This is the classic investment problem. One party may be willing to bargain and to modify the contracts when it has pursued investments. Suppose that a vendor has found that the software can easily have more functionality or higher quality even with limited additional costs. However, the price has been set, so the motivations to deliver such higher value

are minimal if the customer is not ready to agree, which is in turn difficult because the intangible nature of software may make difficult for the customer to understand the nature of such modifications (see the next point). In essence, vendors may not have any incentives to do investments, i.e., spend money to develop high quality code.

- *Unverifiable actions.* Even after entering into a contractual commitment, one party may be unable to determine whatever the agreement has been kept or broken. This is the typical case of intangible goods, like software. It is a not trivial task to assess with objectivity if what promised has been carried out according to the contract.

While we analyse Agile contracting, we should avoid the risk of overstating its problems and overlooking normative and incentive aspects, typical of any contractual relationship. The economics of contracting has both upsides (i.e., coordination and exchange implementation) and downsizes (i.e., specification cost, dynamic inconsistency, and unverifiable actions). What we learn from the Law & Economics theories of contracts is that any contract has its loopholes, thus also Waterfall ones.

Waterfall contracts are well known, in a sense they are easy to agree initially, because parties are usually fully aware of them and there is a history of people using them - meaning that it is more difficult for a manager to be subject to criticism for adopting them[2]. Both specification costs and unverifiable actions have a big impact on the cost of contracting. Traditional software contracts are very expensive; alongside with high specification costs due to very detailed requirements, there is also the difficulty to assess with objectivity the artifact to build. Such barriers have a direct impact on both the contract cost and market efficiency. In Waterfall contracts there are indeed "hidden" costs that indirectly increase the cost of software products. The perceived "reliability" of Waterfall has apparently scarce evidence in practice. What we do know is that Waterfall usually increases the maintenance costs, which are hidden costs belonging to the software's life cycle [377]. However, as mentioned, while there are established routines concerning how to carry out a Waterfall contract, there are very few guidelines about Agile.

First of all we will depict the divergent interests of a software contract, represented in Table 6.1. As seen before, contracts facilitate market equilibrium through coordination and exchange implementation. In software this means that the two parties which suffer from an information asymmetry reach an agreement through a legal binding paper (the contract). A generic organization does not always have the expertise or the man-power to carry out the software, while contractors do. There is asymmetry in the sense that both parties are not aware of the same relevant information, i.e., the (i) price willing to pay, (ii) technological complexity and feasibility, (iii) code reuse, (iv) implicit needs of the customer which may not correspond to requirements. Such problems are overcome with a binding agreement.

[2]Remember the old adage "None has been fired for using IBM." which fully expresses the criticism manages can have by trying new, not yet consolidated, approaches. This phenomena is also exploited by vendors as it is well explained in the *Wikipedia* page on "Fear, uncertainty and doubt" accessed at URL: https://en.wikipedia.org/wiki/Fear,_uncertainty_and_doubt on Sept. 10, 2017.

TABLE 6.1: Divergent interests

	Organization	**Contractor**
Requirement interpretation	Broad	Narrow
Time to market	As soon as possible	Depending on several issues
Quality & Security	Best	Good enough to get paid
Cost	As low as possible	As high as possible

However, some latent interests are not aligned by any contract, due to specification costs, unverifiable actions, and dynamic inconsistency. If time and cost are fixed, requirements have a degree of interpretation but they are easily quantifiable; it is quality and security which belong to an arbitrary or "subjective" dimension which are the most difficult parts to fix in any software contract. Loose quality and security software means unsustainable raising maintenance cost in the long run. Especially mission critical organizations may loose operational capability due to the complexity and low quality of their multi-party systems. Therefore, there is a stringent need in any field to align organizations and contractors interests, in terms of customer needs, quality & security, costs, and time.

From a project management perspective, divergent interests can also be explained with the Iron Triangle [22], represented in Figure 6.1. Scope, schedule and budget are typically opposite concepts, where the optimum is represented by a balanced relation [22]. The variable that does not change is quality, which is the major concern of every software project. A software project with a broad scope, and tight schedule and budget will reasonably be of poor quality. Similarly, in the case of high budget availability but with a stiff schedule and broad scope will also deliver poor quality software due to general low predictability of software project management for short time frames. Finally, the scenario where a software project has to be completed within a short time and low budget with a broad scope is quite unrealistic. Still, the Iron Triangle is extremely useful to understand the three project management drivers for a software system with certain quality standards. When parties with divergent interests engage in a contractual relationship, both need realistic expectations. An unbalanced relationship, which may be considered with favor in a first moment, will unlikely deliver a satisfactory outcome, in terms of software quality. Consequently, the substantial aim of Agile contracting is both to understand and to align such divergent interests. However, those interests are rarely clear ex ante, at the beginning of the contractual relation. Therefore, the continuous specification of those interests are pivotal for a successful project outcome. Clearly, this wisdom fits particularly well to systems which scope evolve in time. So, to tailor the scope (i.e., requirements specification), along with adequate budget and schedule, multiple contractual agreements are needed to formalize each Agile iteration. Consequently, we propose in Section 6.4 a *double–level contracting structure* to enhance the ongoing interest alignment.

Our idea is to develop a *bonus-malus* reward system. In such a model, the price is fixed and represents the maximum awardable amount. According to the development process and product quality obtained, the contractor is paid according to what is delivered and measured. To do so, there must be a quantifiable measure of some kind

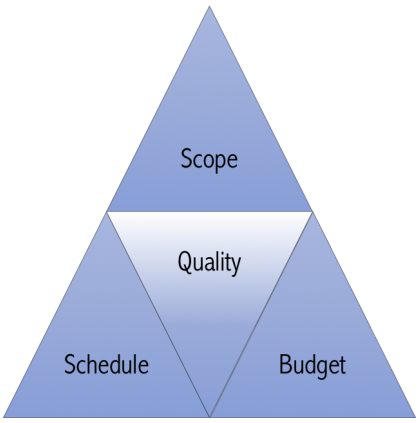


FIGURE 6.1: Iron Triangle [22]

of software size dimension. With all their limitations, we do believe that Function Points [16, 407], or some related variants like *Simple Function Points* (SiFP) [158], represent a measure that is good enough for our purposes. To avoid specification costs, contracts should have a loose - in some way open - requirements list, but a fixed, predetermined SiFP estimate. Moreover, a *bonus-malus* mechanism should be added alongside within the pricing. After each iteration i.e., implementation of user stories, SiFP are consumed and paid. The *bonus-malus* pricing mechanism in this case means that with a high quality code, contractors get a bonus, up to the maximum (fixed) amount. Instead if they deliver a release which exposes some technical debt contractors get some kind of fine, to be recovered after the debt is repaid. Clearly, a critical issue is how to measure the technical debt. Some modern tools like SonarQube are able to measure technical debt [83]. As any metrics, both FP and SiFP have some limitations. For this reason we do not claim that they are the ultimate solution to solve the problem. However, SiFP is an easy measurable metric for business functionalities, which are very close to the Agile definition of User Story. Code quality control is still necessary, to avoid the malicious use of low quality functions, just to increase pricing. Therefore, it is of greatest importance to fix such test and metrics within the contract, even if not implemented. Based on our experience, we suggest that security and code quality should be defined as non-functional requirements in the development process. Especially in mission critical organizations we see how some redundancy of competences within the process improves code quality and security [307]. Thus, a TDD (Test Driven Development) approach set in the contract seems quite suitable for Agile contracting. Within each iteration, the Product Owner (PO) and the contracting development team start with a test oriented development, which has to correspond to the user story development.

Our main idea is that continuous "tensions" and new equilibria between the two parties are the best mitigation drivers that underly to any contract. Continuous discussions, bargaining, and agreement do motivate both parties to carry on their respective tasks. In such way, we think that we can obtain the following results:

- Specification costs are limited, since Agile contracts do not only specify the very general task at the beginning but they agree the details of every user story upfront each iteration; this is a sort of overarching or framework contract.
- Dynamic inconsistency is attenuated through a reward based payment. Contractors will have the economic interest to get the "bonus", which is awarded according to their performances.

- Unverifiable actions are mitigated by a TDD approach, since "quality metrics" i.e., tests, are agreed by the parties within the iterative development process.

Such approach is particularly effective for public administrations, which by our law must use a bidding base. Following our proposal it is possible to define a budget a priori and, at the same time, contractors will work for better quality software, trying to gain the whole amount. Organizations and project owners gain from velocity and requirement satisfaction. From an operational point of view, this solution tackles each critical point that Waterfall does not structurally solve.

Finally, from a contractual perspective, i.e., the economics of the contract, this solution gets all the benefits of contracting, namely coordination and exchange implementation. At the same time some major problems of Waterfall contracts (specification cost, dynamic inconsistency and unverifiable actions) are substantially reduced.

6.4 The Italian Case

Although we are now referring specifically to the Italian case, these considerations are of use also for other countries based on European public procurement rules. In fact, regulation may slightly change, but the constitutional assumptions and procurement characteristic are basically the same or, at least, comparable. For this reason we believe that this research is of good use also beyond Italian borders.

The structure of the procurement law follows some basic constitutional principles, comprehending:

1. free competition,
2. equal treatment and non-discrimination,
3. transparency,
4. adequacy and proportionality.

These are substantial issues, which are always reflected in any concrete application of the law. In the following subsections we will try to explain how to structure an Agile contract, according to those pillars.

6.4.1 The object of the contract

The contractual object has to be *determinate* or *determinable*, according to art. 1346 of the Italian civil code (cc in short). So, the object of the contract needs to be clearly identifiable without further arbitrary decisions. This means that a collaboration program can not be just agreed upfront, if it is not sufficiently determined. At least, some characteristics of the future software product have to be defined. Moreover, according to the procurement law (D.Lgs. nr 50/2016, art. 23.15) the public bid should include a **technical annex**, composed by:

1. calculation of the alleged cost;
2. financial statement of total charges;
3. specific descriptive and performance specifications;
4. minimum bid requirements;
5. awarding criteria;

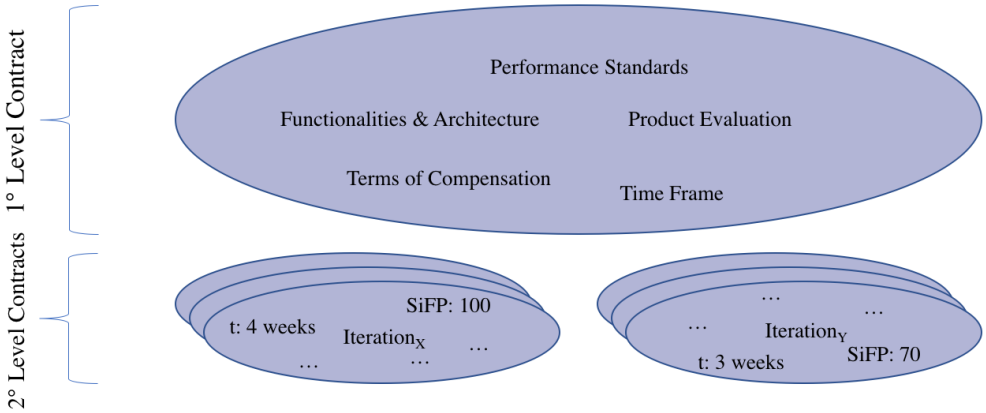


FIGURE 6.2: Structure of the contract

6. possible variations;
7. the possible circumstances of (non substantial) change of the negotiating conditions.

The technical annex is of pivotal importance for Agile contracting, since it is the document (or the set of documents) where the public customer describes the required system and prescribes the methodology. Interestingly, the procurement law applies easily to Waterfall-like contracts but does not hinder Agile contracting *per se*. Consequently, the object of Agile contracts (i.e., the software system to developed, evolve or maintain) needs to be defined *ex ante* at least in functional terms, with the possibility to refine requirements along the way. Thus, a corresponding well fitting structure of the contractual relationship is now proposed.

6.4.2 The structure of the contract

Our proposed solution for Agile contracts consists of a **double-level contracting**. To clarify our idea, see Figure 6.2, which exemplifies the proposed structure.

In European public procurement law, this is allowed by the rules concerning the framework agreements. According to article 33 of the directive 2014/24 EU, a contracting authority may conclude such an agreement, observing the procedures provided by the directive (i.e. the ordinary awarding procedures). In fact:

> "*In general terms a framework agreement means an agreement between one or more contracting authorities and one or more economic operators, the purpose of which is to establish the terms governing contracts to be awarded during a given period, in particular with regard to price and, where appropriate, the quantity envisaged.*" §33.1, Directive 2014/24 EU.

The purpose is to establish the two–level contractual governance. In our case, such a structure between one or more authorities and a single operator is appropriate. The first level contract defines, in general terms, all customer's needs, and in particular the context in which the software will be used to meet such needs:

- the high level definition of software's functionalities and architecture;
- the quality and performance standards to be reached (minimum standards and higher ones, possibly to be paid more by the customer);

- the criteria for the the product evaluation and possibly the definition of done;
- the time frame;
- the general terms of the compensation.

The framework agreement should be limited in time: e.g., not exceeding 4 years. With second level contracts, parties agree the specifications within a variable number of iterations, considering the points listed before, as sketched in Figure 6.2. Before each iteration both parties fine tune the first level issues, in order to meet iteration's scope. In particular, the object of each iteration is specified, along with software's functionalities, performance standard, product evaluation, terms of compensation and time frame. In other words, the framework agreement (first level contract) opens the way to a number of subsequent detailed contracts for the execution of the whole project, each of which can be adjusted according to the results of the previous ones.

The mentioned public procurement rules permit the awarding of such single contracts to the contractor, under the framework agreement, without any new competitive procedure. The only exception is if the contracting authority changes, or if the object of the framework agreement is substantially modified. The key provision therefore can be found in the European directive, according to which:

> "*Where a framework agreement is concluded with a single economic operator, contracts based on that agreement shall be awarded within the limits of the terms laid down in the framework agreement*" and "*For the award of those contracts, contracting authorities may consult the economic operator party to the framework agreement in writing, requesting it to supplement its tender as necessary*" §33.3, Directive 2014/24 EU.

Indeed, the possibility to design a multi–level contract, where specifications are negotiated before each iteration, is an important legal tool for Agile purposes. From our point of view, supplementing the tender would mean negotiating and finding an agreement on the fine tuning aspects necessary before each one of the progressive functional steps or scrum sprints Moreover, such a double-level structure of the contractual relationship between the customer and the contractor has the advantage that the framework agreement does not oblige the customer to engage in the second level contracts.By this way, the customer always has the power to terminate the relationship, after each iteration. This may happen for several reasons, like for instance any dissatisfaction of the relationship, although the general budget of the framework agreement has not been fully spent.

6.4.3 The competition

The competition is a key element for public procurements since it guarantees constitutional rights, such as open concurrency, impartiality, and accountability. It is basically a trade-off between such rights and the utility of the contract. In other words, although it would be more effective to deregulate the competition through *ad hoc* designs, constitutional rights need to be uphold. Thus, the competition should ideally be a Pareto-optimal solution between these contrasting forces. The law guarantees certain degrees of flexibility, in order to find the best partner.

In our case, such an opportunity arises from the fact that the competitive awarding procedure would be limited at the level of the framework agreement with its above indicated general elements of the project. In practice, an efficient and effective selection of the general elements is pivotal to make sure that the advantages of the

best offer will be fully reflected afterwards. Doing so will assure that the second level contracts will be best suited for the general scope of the software system, considering the afterwards non competitive negotiation between the parties for the fine tuning of any iteration. Achieving such a goal needs some caution, especially about the link between quality and performance standards, on one hand, and compensation on the other.

6.4.4 Economic value

The determination of the economic value has to be clear and effective. In the case of framework agreements and subsequent second level contracts, it is possible and appropriate to set binding general rules about compensation. This should be clearly stated in the framework agreement, and applied in the negotiation of the second level contracts. Thus, in the first level contract, the parties should fix the compensation structure, detailing the cases of special awards and penalties.

As an example, they could agree that for each iteration, a mix of a fix rate and a decremental person-day rate (in order to encourage the contractor to be efficient in execution) could be negotiated. Moreover, a quality-related awarding system may also be helpful to enhance project's quality, such as a *bonus-malus* mechanism. On such a basis, second level contracts that are continuously negotiated before each further iteration should respect the general awarding schema. However, small modification to adjust specific necessities may be further negotiated to fully align divergent interests. In other terms, once the general criteria are set in the first level contract, the price for each iteration derives from the corresponding negotiated estimation of the cost of the software system, along with the upon agreed calculation criteria. This means that proper evaluation techniques for Agile contracts are not only possible, but also recommended for the reasons explained in Section 6.3. Indeed, the most important issue to preserve in an Agile relationship is the alignment of interests. Since most of possible discussions may be around the effective value of the software, identifying an accountable and clear way to define the economic value, within the exposed provisions concerning compensation of the contractor, motivates both parties to work together to get the best possible outcome.

6.4.5 Provision of accountable variations

Variations are of great interest for Agile contracts, since they introduce the necessary flexibility along contract life. Generally speaking, public procurement rules make variations of contracts possible, but with clear limits. Any variation should be forbidden if one of these cases occur:

1. if the variation causes a modification such that a competitor could had won the competition, or if other competitors could had participated to the selection process;
2. if the economic equilibrium of the contract changes significantly;
3. if the object of the contract is heavily extended and/or modified.

This applies to the framework agreement and to any amendment of it. Fine tuning of second level contracts would not mean any variation. Our solution simply consists in limited specifications concerning each iteration. Of course, variations may be possible in the case that the system's scope of the iteration is consistently different with the first level contract and can not match the general terms. For such an event,

the general limits to variations should be respected. Although this legal tool is an element of flexibility, a parsimonious use of it is recommended. In fact, a frequent use of variation may harm the general contractual governance. This would introduce opacity in the relationship which should avoided to enhance information symmetry between the parties.

6.4.6 Verification

Finally, also the verification needs to fulfill some legal requirements. Once built, or even during its development, the software should be inspected to see if it fulfills the software's scope. Such inspection should be accountable and the techniques defined upfront.

This complies very well with the Agile philosophy. Since the verification process is transparent, interests alignment is facilitated. The implementation of non-invasive tools is considered an effective way to enhance accountability along all the development process.

6.5 Setting up the contracts

In this section of the chapter we are going to sketch two possible implementation of our proposal, based on the contracting of Function Points and Scrum Sprints. These two examples should be embedded in the first level contract to determine the terms of compensation.

A key point for the contractual governance is the agility of the assessment of the economic value. Too complex metrics are not suited for such an environment, since it clashes with the *need for speed* of Agile iterations. Ideally, any assessment metric which assure fast and reliable (i.e., objective) outcomes is suitable for this purposes. Accordingly, we propose here two valuable but different examples. Functional size measurement methods focus on the code, while Scrum Sprints on the process. Clearly, both have upsides and downsizes briefly discussed. Although there may not be one best solution, rather several Pareto–optimal ones, it is of pivotal importance to understand the own contractual environment under which the software system is developed to use one of the proposed solutions or even new ones.

6.5.1 Contracts with Function Points

Function Point Analysis (FPA) [16, 450] provides enough objectivity in the evaluation process, independently from the used technology. This is the reason why FPA is a suitable option to guarantee the proper flexibility of the Agile methodology within the Italian constitutional framework discussed before. For the sake of simplification, also novel estimation techniques based on FPA, like Simple Function Points (SiFP) [158, 281], may represent a suitable and easy measurable metric, as already discussed in Section 6.3.

The definition process of SiFP - a quite straightforward functional size measurement method - is represented in Figure 6.3. The idea behind SiFP is to provide a good evaluation of the functional size of an application, without redundant basic functional components (BFC) i.e., DET - Data Element Type, FTR - File Type Referenced, and RET - Record Element Type. Therefore, only two BFCs are needed, according to this method:

- Unspecified Generic Data Group (UGDG)

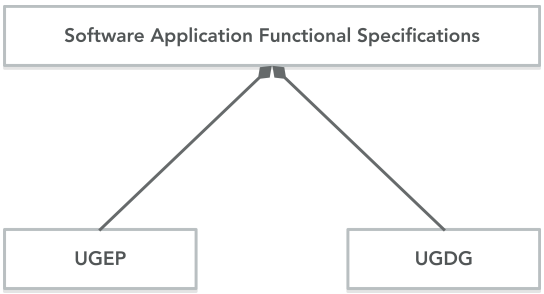


FIGURE 6.3: SiFP definition structure

- Unspecified Generic Elementary Process (UGEP)

Consequently, BFCs are of only two types: data object type, and transactional object type. Hence, the size of any software application may be expressed as the number of SiFP:

$$SiFP = M_{UGDG} + M_{UGEP} \tag{6.1}$$

Where M_{UGDG} and M_{UGEP} indicate the measures of the size of BFCs, as a result of IFPUG measurement rules. Interestingly, FPA measurements take into consideration e.g., new development, functional enhancement, and software assets, which SiFP does not. Still, SiFP are convertible to traditional FPA (like IFPUG), due to the high correlation through a regression analysis [5]. Therefore, from a practical perspective SiFP are convertible to any FPA, since functional size measurement methods are strongly correlated from a structural point of view.

Moreover, besides the simplification of BFCs, measurement weights are fixed, with respect to IFPUG. In particular, [281] propose that:

$$SiFP = 4.6UGDG + 7UGEP \tag{6.2}$$

Where SiFP are 4.6 times the number of elementary processes, without considering the primary intent, and 7 times the number of logical data files, without considering if they are internal or external.

Another strong point in favor of Function Points is that these are known and already used within the Italian public sector[3]. This means that it would be quite effective to write an Agile contract, based on the already acquired experience. FPA provides the right *tension* between interests in order to let align them, since it is an accountable process. Moreover, a *bonus-malus* effect would also help towards this direction. This mechanism should induce the provider to deliver not just average quality functionalities but high-value ones. We remark that although the delivered functionality can be first estimated and then assessed by FPA, there is limited guarantee for quality. In fact, FPA does not assess quality itself but only if the software computes a certain number of functionalities. Exceptional delivered quality has to be economically recognized, beyond the delivered functionalities. Similarly, also low quality should be discouraged. For this reason the use of a non-invasive quality tool to assess ongoing quality of software products is of greatest importance. It does not represent a legal issue, since the customer can easily include this methodological requirement in the competition call. Such a tool may compute not only the number of developed

[3]See for instance http://www1.interno.gov.it/mininterno/export/sites/default/it/assets/files/22/0011_disciplinare_di_gara.pdf retrieved on Aug. 23, 2017

functionalities but also judge their quality, according to industrial benchmarks (i.e., ISO/IEC 25010:2011). An example of such a tool is SonarQube **[CamPap13]**.

So, also the development methodology becomes of importance, since it is complementary to the non-invasive tool. Furthermore, the Test Driven Development (TDD) method [42] provides a useful approach to develop mission critical software with the highest attention to quality and security. For this reason we now sum up the three keystones of an Agile contract with FPA. In our proposal Law & Economics aspects of contracts are maximized, upholding constitutional duties of the contracting authority.

1. Specification costs are minimized by the methodology. After several iterations fine-granular functionalities are negotiated.

2. Dynamic inconsistency is is mitigated by a *bonus-malus* mechanism.

3. Non verifiable actions are mitigated by a Test Driven Development and the implementation of non invasive metrics.

These are the main characteristics for a transparent relationship which maximize the contract utility.

6.5.2 Contracts with Scrum Sprints

Another suitable way to write Agile contracts for the public administration are Sprint-based ones. In this case, Scrum Sprints are the base for terms of compensation. So, as in the other case, functionalities are described at a high level in the object of the contract but the economic value is not determined by the FPA but by the development iterations. It is a sort of body rental contract, where man-hours are organized in Sprints. Thus, for a team with 5 people for an iteration of 5 weeks (considering a 40 hours week), each Sprint will account for 5000 hour/person. The requirements refinement (through User Stories and continuous iterations), is part of the contract life. Both parties should be aware of the methodology, not only to avoid misunderstanding but also to prevent miscalculation of the effort. The aim is to build a win-win relationship, where parties are aligned to the goal and are treated fairly. A win-lose solution would be rather suboptimal, since there is no guarantee for a long-term engagement.

1. Sprint definition has to be clear in terms of temporal duration and people committed. In such contracts people play the greatest role. The level of expertise, seniority, and skill should be taken into consideration while designing Scrum teams.

2. The chosen Agile method has to be clear to both customer and provider to organize and setup the development. User Stories estimation is a sensible issue here. An overestimation, but also a underestimation may lead to misinterpretations between the parties as also frustration.

3. The *bonus-malus* mechanism described in the previous section should be clearly stated.

4. The use of monitoring and non-invasive tools is also an important issue to both interests alignment of all parties and improving accountability, as explained in the last section.

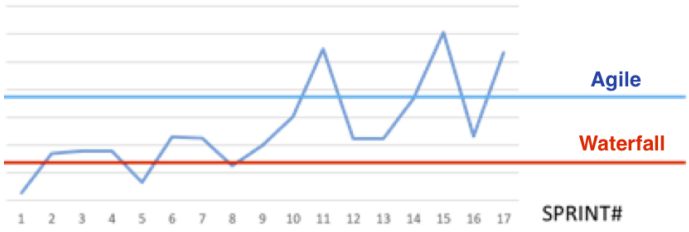


FIGURE 6.4: Effectiveness of the contract structure

6.6 Case Study

The ideas exposed in this chapter have been elaborated and evaluated in a contract with the Italian Defense Acquisition Process, partially described in [**MFRCR16**]. For this project we did not use the proposed contractual schema, although some principles were implemented (e.g., the *bonus-malus* mechanism).

The reference project has been the LC2Evo technology demonstrator. The project lasted about 87 weeks of work, that is, about two years. Sprints lasted 4 or 5 weeks each. Overall, the cost of the project was about 2.6 Million Euro. In this case instead of Function Points, Scrum Sprints have been used to define the contract. After each Sprint, the Definition of Done and acceptance criteria were assessed by external Army engineers. Their assessment were also useful for the subsequent Sprint planning.

A generic +50% increment factor was applied to take into account the better effectiveness and the reduced risk associated to the LC2Evo improved Agile development environment, with respect to the previous Waterfall development cycle.At the end of the project this +50% increment factor has proven to be pessimistic, since the effectiveness of the improved Agile development cycle was much better than expected. Moreover, due to an important cost reduction, leftover funds where used for extra hardware improvements.

Using a time and material estimation for a predetermined number of iterations (six to eight as we refer to LC2Evo) may be seen as a sort of "body rental" contract, where man-hours and materials are allocated in Sprints. The requirements refinement (through User Stories evolution and continuous iterations) was included in the contract. Both parties were trained and fully aware of the Scrum method, to avoid misunderstanding and prevent miscalculation of the effort. Figure 6.4 summaries the results that have been obtained using this approach. Unfortunately, specific details are classified for national security reasons. However, the resulting proportion provides a fairly accurate visualization of what happened. After an initial phase, the productivity increase of the new Agile approach was quite evident. Still, the merit of such productivity improvement can not be merely attributed to the different contracting structure, since the project followed a more traditional Arms & Materials contract. However, organizing the contract in an Agile fashion was a clear prerequisite for the project's success.

Finally, this case study gave us the relevant experience to design the new Agile contractual scheme, suitable for the Italian public administrations.

6.7 Conclusions

This chapter is an attempt to carry out a foundational work about Agile contracts. Starting from the Law & Economics of contracts, we explained how relevant principles

should be wider understand by the Software Engineering community. We pointed out how, through the alignment of interests, reduction of asymmetry and flexibility, Agile could be wider use in today's Software Engineering environment, especially within the Public Sector. Indeed, the awareness of software engineers about the economic motivations behind a contractual relationship is of pivotal importance to enhance software quality. Information symmetry is of greatest importance for Agile software development, due to structural reasons related to the day-by-day relationship with the customer and the short development–deployment iterations.

Companies which strive for competitive advantages can easily customize Agile contracts according to the contractual freedom principle (i.e., art. 1322 cc). This is not the case for public administrations which need to follow strict constitutional duties regarding public procurements. However, also these organizations (especially in the defence & security domain) need their software systems to evolve rapidly, to address new-world scenarios. Therefore, this chapter provides the first discussion about Agile contracting for the Public Sector.

Doing so, we highlighted the keystone for Agile contracting within the Italian public administration. These recommendations have a direct impact on all civil law countries, since they face similar procurement law principles. To contextualize better our proposal, we used a case study where Agile contracting principles has been implemented. The outcomes of the presented project are positive and significant.

Future work will proceed in two main directions. Firstly, both foundational as empirical studies about the implications and implementation of Agile contracts will be carried out. Accordingly, a detailed framework for an Agile contract for the Italian public administration will be presented in the near future. Secondly, the empirical validation of such contracts needs to be further studied. In particular elements related to the assessment of non-invasive measurements, effort estimation based on Sprints or SiFP, as well as social aspects of the negotiation should be considered for further studies.

Chapter 7

Legal Implications of Software Reuse

7.1 Introduction

A *software clone* is a fragment of source or executable code, that is copied in the same program or in a different one, whereas the act of copy is called *software cloning* [382].

Software cloning is a form of software reuse; in fact clones are identical or similar pieces of code, designs, or other artifacts exploited during the development of a software system. The copy-paste of someone else's code fragments into a different author's software program is a widely used programming practice, ranging from 5%-15% of the code base [397] up to 50% [386]. On average, the reuse of other people's code in large software programs is estimated around 20%-30% of code [29].

There are several ways of reusing code that are more formal (such as software components [183], web services [376], etc.) in which licensing problems are addressed explicitly (e.g., in open source software [363]). However, developers sometimes prefer different and more informal approaches [384]. There are many reasons why programmers copy software fragments and these reasons are largely studied in technical literature [245]. Several authors have already explored a model that studies the interaction and tracking of software licenses. For example, [172] developed a model which describes the interconnection of components from a legal point of view, using document integration patterns that are commonly used to solve the license mismatch problem in practice. For Open Source licenses, [359] proposes an approach to automatically track changes occurring in the licensing terms of a system. However, those reasons are not the focus of this study; what we want to address here are not the technical advantages or disadvantages of cloning software but the behavior of courts. In fact, this work is a study on the main rulings of software cloning from a Software Engineering perspective, following an approach started in [49], where the focus was on how software patents can influence software designers.

The issue of reuse by cloning is widely studied in Software Engineering, for instance, [382] lists hundreds of papers; however, in some situations cloning is considered unlawful. In fact, since in several countries, and especially in Europe, software is protected by the copyright law, software cloning is a form of plagiarism. We found this topic particularly relevant from a society's perspective, since this aspect of Software Engineering has wide cross effects, well beyond the technical dimension.

However, the definition of plagiarism for software is controversial. For instance, software clones are known to be closely related to various issues in the design of software for games, especially with respect to originality and creativity, qualities that have to be evaluated when an investigation of plagiarism takes place. For instance, in some competitions for software designers, notably in the World Computer Chess

Championship, there is an “originality" rule, which requires that all competing programs must either be original or quote other programmers whose work was used. Such a rule has been invoked a number of times, accusing some author of cheating by plagiarizing code to create a program [103]. These discussions about plagiarism are even more intriguing in the case of open source software [205, 388]. “If to plagiarize is to borrow *too much* code, then one needs to decide exactly how much is too much” [103].

Deciding about plagiarism is difficult. Trying to demonstrate that a program has been copied is not simple, for instance there are clones that reproduce only the functionality of a program, while the source code is different.

We did not find in literature a similar research dealing with court's perspective. We are only aware of a similar paper published in 1996 which outlines some legal implication regarding software reuse in general within the European Union [442]. The main contribution of this work is to survey the case law of these issues, as the court's point of view.

With this chapter we want to offer an insight for researchers and practitioners to understand the ‘way of thinking’ of US and EU courts when dealing with software cloning and, more generally, to IPR issues.

The main considerations are summarized after each section. The structure of this chapter is as follows. In Section 7.2 a brief explanation of the different clone types is given. To understand the main reference points of courts, in Section 7.3 a brief overview of the main laws are depicted. In Section 7.4 we carried out all relevant US and European Court of Justice case laws. A manual analysis of both case law was performed. Moreover, an analysis of the European Court of Justice was carried out, to compare both jurisprudential leanings of the courts. We found out that one ruling has a particularly disruptive nature, thus, in Section 7.5 we discuss it since we believe it will have a huge impact on software copyright in general. In Section 7.6 we discuss some of the major implication of this chapter. Eventually, we conclude and outline some future research in Section 7.7.

7.2 Background: Types of software clones

Software clones are not just copy-paste fragments of different codes. Rattan *et al.* [382] identify four types of clones, summarized in Table 7.1.

Assessing the lawfulness of the reuse of program fragments is not straightforward. Clone detection tools, even the most reliable ones, are based on heuristic methods, i.e., they make probabilistic judgments, so it is legally impossible to use them to state the existence of cloning, *beyond reasonable doubt*. Since the provision of the reasonable doubt is a fundamental human right, courts have some obvious difficulties in assessing legal responsibilities when a software contains clones of type 4 (functional clones) but also of type 3 (near miss clones).

Since cloning is a complex issue, *a priori* it is neither a good, nor a bad habit to clone software fragments. In Table 7.2 we summarize some of the main remarks in literature about cloning. This is relevant for the analysis of the case law as it shows the level of maturity of the debate from a Software Engineering perspective.

7.3 Short comparison of two legal frameworks

Courts do not take subjective decisions, they interpret the law. Here, we briefly describe the legal frameworks of the US and the EU regarding software's IPRs. Both US

TABLE 7.1: Clone types in Rattan *et al.* (2013)

Type	Main characteristics
1–Exact Clones	Program fragments which are identical, with the only exception of white lines and comments
2–Parameterized Clones	Program fragments structurally or syntactically similar, with exception for identifiers, literals, types, and layouts
3–Near Miss Clones	Program fragments copied with some modifications in the source code, i.e statements insertions/deletions, besides identifiers, literals, types, and layouts
4–Semantic Clones	Functionally similar program fragments, which are not formally identical

TABLE 7.2: Advantages and disadvantages of code cloning

Advantages	Disadvantages
Clones are useful if different customers share similar requirements [256]	High maintenance costs [318]
Some programming languages encourage the use of templates, which result in software cloning [256]	Propagation of bugs: if a clone contains an error, it will spread rapidly over other parts of the program [232]
The use of clones can respond, sometimes, to efficiency requirements in the development [244]	Cloning discourages the use of refactoring, leading to a bad design of the system [282]
Using clones reduces the time required to develop a program [244] [245]	Using clones increases the size of the code, leading to a less efficient system [262]

and EU copyright laws very briefly described, to give a short overview and references about courts' starting points.

7.3.1 IPRs in the US

The US law system is based on "Common Law": this means that previous judgments are binding. Therefore, there are several past cases that are relevant for a court to issue its judgment. Nevertheless, there is a structured codex, where all courts shall take reference to. For copyright cases it is the Title 17 of 1976 and its amendments. The last one is the Reauthorization Act of 2014. Title 17 is composed by 13 chapters and 14 appendixes. Each chapter regards a specific issue of copyright, as described in the following Table 7.3.

Table 7.3: Chapters of Title 17 (Copyright Act)

Chapter	Subject
Chapter 1	Subject Matter and Scope of Copyright
Chapter 2	Copyright Ownership and Transfer
Chapter 3	Duration of Copyright
Chapter 4	Copyright Notice, Deposit, and Registration
Chapter 5	Copyright Infringement and Remedies
Chapter 6	Importation and Exportation
Chapter 7	Copyright Office
Chapter 8	Proceedings by Copyright Royalty Judges
Chapter 9	Protection of Semiconductor Chip Products
Chapter 10	Digital Audio Recording Devices and Media
Chapter 11	Sound Recordings and Music Videos
Chapter 12	Copyright Protection and Management Systems
Chapter 13	Protection of Original Designs

Table 7.4: Appendixes of Title 17 (Copyright Act)

Appendix	Subject
Appendix A	The Copyright Act of 1976
Appendix B	The Digital Millennium Copyright Act of 1998
Appendix C	The Copyright Royalty and Distribution Reform Act of 2004
Appendix D	The Satellite Home Viewer Extension and Reauthorization Act of 2004
Appendix E	The Intellectual Property Protection and Courts Amendments Act of 2004
Appendix F	The Prioritizing Resources and Organization for Intellectual Property Act of 2008
Appendix G	The Satellite Television Extension and Localism Act of 2010
Appendix H	Title 18 - Crimes and Criminal Procedure, U. S. Code
Appendix I	Title 28 - Judiciary and Judicial Procedure, U. S. Code
Appendix J	Title 44 - Public Printing and Documents, U. S. Code
Appendix K	The Berne Convention Implementation Act of 1988
Appendix L	The Uruguay Round Agreements Act of 1994
Appendix M	GATT Trade-Related Aspects of Intellectual Property Rights (TRIPs) Agreement, Part II
Appendix N	Definition of Berne Convention Work

Appendixes concern integrations with international standards and agreements and amendments of the Copyright Act of 1976. They are composed as detailed in Table 7.4.

Courts may take into consideration other laws. For instance, anti-monopoly provisions may apply for cases where functional cloning is permitted, to restore market

competition. Moreover anti-fraud laws could be claimed in cases where hardware-software is cloned.

In short, even though software cloning is most closely related to copyright infringements, it is not exclusive. Also other laws and provisions could be used by courts to restore justice. Hence, the issue of cloning is more complex and broader than just copyright infringement.

7.3.2 IPRs in the EU

Similarly, the European Court of Justice (ECJ) has to follow the EU law. Moreover,the ECJ acts like a Supreme Court of all EU Member States. The particularity is that each Member State has its own national law, which is different from the others. However, each ECJ judgment has to be considered binding for each Member State's court. We can consider the EU law system as a mixed civil law and common law system. The most common EU laws regarding copyright are represented in Table 7.5.

Table 7.5: EU Directives regarding Copyright

Directive	**Subject**
Council Directive 93/83/EEC	Copyright and related rights: satellite broadcasting and cable retransmission
Directive 98/84/EC	Protecting electronic pay services against piracy
Directive 96/9/EC	Legal protection: databases
Directive 2001/29/EC	Copyright and related rights in the information society
Directive 2001/84/EC	Resale right for the benefit of the author of an original work of art
Directive 2006/116/EC	Copyright and related rights: term of protection
Directive 2006/115/EC	Rental, lending and certain other rights related to copyright in the field of intellectual property
Directive 2009/24/EC	Legal protection: computer programs
Directive 2012/28/EU	Wider access to copyright material — Orphan works

7.3.3 Consequences for software cloning

The two legal systems deal differently with software cloning, as summarized in Table 7.6. For our purposes, the differences mean that the concept of "*software cloning*" in the US is quite mature compared to the European one. Since US courts are bound to a common law legal system, their judgments reflect in some sense the *Zeitgeist*, namely the spirit of the time. Such judgments are based on both previous court's decisions and on the judge's interpretation of the case. Therefore, common law judges are much more exposed to the spirit of the time than the EU ones. In fact, several judgments refer directly to such concept, so we can find easily such judgments in the legal databases. Instead, the ECJ is much younger than the US legal system. Since the EU legal system is based on a civil law system, all software cloning judgments

Table 7.6: Main differences between the US and EU legal system concerning this chapter

US	EU
Common Law (Judicial decisions are binding – decisions of the highest court can generally only be overturned by that same court or through legislation)	Civil Law (Only legislative enactments are considered binding for all. However, ECJ ruling are considered to be binding in all Member States of the Union)
Long lasting judicial tradition	Recent establishment (1952)
Extensive freedom of contract, courts are more sensitive to the Zeitgeist. Judges may interpret the undergoing issue according to 'language' of the counterparties to reestablish the rule of law.	Less freedom of contract and reference to the law. Courts may reestablish the rule of law according to the legislative interpretation on which counterparties are bound

have to be explicitly related to the IPR acts. Unfortunately, none EU law treats "software cloning" as such. This means that in the Eur-lex database used to perform our investigation we expect to find most judgments about software cloning under the keywords "*software & copyright*".

7.4 The case law

7.4.1 Research Protocol

Adapting a Systematic Literature Review (SLR) for law cases has strong limitations, since case law is not comparable to literature articles. Moreover, articles can be found in the editor's database (e.g., Springer, Elsevier, IEEE, ACM); whereas rulings do not have editors. Rulings are written by judges, after that employees of commercial databases (e.g., LexisNexis) put them on line. Therefore, some rulings are not into any database. Especially rulings issued by lower courts are hardly collected, since they are considered of low relevance.

Moreover, search criteria for a SLR are different from a case law research. In SLR keywords used represents the scientific topic the article addresses. Case law induce a technical keywords, but from the law domain, not Software Engineering ones. If, with a SLR we have a direct technical feedback about the chapter's topic, searching within case law it is always a question of the right interpretation, judgment per judgment.

The arbitrary dimension in the analysis of case law is ineluctable. Nevertheless, we elaborate a design for a case law research protocol to systematize our research [102]. To comply as much as possible with the empirical software analysis tradition we modeled our design according to the framework proposed by [400]. We elaborated and followed this protocol:

1. identification of the country/legal system where to carry out the analysis;
2. definition of an appropriate database for the case law;
3. definition of an appropriate query, according to the legal system;
4. manual identification of the relevant cases:
   (a) exclusion criteria: not relevance to the topic;
   (b) inclusion criteria: relevance to the topic;

5. manual analysis of the relevant cases.

According to our research protocol, we choose to analyze and compare the US and the EU legal system's leaning to the cloning issue.

The database identified for the US is LexisNexis, which is one of the most complete and reliable law databases for the US case law[1]. For the EU we used Eur-Lex[2]. Eur-Lex has one big advantage, it is the official Law database of the European Court of Justice. So, any judgment of the ECJ is available in the Eur-Lex database.

The query chosen for for the US was "software & cloning" in the Academic Search for State and Federal Cases of all available judgments of all US courts. The outcome were 85 cases. The identified query for the EU was "software & copyright" in the textual research and the output were 27 cases. Both queries were carried out on July 29th, 2015. As explained before in Subsection 7.3.3 the two legal systems are different. To respect these diversities we had to adapt our queries, according to our protocol. No EU law refers directly to "software cloning", but to "software copyright". Similarly, the US adopt better the *Zeitgeist* and are more confident to speak about software cloning as such.

Both outcomes of 85 cases for the US and 27 for the EU were reproduced in two tables, one for the US[3] and one for the EU[4] with all relevant identifiers. After that, case by case were manually analyzed.

In total, the excluded cases were 51 for the US and 23 for the EU. The inclusion, or exclusion of cases were carried out by the authors on a qualitative basis. However, also the excluded cases were clearly mentioned within the tables, explaining why they were exclude. The motivation of exclusion was the non relevance of the subject matter. The remaining relevant cases were then chosen for the manual analysis.

Relevant cases were analyzed one by one. In total, analyzed cases were 34 for the US and 4 for the EU.

7.4.2 First considerations about the outcome

The chosen queries were kept as general as possible to catch the highest number of relevant cloning cases. As a consequence,the relevant cases are a minority of the dataset.

However, we are aware of at least 5 more cases, which were not captured by our query. We found them by serendipity, while studying the topic. For completeness we analyzed also these cases.

Moreover, we took just these two systems, because we are most aware of, also due to the language. Definitively it would have been interesting to study also other legal system regarding cloning issues. Unfortunately, there is a huge language barrier that we have to overcome.

We remark that we only considered case law at EU level, since each Member State has its own case law in its own language.

7.4.3 The United States

Claims regarding software cloning are rare, probably because they are very difficult to prove. We analyzed all 85 output cases on LexisNexis about "software & cloning" and

[1] www.lexisnexis.com

[2] www.eur-lex.europa.eu

[3] For the US Case Law: www.cs.unibo.it/~cianca/wwwpages/dataxpapers/US_Case_Law_LexisNexis_After_1944_software_cloning.pdf

[4] For the EU Case Law: www.cs.unibo.it/~cianca/wwwpages/dataxpapers/EU_Case_Law_EUR-LEX_software_copyright.pdf

TABLE 7.7: Number of cases per cloning area

Type of Software Cloning	**# Cases**
A: Software and Hardware cloning issues related to physical devices	13
B: Software cloning issues related to competition and antitrust issues	7
C: Software cloning issues related to misappropriation of trade secrets and copyright infringements	14
NA: Not Applicable. This is not a case related to software cloning	51

created the table US Case Law. Interestingly, as shown by our table, the US case law regarding cloning issues in software is wavering and considers mainly copyright issues. There is no single judgment of the Supreme Court about the issue. This means that there is no unequivocal interpretation about cloning, but courts judge, case by case, according to the specific issue (our A, B or C cluster).

We found no case about software clones of types 1, 2, and 3; instead the US courts dealt with functional clones (type 4), as shown in the table. Courts tend to judge over general issues regarding copyright, and do not enter technically in the cloning issue.

Interestingly, courts seems to use common pattern. Therefore we cluster the cases according to these patterns. It was quite surprising to discover that, with regard to software cloning, US courts lead their judgment within one of three cluster. This is probably due the fact that sticking to the law, the type of evaluation needs a certain level of homogeneity. The choice of putting one case in a cluster were made by the authors, according to the relevance. Since any case is describe, the relevance to the cluster may be confirmed also by the reader.

So, to analyze better the Case Law, we clustered the cases into three areas in Table 7.7. In the next subsections a general description of each cluster will perused, to motivate our findings.

According to our research, in the US, we found no relevant case regarding software cloning of types 1, 2, and 3. We do not know if no one has ever claimed a court about the resolution of a cloning issue or courts do not judge over these issues due to their technical complexity. Probably, according to the plaintiff's strategy, it is better to appeal the court for some more evident issues (like a similar GUI) rather than a piece of code. Courts want to deal, apparently with what they see, like Graphical User Interfaces, rather than discuss about the likelihood of a clone detection match. The only case law we found regards type 4 clones (indeed, with a negative outcome for the plaintiff). Apparently, courts tend to consider the broader picture of a program's features. Even though there are, at the state of the art, excellent detection techniques[5] Courts do not judge over heuristic techniques, even though they are considered highly reliable, thus preferring to judge over generic issues, like graphical interfaces.

This seems contradictory. If semantic clones are the most difficult ones to identify, why is all the available case law about these clones? Again, a reasonable explanation is that for a court it is easier to judge about the output of a code (like a GUI, or general functionalities) rather that deciding on which degree a code has been copied.

[5]For a comprehensive and recent survey, see [382].

In the next three subsections we deepen our case cluster. The case law is represented in the table US Case Law. Please consider that we put together court's statements regarding cloning issues.

In Subsection 7.4.4 we analyzed cloning issues related to physical devices which may not be strictly related to software cloning. However, we found it interesting, because it shows how courts deal with cloning of technological issues. This gives us some insights about court's behavior. In Subsection 7.4.5 courts deal with the well known issue of antitrust and monopoly in the technology market. Here, judges weight the interests of free market competition and individual propriety rights. Subsection 7.4.6 is the more straightforward case of software cloning and copyright infringements.

7.4.4 Software and hardware cloning related to physical devices

Interestingly, cases belonging to the first cluster A (Software and Hardware cloning issues related to physical devices) are all concentrated before 2000, so among the first ones. Between 1990 and 2000 almost all cases were about software and hardware cloning of phone and television devices. Out of 18 relevant cloning cases in the last decade of XX century, 12 were related to such issues[6] Only in 2012, in United States v. Harris, a US court treated a cloned device issue.

However, this case is related to a cloned software and hardware tool that enables a free and faster Internet access.

In United States v. Davis of 1992, Davis was convicted of violating various federal statutes and copyright infringement regarding cable television and its satellite-signal system. Once completed, Davis's modifications made it possible for the cloned modules to descramble and decrypt satellite programming without the knowledge of the cable companies. The modifications also made it all but impossible to use the device in any legitimate fashion.

Similarly, in United States v. Yates of 1995, the court held that cloning involved reprogramming a cellular telephone so that its electronic serial number and mobile identification number combination was identical to a legitimate customer's account. The court ruled that the defendant violated the law because cloning involved the use of an altered telecommunications instrument to obtain access to pay services for the purpose of defrauding the carrier.

In these cases, the outcome for the defendant was rather negative. Courts defined these cases within other legal domains (e.g., telecommunication law), with crimes clearly defined by the law.

7.4.5 Software cloning related to competition and antitrust issues

Case Law of the B cluster (Software cloning issues related to competition and antitrust issues) started at the very beginning of the XXI century. These cases are interesting because they show how courts (re)act to monopolistic behaviors. Usually, functional cloning is permitted (also encouraged) to create competition.

The most relevant cases involve Microsoft Corporation. Probably, the most interesting case is the first one: the 1999 United States v. Microsoft Corp. The US government claimed that Microsoft violated antitrust provisions of the Sherman Act, which is the most relevant antitrust law. The court concluded that Microsoft had monopoly power in the strategic market of PCs because the defendant could substantially raise its prices without losing business to a commercially viable alternative, since Microsoft's market share was large and stable, and the related market was protected by

[6]For a comprehensive overview please cfr. the table US Case Law.

a high barrier of entry. The court further found that Microsoft purposefully leveraged its monopoly power in the market to thwart competition in other software markets. Specifically, through restrictive OS licensing agreements with computer manufacturers, Microsoft achieved a higher market share in the web browser market. Microsoft protected its monopoly and hindered innovation by imposing barriers to entry against various cross-platform software, “middleware", and network applications.

Here, cloning does not have a major importance, since the court intended to protect market competition. In fact, in cluster B cases, usually courts tend to see cloning as positive, since it creates competition, which is more important than property rights (e.g., copyright) [301]. Antitrust is one of the most studied subjects in Law and Economics literature [223]. Courts tend to restore equilibrium in the market, which is a common good.

Therefore, any possible act of competition is encouraged (directly on indirectly), even functional cloning.

Microsoft was sued by several other public bodies, for very similar accuses, like New York v. Microsoft Corp. in 2002 or Massachusetts v. Microsoft Corp. in 2004. Also common in cluster B are the cases of private corporations sueing other corporations for unfair competition. In Sun Microsystems, Inc. v. Microsoft Corp. in 2000, the court granted a preliminary injunction, finding that Sun had a reasonable chance of success on the merits, the hardship to Sun of Microsoft's continuing its potentially unfair competition outweighed the burden on Microsoft, and Microsoft was likely to continue harming Sun if the injunction were not granted.

7.4.6 Software cloning related to misappropriation of trade secrets and copyright infringements

The last cluster - cluster C - of rulings related to misappropriation of trade secrets and copyright infringements is, probably, the most connected to software issues. It includes cases from 1990 to 2014.

We found no clear trend followed by the courts, because they protect the plaintiff or the defendant, case by case. More precisely, we cannot clearly state that courts tend to protect inventors from software cloning.

The oldest case in this cluster is Lotus Development Corporation v. Borland International, Inc. of 1990. The court dealt with generic issues, i.e., whether the copyright does extend to the text or layout of a program's menus, stating that it is not extendable. The court held that if the expression of an idea had elements that went beyond all functional elements of the idea itself, and beyond the obvious, and if there were numerous other ways of expressing the non-copyrightable idea, then those elements of expression, if original and substantial, were copyrightable.

Also the FASA Corp. v. Playmates Toys case of 1994 is related to the cloning of playing symbols. The court indicated that a comparison of the game materials revealed that there were marked similarities between the two sets of playing symbols programmed in the games, but no substantial evidence of copyright infringement.

On the contrary, in United States v. Manzer, the jury determined that computer programs sold by Manzer were derivative of copyrighted material, and that the software contained sufficient notice of its protected status. Likewise, Tradescape.com v. Shivaram of 1999. Tradescape sued Shivaram for copyright infringement and theft of trade secrets concerning online day trading computer software. Shivaram, a software consultant that used to work for Tradescape, developed a software program that allowed for online day trading. Tradescape established a likelihood of success on the

merits on its copyright infringement and theft of trade secrets claims because it provided sufficient direct and circumstantial evidence of copying of protected material.

Another important case is Oracle Am., Inc. v. Google Inc. of 2012. Oracle wrote 37 packages of Java source code, published their "application programming interfaces" (API), and licensed them to others for writing "apps" for computers, tablets, smartphones, and other devices. Oracle alleged that Google's Android mobile operating system infringed Oracle's patents and copyrights. The jury found no patent infringement, but ruled that Google infringed copyrights in the 37 Java packages and a specific routine, "rangeCheck". Hence, copyright protection extends to all elements of an original work of computer software, including a system or method of operation, that an author could have written in more than one way.

The 14 cases in cluster C are interesting because there is not a clear positive or negative jurisprudential trend followed by the courts. Evaluating case by case, courts tend to value positively or negatively cloning issues. Usually, there are considered much more elements than just a simple "copied" or "non copied" answer. There is always a human, arbitrary element in any judgment. Thus, any kind of reasonable expectations of an outcome of similar cases are rather difficult.

7.4.7 Other cases related to the United States

In this chapter we study the behavior of courts when they have to deal with software cloning. However, some cases of software cloning are not classified "software & cloning" inside legal databases. Therefore, analyzing all cases of software cloning, strictly speaking, is nearly impossible, since we should analyze any law case where software is involved. Moreover, even after the analysis of such cases, still you have not the guarantee that you took all, since not all cases are imported in such databases.

By serendipity search we found five other relevant cases where software cloning is involved. These cases are listed and explained in Table 7.8. They are all related to copyright infringements or other infringements. Since they do not fall in the chosen query "software cloning", we kept it separately. These five cases, *de facto* confirm our previous analysis.

The use of graphical user interface (GUI) elements that are similar between two systems were examined Apple Computer, Inc. v. Microsoft Corporation and Hewlett-Packard Co.; the court stated that such a (re)use does not represent an infringement. These claims do not deal with clone type 1, 2, or 3 but only with functional ones.

We believe that US courts do not clearly protect a copyright holder. In fact they have a waving aptitude, from case to case, according, mainly, to jurisprudential issues. This is not the case of the European Court of Justice case law, discussed in the next section.

7.4.8 The European Union case law

Only four cases relate to software cloning. The others deal with jurisprudential issues of abuse of dominant position[7]. Other cases regards patent or copyright registration issues. Since the Office for Harmonisation in the Internal Market (OHIM) is a European agency, any cases regarding such issues is of competence of the ECJ. Moreover, we have some cases regarding data protection and public competition against the EU issues.

[7]Consider that the European Commission was established to create the Single Market. Actions against abuse of dominant position are among the most frequent ones, to foster the European competition [213].

Table 7.8: Other US case law.

Case	Court	Year	Relevant output
Step-Saver Data Systems, Inc. v. Wyse Technology	3rd Cir.	1991	Characterizing the transaction as a license to use software is a habit which do not correspond to today's world.
Computer Associates Int. Incv̇. Altai Inc.	2d Cir.	1992	to claim for a copyright infringements there has to be claimed substantial similarities.
Lewis Galoob Toys, Inc. v. Nintendo of America, Inc.	9th Cir.	1992	Any consumer can do changes to licensed computer games for personal use.
Apple Computer, Inc. v. Microsoft Corp.	9th Cir.	1994	Certain components of a GUI do not underly to copyright law.
Meshwerks, Inc. v. Toyota Motor Sales U.S.A., Inc., et al.	10th Cir.	2008	3D models are not protected by copyright even if they represent the original.

Regarding the case law, the ECJ has relevant rulings regarding software cloning just for type 4. So, easily detectable clones (of type 1, 2, or 3) are not relevant at for the EU court.

What distinguishes the EU from the US is the different approach to copyright. The ECJ fully recognizes the legal dignity of computer programs as copyrightable goods (UsedSoft GmbH v. Oracle International Corp.). However, at the same time, it gave a loose protection to functional cloning (Bezpecnostni softwarova asociace v Ministerstvo kultury.). Moreover, the ECJ went way beyond with the SAS Institute Inc. v World Programming Ltd. case, which has several disruptive elements regarding copyright law, which will be analyzed in the next section.

Furthermore, consider that these sentences are a Supreme Court pronouncement, thus binding for all Member State jurisdictions.

Both US and EU courts never judged about type 1, 2, or 3 clones, which are relatively easy to detect with state of the art heuristic tools. The available case law concerns only type 4 clones (semantic ones) and other, general, copyright issues. Synthetically, we can say that:

- courts have an apparent difficulty to deal with clone detection issues;
- we did not find one single law case about clones of type 1, 2, or 3;
- the only cases treated by courts concern type 4 (functional) clones;
- the few cases regarding functional clones have, within the EU, a 'loose' copyright protection.

7.5 An ECJ disruptive ruling

After the SAS Institute v. World Programming Ltd[8] ruling by the European Court of Justice the legal consideration of the EU justice system about copyright of software changed radically.

This ruling introduced diverse major consequences about how to interpret software, from a legal perspective. The implications for software programs and related lawsuits are of greatest relevance for the software community in Europe.

Going into details, the ECJ stated three important principles regarding the interpretation of Directives 91/250 and 2001/29:

1. The first, and most important part of this ruling is that the Court, stating that *the legal protection of computer programs is to be interpreted as meaning that the functionalities of a computer program and the programming language are not eligible, **as such**, for copyright protection. It will be for the national court to examine whether, in reproducing these functionalities in its computer program, the author of the program has reproduced a substantial part of the elements of the first program which are the expression of the author's own intellectual creation.* So, since software's features are considered as "principles" or "ideas," by the Court they are not copyrightable expressions by themselves. The copyright of software is so, no more considered an absolute assumption but a relative one. This does not mean that computer programs are not copyrightable. They are a form of expression of the intellectual creation of the programmer but the "principles" or "ideas," themselves are not protected by law.

2. Moreover, regarding reverse engineering for interoperability issues, the ECJ affirms that *it is not regarded as an act subject to authorization for a licensee to reproduce a code or to translate the form of the code of a data file format so as to be able to write, in his own computer program, a source code which reads and writes that file format, provided that that act is absolutely indispensable for the purposes of obtaining the information necessary to achieve interoperability between the elements of different programs. That act must not have the effect of enabling the licensee to recopy the code of the computer program in his own program, a question which will be for the national court to determine. [...] Acts of observing, studying or testing the functioning of a computer program which are performed in accordance with that provision must not have the effect of enabling the person having a right to use a copy of the program to access information which is protected by copyright, such as the source code or the object code.* The direct consequence of this statement is that any software engineer, who acquired a license of a software can freely observe, study or test it to fix interoperability or for education purposes. So, any software, which has been acquired legally, can be studied and the copyright holder is not able to prevent it. Interestingly, even though someone would study the program, to copy it, this could not be considered a copyright infringements.

3. The last paragraph of the ruling, which is less relevant from our perspective, is about the copyright of the user manual. According to the ECJ, *the reproduction, in a computer program or a user manual, of certain elements described in the manual for another computer program may constitute an infringement of the copyright in the latter manual if – a question which will be for the national court to determine – the elements reproduced in this way are the expression of their*

[8] C-406/10, 02.05.2012

> *author's own intellectual creation.* Also for this case, the expression, original creation of the author, is protected by copyright law. Not protected are *keywords, syntax, commands and combinations of commands, options, defaults and iterations* singularly, but *the choice, sequence and combination of such elements that the author may express his creativity in an original manner and achieve a result which is an intellectual creation.*

The disruptive nature of this ruling is quite clear. Even though before there were no real case law regarding cloning or copying issues, about copyright infringements, this sentence has a big impact in the computer science community because it states relevant issues that have a direct impact for programmers, at least in Europe. With the SAS judgment we can figure out the following direct consequences for software engineers:

- it is possible to reproduce "principles" or "ideas" of other people's software programs;
- it is possible to profit from others' "principles" or "ideas," since they are not protected by copyright;
- my own "principles" or "ideas" are not copyrightable, so everyone can get full inspiration from them;
- European courts are not the right place where to defend "principles" or "ideas" because no legal paradigm protects them;
- courts within the EU may intervene if the source or object code itself is copied;
- the source or object code of any program can be studied, without any permission of the licensor for "study" purposes. Therefore, if someone studies the source code of a program to get its "principles" or "ideas" to exploit them, no one can, *de facto* prevent it. Even though the court handles with the case which regard reverse engineering for interoperability issues, it is easy to bypass this case. In principle, any programmer could claim to have "studied" it for interoperability. So, even if nothing would come out (in terms of interoperability) still the programmer could have been studied the program, without any restriction. Finally, no one could claim copyright issues if he gets "principles" or "ideas" of that program, for his own program.
- it is legal (at least in Europe) to copy/clone "principles" or "ideas" of any program, also for profit.

7.6 Impact on software

This chapter shows how an interdisciplinary approach to software may bring enriching elements to the community discussion. In this case, bridging legal considerations within the clone literature brings some relevant insights about an everyday aspect of software: its legal protection. Here, we represented the behavior of US and EU courts when it comes to software cloning and, more generally, to IPR issues. In Table 7.9 we outlined the main differences between the two legal systems, with respect to software cloning.

For the US we found out that, at the moment, the Supreme Court has not issued a ruling regarding cloning issues in software. This means that first and second level

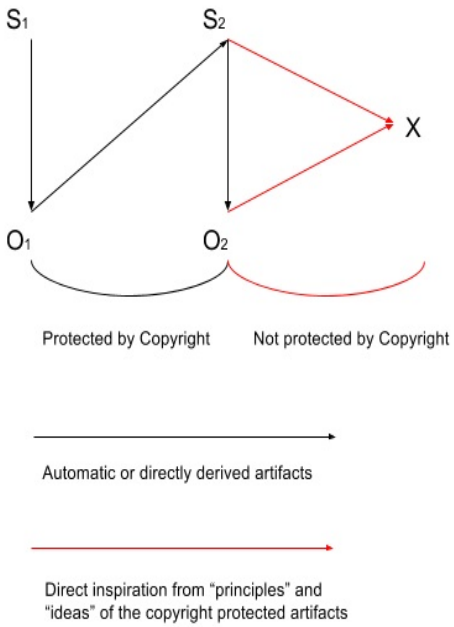


FIGURE 7.1: Copyright protected and not protected reengineering according to the ECJ.

courts will continue to use the pattern identified and discussed in Subsection 7.4.3. So, new cases of software cloning in the US will fall under cluster A, B or C.

For all EU Member States, the SAS Institute v. World Programming Ltd ruling will have a disruptive impact regarding reverse engineering. So, from academic, non profit or FOSS application, up to commercial, closed source and business applications, reengineering has to be considered, at least permitted. Anyone can use "principles" or "ideas" of another software artifact for the proper use. And no one can legally claim an IPR protection if someone else exploit the own original "principles" or "ideas" used in someone else artifact. If the software community in Europe wants to have a more stringent protection about the software, European courts (of all Member States) are not the suited place to get this protection.

One big issue is the architecture visualization of a system [166]. In literature, there are different visualization tools proposal, to cope with the complexity of computer programs [276]. But to what extend, from a legal point of view, a software engineer can apply these visualization tools to study and understand better the architecture of a program? Which kind of authorization has he to ask for? According to the ECJ the answer is simple: there is no limitation. Anyone can *de facto* study the architecture for any purposes. First of all for interoperability issues but also for education ones. This means, basically everything. Furthermore there is no need for any kind of permission to do that.

Furthermore, it is of highest interest interpreting in an operative way the SAS ruling. The ECJ refers explicitly to "principles" or "ideas". So, direct derived work is still to consider protected by copyright, since it is much more then not just "principles" or "ideas" . The typical example is the relationship of the source code and the object code, explained in Figure 1. The object code O_1 is automatically derived by the source code S_1, elaborated by a compiler. Therefore, also the object code is protected by copyright, as it is an automatic execution of a human creative activity, namely the source code.

Table 7.9: Copyright protection within the EU and the US

	EU	**US**
Difficulties to deal with cloning issues	✓	✓
Courts do not judge over cloning cases of type 1, 2 or 3	✓	✓
Clear clustering of case law	X	✓
Software is patentable	X	✓
The use of "principles" or "ideas" belonging to other people is permitted	✓	X
It is possible to profit from others "principles" or "ideas"	✓	X
Courts are the right place to defend your own "principles" or "ideas"	X	✓

Going ahead, what happens to an object code O_2 which is decompiled in a source code S_2, which is slightly different with regard to e.g., the identifiers? Is it still a copyright protected artifact? The answer is yes. As the automatic compilation, also the decompilation is an automatic and directed process to get to a derivative product S_2 from an original S_1 protected one.

But what happens is if the relationship is not automatic or direct? Well, the answer is it depends. However, we can reasonably state that if the change to the software exposed to reverse engineering is not trivial and the new artifact X follows just "principles" or "ideas" of S_1, it is for the ECJ a new artifact with no legal relation to S_1 or S_2.

Ultimately, the fine line between derived artifacts protected by copyright (S_2) or new one (X) inspired by the original one, depends on the "degree of derivation". If the derivation is verry loose, and the X artifact recalls just "principles" or "ideas" of S_1, the ECJ would not claim any copyright infringment. On the other hand, if this recall is more than just a simply "inspiration", well the ECJ would seen this as infringment.

Unfortunately, there is not a easy answer to this issue. Since we are dealing with courts, there will always be a degree of arbitrarity. Nonetheless, we found out that copyright protection of software within the EU is relatively loose.

For all EU Member States, the SAS Institute v. World Programming Ltd ruling will have a disruptive impact regarding software cloning issues. What we define as functional cloning or type 4 clones are perfectly admitted in all thinkable use. So, from academic, non profit or FOSS application, up to commercial, closed source and business applications, functional cloning has to be considered, at least permitted. The use of "principles" and "ideas" of software is free and may be used for the proper purposes. No one can legally claim an IPR protection if someone else exploit the own original "principles" or "ideas" used in someone else artifact.

What the community needs to understand very clearly is that all courts of EU's Member States are not the suited places for a stringent protection of software artifacts. Such a conclusion has a wide overall impact on society.

7.7 Conclusions

In this chapter we analyzed the difference in behavior of US and EU courts.

Courts, usually, do not enter in cloning issues concerning program fragments that are identical. The case law we found is all is about semantic clones, which only clone detection heuristics can figure out. However, courts seem not to rely on such

techniques, using, for their judgments, more general principles, like the use of software as method of operations.

The US courts have a waiving attitude and decide over software's copyright protection case by case. Furthermore, no Supreme Court ruling relevant for software cloning issue was found.

The approach of the European Court of Justice appears looser in terms of degree of legal protection of software's copyright. Undoubtedly, the SAS judgment has to be considered as disrupted in term of copyright protection. This has a wide relevance not only for the Software Engineering community but also for everyone dealing with intellectual propriety. According to the ECJ position, all Member States courts within the European Union, have to align their future rulings, since it is a pronouncement of a higher court. The Court stated that cloning of "principles" or "ideas" (semantic clones) can not be an infringement of copyright, since "principles" or "ideas" are not copyrightable.

Therefore, the question "how much is too much", at least for courts, makes little sense. Both, US district and federal courts and the ECJ disregards cloning cases, strictly speaking i.e., type 1, 2 or 3. The only relevant cases are functional cloning ones.

A further interdisciplinary research effort may investigate future rulings of lower court within the EU. So, we could analyze the level and type of interpretation given by these courts to the SAS ruling. What we want to state is that the right interpretation of interdisciplinary issues in Software Engineering, like the effect of courts into the protection of the propriety of software is crucial, since it has a big impact on the community on both copyright and commercialization aspects. In particular, it could be of interest to analyze how European software houses complies with this new leaning of courts. Moreover, a wider analysis of more legal systems (e.g., India, China) could give us better insights about court's behavior in cloning cases.

Chapter 8

Cooperative Thinking

8.1 Introduction

Literacy is a personal skill, needed by any citizen to interact with society. The individual scope of literacy has deeply influenced teaching methodologies and especially students' evaluations, concentrating most educational efforts on the individuals; see for instance [435] for a discussion about individual social differences in the acquisition of literacy.

Computational Thinking is a new form of literacy [486]. It is a concept that has enjoyed increasing popularity during the last decade, especially in the educational field. Computational Thinking is also considered an individual skill, and practiced and trained as such [233, 494].

However, such an approach does not match current teaming structures of both science and business, where problems and projects have become so complex that a single individual cannot handle them within a reasonable time frame. To handle the increasing complexity, especially in engineering software systems, developers should be educated to act and operate as a team [144]. This is already happening in the business world. In fact, teaming is considered the key driver to Digital Transformation, where solutions are not provided by individuals but by self-organizing teams [145]. Digital Transformation is often subject to"wicked problems", which do not have an unique solution but many Pareto-optimal ones [391]. This also applies to software development when complexity becomes very high [161].

In Software Engineering, the role of the team and teamwork in general is especially crucial when Agile methods are used. The Agile principles acknowledge that important information and know-how might not be available at the beginning of a project [387]. Reaching the development goal requires several iterations, to build incremental solutions of increasing value for the users.

A key agile team–building factor is *self-organization*, meaning that any member of the developing team contributes with her knowledge, ability, and technical skills in order to work out a solution. Since each team member is responsible for the project as a whole, it is in everybody's interest to organize work at best – not bounce responsibilities. Moreover, teams are not static but they modify their structure according to necessities, which change over time. Not surprisingly, some organizations have built their competitive advantage and success on this model [18]. They comply with Conway's Law, according to which "organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations" [107]. A consequence of this observation is that organizations have to modify their *communication structures* accordingly to the problem which need to be solved. Therefore, flexible and self-organizing teams are best suited to comply with such pivotal evidence for any organization.

We argue that Agile principles and values should enrich the current efforts to establish Computational Thinking as a fundamental literacy ability. We call such a combination Cooperative Computational Thinking, or *Cooperative Thinking* for short. From a pedagogical perspective, it is grounded in Johnson & Johnson's Cooperative Learning approach, where students must work in groups to complete tasks collectively toward academic goals [229]. We suggest a team-oriented approach to educate software engineers in Computational Thinking. Educators should not just promote some good Software Engineering practices; rather, they should foster collaboration skills and train student teams to cooperate on wicked problems. Programming skills are usually considered personal ones; in most cases — job interviews, university exams, official certifications — the focus is always the performance of the *individual*. We lack a general approach to enable group skills in this context. Even if this idea may be widely shared by the community, we did not find any evidence of a comprehensive approach to it. This is probably due to the lack of explicit awareness of such concept as enabler of Digital Transformation processes: we may use it implicitly without recognizing it.

In this chapter we analyze processes and interactions in four different learning modalities that mirror some standard software development models: solo programmer, pair programmers, self-organized teams, and directed teams. We report differences, practical and educational issues, their relative strengths with respect to developing Computational Thinking skills on one hand and how they impact Agile team-related skills, that form the base of Cooperative Thinking, on the other.

As a result, we developed a model for Cooperative Thinking, contextualizing in relevant pedagogical theories. We provide results based on empirical and theoretical evidence; they can be applied to daily teaching practices.

This chapter is organized as follows. Section 8.2 provides background information on related research on Computational Thinking and Agile education. In Section 8.3 we present the methodological framework used for this research synthesis. Section 8.4 presents the investigations we performed in teaching Cooperative Thinking comparing four modalities for organizing software development classes. Aggregated insights from our synthesis are presented in Section 8.5, where we propose actionable solution for educational practitioners. We discuss the synthesis of our research in Section 8.6, presenting the details of the extension of Computational Thinking with Agile practices, that we call *Cooperative Thinking*: self-organized teams are an effective way to enact and support Cooperative Thinking. Finally, in section 8.7, we summarize our vision, outline our future research, and draw our conclusions.

8.2 Related works

Computational Thinking has generated a lot of interest in the scientific community [486]. It is related to problem solving [368] and algorithms [246], because it is the ability of formulating a problem and expressing its solution process so that a human or a machine can effectively find a solution to the stated problem.

However, several scholars argue whether the Computational Thinking concept is too vague to have a real effect. For instance, a recent critique has been advanced by [126]. He claims that Computational Thinking is too vaguely defined and, most important in an educational context, its *evaluation* is very difficult to have practical effects. This same idea can be found in the CS Teaching community. [34] and [214] for example, try to decompose the Computational Thinking idea itself, in order to have an operative definition. [198] notes that computing education has been too slow

moving from the computing programming model to a more general one. [57] even wonders if the Computational Thinking concept is at all useful in Computer Science, since it puts too much importance on abstracts ideas. It is also remarkable that there is some research trying to correlate CS and learning styles [215, 457, 17], but generally inconclusive.

Though the Agile approach to software development is eventually going mainstream in the professional world, *teaching* the Agile methodology is still relatively uncommon, especially at the K-12 level. Moreover, a Waterfall-like development model is often the main development strategy taught in universities [266]. Moreso, it is usually limited to an introductory level and rarely tested firsthand. In practice, Agile is learned "on the field", often after attending *ad hoc* seminars. Interest in the field is however rising, and curricula are being updated to reflect this [436, 265]. An interesting and complete proposal has been proposed by [304]. The chapter presents the "Agile Constructionist Mentoring Methodology" and its year-long implementation in high school; it considers all aspects of software development, with a strong pedagogical support.

To summarize, programming remains a difficult topic to learn and even to teach, both at university and high school level; the ability to design and develop software remains an individual skill and taught as such.

Some studies, however, tackle the idea that hard skills expertise should be complemented with soft skills, possibly introducing active and cooperative learning [231]. For example, in [392], a long list of so-called soft skills expertise is paired with various developer roles. In [84] the problem is well analyzed, but arguably the proposed solution is not comprehensive. [305] presents an example of how to promote cooperation within a software project; however generalizing the proposed scheme seems difficult. We note however that the approach is hardly systematic, and no general consensus exists on how to proceed along this line.

8.3 Research Methodology

Meta-analysis is a widely known and old research procedure, firstly methodologically supported by the work of [175]. The first meta-analysis was probably carried out by Andronicus of Rhodes in 60 BC, editing Aristotle's 250 year older manuscripts, concerning the work *The Metaphysics*. The prefix meta- was then used to design studies whose aim is to provide new insights by grouping, comparing, and analyzing previous contributions. Accordingly, we use the term meta-analysis to indicate an analysis of analyses. In this sense, there are a variety of analysis of analyses, like systematic literature reviews, systematic mapping studies, and research synthesis.

According to [109], a research synthesis can be defined as the conjunction of a particular set of literature review characteristics with a different focus and goal. Research synthesis aim to integrate empirical research in order to generalize findings. The first effort to systematize from a methodological perspective a research synthesis was performed by [108], building on the work of [225], proposing a multi-stage model. The stages are the following: (i) problem definition, (ii) collection of research evidence, (iii) evaluation of the correspondence between methods, (iv) analysis of individual studies, (v) interpretation of cumulative evidence, and (vi) presentation of results.

Following the multi-stage framework suggested in [108], we provide our problem definition, framed as research question (RQ).

- RQ_1: Is Computational Thinking scalable to teamwork?

Table 8.1: Investigation list

Title	Focus of Experiment	# Subjects	Ref
Learning Agile software development in high school: an investigation	Pair Programming, Timeboxing, User Stories, Team Development	84	[312]
Teaching Test-First Programming: assessment and solutions	Pair Programming, Social dynamics	102	[315]
Agile for Millennials: a comparative study	User Stories, Scrum, Waterfall, Team Development, Timeboxing	160	[314]

To answer this question, we looked back to some previous works of ours, investigating the phenomenon on different perspectives.

All analyzed papers are both homogeneous and comparable, as depicted in Table 8.1.

We both provided insights on single papers in Section 8.4, and provide an interpretation of cumulative evidence in Section 8.5.

As an outcome we propose a new educational framework, namely Cooperative Thinking, which we use to answer to our research question in Section 8.6.

8.4 Results

We performed some experiments collecting several insights regarding teaming for solving computational problems, as listed in Table 8.1.

For the purpose of this research synthesis, we abstracted our empirical knowledge and mapped learning models to learner types. To do so, we used the well–known Kolb's learning style inventory [261, 235], consisting in:

- Individual learning (best suited to Assimilators)
- Paired learning (best suited to Convergers)
- Directed group learning (best suited to Divergers)
- Self-determined group learning (best suited to Accommodators)

This classification supports our inductive epistemological approach, allowing us to contextualize already collected evidence into a broader theoretical framework. Hereafter, we make our considerations for the four learner groups.

8.4.1 Individual learning

Directed Individual learning (short: Individual Learning) corresponds to the most common form of teaching, practiced everywhere in practically every subject and most often associated to Directed Instruction.The typical form consists of a lecture on a new topic, followed by individual study and exercises, then finalized in some kind of assessment (test and/or capstone project); teaching is generally sequential, each concept built on previously acquired knowledge. The main advantages of this model are its simplicity and efficiency; a single individual can teach a full class of people at the same time; moreover, we all already have have plentiful experience with this method.

More recently, by using modern technology this model can scale almost indefinitely. Practical examples include the Khan Academy, Udacity, and other MOOCs. An interesting aspect is that the sequential progression is ideal for stimulating Computational Thinking concepts — especially Problem Solving. Once a topic is mastered, it can be used to tackle more complex concepts or deepen and reinforce the significance of an acquired one. Another advantage is assessment; for instance, it is very easy to evaluate a program written by a student thanks to standard testing frameworks, to the point that automatic evaluation is becoming more and more common — a decisive point in case of e-learning on very large classes.

This model is dominant, however it has several limitations. One of most important ones is the fairness of the assessment. The difficulty of the assessment test is usually tailored upon the average student, resulting in a Gaussian curve grade distribution. In this model, students falling behind at the beginning of the course rarely have the capacity to catch up, as the time allotted is the same for every student; additional information, requirements, time–demanding exercises pile up very quickly. People experiencing learning difficulties have very few options. Those who can afford it resort to privately paid tuition, but for the rest the road a failing grade is almost certain. A consequence is the so called "Ability Myth": it states that each of us is born with a set of abilities that hardly change during our lifetime [438]; in fact, this phenomenon is in most cases the effect of accumulated advantages [435].

Another drawback is the absence of positive social interaction. Direct teacher/student communication is constrained by the available time. Student/student interaction rarely includes exchanges of ideas or effective cooperation; more often than not, it results in direct competition or in nonconstructive and illegal help (i.e. cheating). All of this might have a negative influence on overall motivation, especially in less-than-average performing students.

In our experiments [312, 314, 313], we simulated a working day in a software house; the teacher assumed the role of the software house boss, and selected a number of students who were previously categorized as either "good", "average" or "poor" programmers. Each student was given a moderately difficult task using a new work methodology (either TFD or User stories) within a limited time-frame. Without much surprise, both performance and the perceived utility of the activity mirrored their current skill level.

Individual learning help foster Computational Thinking but it is not useful (or maybe detrimental) to develop social skills needed for Cooperative Thinking. According to Kolb's learning inventory [261], this teaching model best suits *Assimiliative* learners, since they like organized and structured understanding and respect the knowledge of experts. They also prefer to work alone.

8.4.2 Paired learning

Paired learning (also called *Dyadic Cooperative Learning Strategy* [427]) is also a common technique but far less popular than the previous one. The basic principle involves the teacher posing a question or presenting a problem, then the students discuss in pairs and find their own way toward the solution; pair members are often switched, sometimes even during the activity.

The role of the teacher in this case is quite limited, as she acts as a general coordinator and facilitator of the class of pairs. In the software development field, we find an obvious transposition of this model in Pair Programming, one of the key Agile programming practices and, to a lesser extent, in some training techniques (Randori and Code Retreats among others [394]).

According to [121], this model has beneficial influence on retention, understanding, recall, and elaborate skills at the cognitive level; it is particularly effective on mood and social skills, and introduces the idea of software being an iterative, evolutionary process. As it promotes knowledge sharing, it can help less skilled individuals to improve themselves taking inspiration from their partners. However, it is more difficult to develop a teaching progression using only this model, and in any event, it would be rather slow. In addition, psychological and personal factors become important, since partner incompatibilities and social difficulties might dramatically change both the learning outcomes and the quality of the code produced. Assessment is more difficult than in the previous case; though automatic evaluation is still possible, some extra steps are required by the teacher to deduce the effective contribution of each member of the pair to the final work.

We tested firsthand this effect in our experiments [312, 315]. We proposed the same method and problems stated in Sect. 8.4.1, but in this case we paired students according to six possible pair types, classified as homogeneous (good-good, average-average, poor-poor) or as non-homogeneous (good-average, average-poor, good-poor).

According to our results, homogeneous pairs performed generally equal or worse than their solo counterparts, but non-homogeneous pairs had statistically better results. In the latter case a form of epistemic curiosity [230] appeared, possibly unconsciously, and was a key motivating factor for the pair; the resulting interaction helped both to solve the task at hand and to develop social skills. Computational Thinking was also stimulated, but a little less than with the previous model, since the "effort" was split and each single task was not really challenging, requiring expertise more than logical reasoning.

In addition, both students and teachers praised the new methodology and its positive effect on mood. However, the retention rate was very low, much worse than expected; in an interview conducted some time after the experiment was over, students generally only had a vague idea of the techniques used and only about 5% of them was able to name them correctly.

To summarize, paired learning has beneficial effects on social skills related to Agile development, and generally is useful in leveling skills upwards. Knowledge building will however be much slower than in the traditional approach. This teaching model better suits *Convergent* learner types, since they want to learn by understanding how things work in practice, like practical activities and seek to make things efficient by making small and careful changes.

8.4.3 Directed group learning

Group learning is one of the many facets of Cooperative Learning, which is becoming fairly common in modern, constructivist-influenced education [78]. It is also a common practice in some working environments, notably in the health context for nurses [482]. Group learning in a Software Engineering lab class is best exploited by developing a full software project, not simple exercises or abstract analyses. So, it is natural to join Group Learning and Project-Based Learning strategies, especially using the Jonassen variant [222]: a complex task taken from real-life with authentic evaluation, comprehensive of all phases of development.

We are aware that many software development methodologies exist, and each of them can be transposed in an educational context promoting different behaviors and skills. One of them is the Waterfall model, probably the oldest one but still quite popular in the industry.

Waterfall embodies in many ways all the tenets of our prevailing culture, such as linear hierarchies, top-down decision making, accepting the assumptions, acquire all information in order to prepare a detailed plan and then following it — values that have forged the way traditional education was conceived and in most cases is still carried out.

A Waterfall school project will see the teacher assume the role similar to that of a senior project leader, assigning tasks and roles to students according to their skill, knowledge, and ability and applying a certain degree of control. The teacher's role will be very important at the beginning of the project, as students generally lack the ability to perform a thorough analysis and comprehensive design phase. As the project continues its course, the role will be more oriented to control, checking that documents are properly written, modules developed and tested, directing the flow of the entire operation. Assessing a group project is considerably more complex that both previous model, since it involves not only the final product, but also the process used and the interaction among the student and their relative contributions. To resolve it, usually a combination of traditional evaluation (automated or not), direct teacher observation and peer evaluation is used, forcing students to evaluate and reflect on the quality of their work.

In a different experiment, we decided to give students a very challenging task, almost impossible to solve. They had to build from scratch a complete dynamic website, a task we estimated in about 30 man-hours to complete when handled by experts. We only gave them 6 hours. This forced teams to make hard decision as to what was the most suitable course of action in order to make the best use of the allotted time and resources.

Then some extra restraints were imposed on the group, such as:

- A rich set of artifacts, such as a complete SRS, ER-diagrams, management priorities, UI-Mockups.
- Specific roles (programmer, UI-expert, tester, ...) and hierarchies (chief programmer, for example) were imposed.
- A predefined time schedule.

From an educational viewpoint, the target product was definitely outside a single student's zone of proxymal development [471], but was theoretically doable as a team effort. From a different viewpoint, such a target looks like a wicked problem, since students lacked the knowledge and the competence to complete the task, and were requested to acquire them along the way [476]. The great amount of information and in general the directive role of the teacher gives the opportunity to highlight whatever learning goal is deemed important.

Results show that, under these conditions, groups tend to concentrate on non-functional requirements and process-related goals instead of pursuing the main goal: delivering a working product to the "customer". The products, on average, had very few working features, but the defects were hidden under a pleasant user interface, close to the one proposed by the "management". Roles were interpreted rather closely to the given instructions (barring a few cases of internal dissent), timing was impeccable, and even the documentation was acceptable. Teacher-student interactions were not intense, but rather limited to simple yes-no questions. Students reported great satisfaction for both the activity and the product realized, asserting it was an activity both useful and fun [314].

To summarize, this teaching model promotes the use of social skills, while leaving the steering wheel in the hand of the teacher. This power can be used to provide a

meaningful learning path, though slower that Individual Learning and with a non-trivial evaluation method. It also does not seem to stimulate enough other interesting skills, such as decision making. It better suits *Divergent* learner types, since they and will start from detail to logically work up to the big picture. They like working with others but prefer things to remain calm.

8.4.4 Self-directed group learning

This model is a different version of Group Learning, radically different than the previous one in that students have a strong degree of autonomy. It applies to K-12, adult education and business/industrial environments, for example [186].

In this case, the teacher becomes a mentor and a facilitator, and invests a large amount of trust on the learners.

Most of what we said on Project-Based Learning in the previous subsection holds. In this case, the granted freedom can be a powerful weapon in the hands of the group, but it might also backfire.

It is easy to see that several Agile values are connected to this learning model: most prominently, shared responsibility and courage. Agile strongly promotes an adaptive approach to software development, where each iteration acts as a feedback for the next one. Teams should be self-organized, and great emphasis is put on communication, both within the team and with the stakeholders. This means that the teacher must *become part of the team* in order to maintain a high level of communication. It also means that the teacher cannot distribute grades in a standard way, as he will be directly involved in the process (effectively becoming a 'pig', and not a 'chicken', referring to the classic Agile metaphor). Grades should therefore come from reflections, group and/or personal and peer evaluation, and must include an evaluation of teacher work, as any other team member.

In our experiment, we kept the same general structure outlined in section 8.4.3, but within the same class we assigned the same project to a different, potentially equivalent, team. This allowed for a direct comparison of results, since it ruled out biases due to different teachers, learning environments, or curricula. We have chosen the Scrum methodology, because it is arguably very different from Waterfall and it does not really mandate any practices, giving maximum freedom to the teams [399]. The teacher assumed the role of the Product Owner in this specific case; alternatively, the Scrum Master role could be chosen as well [314].

The teams were given much less information and limitations with respect to Waterfall teams:

- A list of prioritized user stories.
- A 'definition of done' (as in Scrum): it is a definition of how a result can be considered to have some value, in terms of simple activities like writing code in a standard format, adding comments, performing unit testing, etc.
- The sprint length.

Everything else was to be decided by the team. Scrum teams also had the additional difficulty of having no experience with self-organization, whereas traditional Waterfall methodologies and roles were taught as part of standard curriculum.

Results show that Agile teams performed generally better than their Waterfall counterpart in the same class with respect to overall product completion and number of featured delivered. This is not surprising, since Agile privileges the functional dimension over the non-functional ones. It is interesting to note that many chose

challenging but interesting tasks, possibly failing along the way. However, with respect to code quality, Agile teams fared worse than their counterparts. First, code was less readable and with worse Cyclomatic Complexity evaluation; second, the final product on average had severe usability problems, since this was not an explicitly stated goal. In general, teams underestimated the effort needed on the first sprint but guessed much better their second sprint, during which they were much more productive. Teacher-student interaction was also not very intense – suddenly cooperating at peer level with an older, experienced superior is not an easy task for anyone. Students reported great satisfaction for this activity, slightly more than for the previous model.

So, both types of Group learning (directed like Waterfall and self-directed like Scrum) missed the main point of the activity, which was to provide a valuable product for the customer. What is interesting is the motivation for such failures. Scrum teams concentrated their effort to reach a goal, possibly a difficult one, displaying Courage, a key XP value. Waterfall groups tended to "play safe", and concentrated on less risky objectives (user interface, process oriented goals) and working on what they most comfortable with, a pattern more in line with logical reasoning.

The self-directed group model strongly promotes the use of social skills and other qualities relevant to Cooperative Thinking. However, the learning rate could be exceedingly slow; moreover, evaluation requires great attention and balance. It better suits *Accomodative* learner types, since they display a strong preference for doing rather than thinking. They do not like routine tasks and will take creative risks to see what happens.

8.5 Implications for practice

Kolb's model identifies four basic types of learning experiences (Active Experimentation, Concrete Experience, Reflective Observation) and four basic types of learners (Converging, Accommodating, Diverging, Assimilating). Kolb suggests to alternate these learning modalities in order to stimulate different aspects of the learners' mind, even if an individual is more oriented to a specific king of learning activity. We therefore classified four types of learning experiences specifically related to lab classes that can be appealing to a particular learner type, as shown in Fig. 8.1.

Table 8.2 summarizes the content of this section. Traditional teaching concentrates on individual learning, thus favoring Assimilating students; we argue that a more balanced approach is beneficial in general, and in particular can stimulate and develop focused social skills that are essential for developing an effective Cooperative Thinker.

We understand that Kolb's classification is crude, as it cannot capture the complexity of teaching and learning in a social environment, be it at school or on the workplace; yet, even this simple model is powerful enough to analyze the situation and plan activities to reach our goals.

Cooperative Thinking is a general theoretical concept, just like Computational Thinking. Educators should do their best in order to have students understand and

Table 8.2: Learning model influence on learner and teacher's role

	Teacher Role	**Learning Path**	**Computational Thinking**	**Social skills**	**Agile skills**	**Ease of Evaluation**	**Preferred Kolb Learner Type**
Individual Learning	Boss	+++	++	–	–	++	Assimilator
Paired Learning	Facilitator	-	+	+	++	+	Convergent
Directed group learning	Project Leader	+	=	++	+	-	Divergent
Self-directed group learning	Teammate	-	=	+++	+++	–	Accommodator

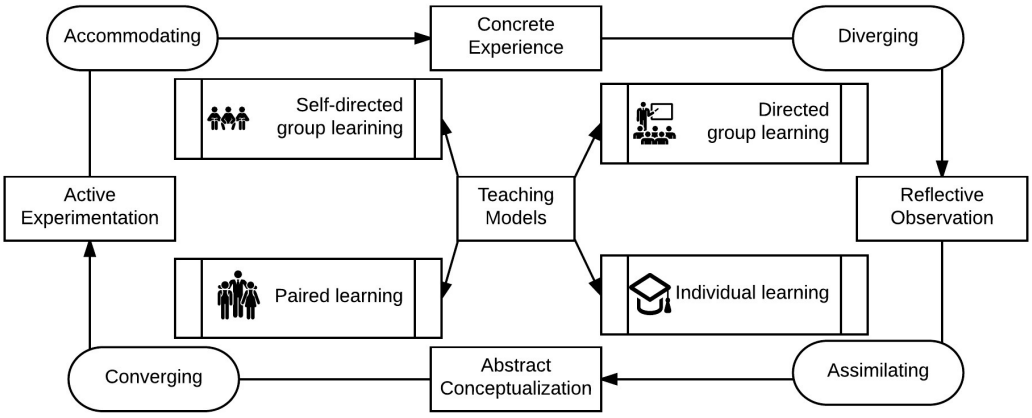


FIGURE 8.1: Teaching activities mapped to learner types, following the taxonomy of [261]

be able to put theory into practice. Is the educational system able to accept this change? Our discussion concentrates on teaching software development lab classes.

Usually, only individual performances are evaluated in lab classes of both high schools and colleges alike: it is less common to evaluate the teamwork. We will now describe some teaching models that can be used to promote the emergence of the two pillars of Cooperative Thinking: Computational Thinking and Agile practices. We have evaluated the impact on students designing and performing a series of learning experiments that exposed software development students to Agile practices and values.

In this chapter we analyzed a series of teaching strategies for software development, each with advantages and disadvantages and having a different impact on cognitive, reasoning and social skills that collectively concur to create what we called Cooperative Thinking.

Traditionally, education has considered literacy and knowledge in a broad sense. Consequently, the quality of education is often tied to fundamental skill expertise; one of the most recognized indicator is the result of the international PISA test, that evaluates how effective a country has been at deploying their prescribed math, science, and reading curricula. In this perspective, it makes perfect sense for educational institutions worldwide (and universities foremost) to favor individual learning as the primary – if not only – teaching strategy. For instance, a consequence of this is that several efforts are spent in schools on overcoming individual differences among students: see for instance the well known discussion of the "Matthew effect" in [435], which is a social selection process resulting in a concentration of resources and talent.

However, in the future, "pure" knowledge might become less important, even to the point of becoming a commodity, and soft skills could raise in importance. An educational system focusing on hard, technical skills could have difficulties in promoting soft skills. As [500] pointed out, there is an inverse correlation between PISA test scores and entrepreneurial capacity, a measured by the Global Entrepreneurship Monitor (GEM), the world's largest entrepreneurship study. Specifically, the countries with the top PISA scores had an average GEM:PISA ratio of less than half of the mid- and low-scoring countries, indicating a potential shortfall in PISA's measuring purpose to understand if students are "well-prepared to participate in society" [355]. And this might as well be true in Computer Science.

Notably, the ability to solve complex issues or *wicked problems*, is a requirement for new product development and innovation & entrepreneurship in general [80]. Wicked

problems usually have no single perfect solution but many Pareto-optimal solutions. The traditional educational paradigm is not tailored to train people able to handle similar situations; PISA-like evaluations are meaningless to determine the educational system's efficiency, since the only offers an evaluation of the *individual*.

So, the gap between a formal educational background and real-life wicked and complex problems becomes larger. Actually, it will increase along with Digital Transformation processes, where the level of predictability decreases and uncertainties increases [381].

Therefore, the introduction of other teaching strategies that foster social skills and cooperation is very important, and should also be factored in grading activities. Note that we do not advocate a complete suppression of the Individual Learning strategy; on the contrary, it should be *complemented* with other strategies in order to obtain an overall balanced and blended mix tailored to specific situations – there is no silver bullet in education. This proposal will also have the extra bonus of potentially appeal to all learner types, even those that traditionally are less inclined to pick Computer Science as their course of study.

Given all the above considerations, we recommend all strategies we mentioned be used in teaching software development, in order to promote different but equally important skills and possibly favoring different learning styles. This strategy mix should begin as soon as possible and continue throughout the entire study path, up to and including the university tier. Otherwise, it might be too late to develop the full potential of Agile-related skills and, consequently, Cooperative Thinking.

8.5.1 Learning path

We assume that most CS courses are strongly oriented toward individual learning, the goal being to introduce and grasp the basic elements of CS and, specifically, programming; a short to medium-length programming project of average difficulty is usually included.

As soon as possible, Pair Learning should also be presented. Specifically, Pair Programming should be introduced first and actively enforced as one of the main practices for class exercises throughout the course. Other Agile practices could be introduced (such as Test-First Development, Continuous integration, ...) along with the necessary software tools (like git or Jira). A project that verifies what students learned should be simple in terms of programming complexity but rich in process experiences, in that elements of Agile must be used and their use verified.

Next, forming the team is an important factor. We know that simply putting together people and telling them to work on a project is not enough to have an even decently efficient group. Preparation is in order, requiring some careful people selection, team-building exercises, and some short project to test how the teams work. Finally, a team-oriented project of moderate to high difficulty and length should be realized by the students.

The final step is, of course, proposing a demanding project to the teams and give them ample freedom. At this point students should have a solid knowledge of the programming language and development methods, a grasp of basic Agile practices, and some working experience with all necessary tools; moreover teams should know their strengths and weaknesses. This activity can actually be a course capstone project and should contribute significantly to the students' grade.

Our proposal requires formalization, testing, and formal validation. Though every step is nothing new or complicated, the overall teach process is. Our research group

is currently working on a comprehensive proposal and its field testing in both K-12 and university classes.

8.5.2 The influence of the context

We discuss now the validity of this study in the different contexts of High School and University classes.

First, we examine some distinctive features of learning in high school:

- The learning activities encompass several years. During this long time period, teachers and learners get to know well each other and develop a relationship that has strong effect on the quality of their cooperation.
- The evaluation of the students is based on several factors. One is certainly the overall performance (tests, lab results), but many other aspects are factored in: initial level, handicaps, effort, proper behavior. This implies that the teacher must exert some form of control and surveillance, even due to age considerations.
- Learning goals tend to be broad-scoped, leaving advanced topics only to the best students.

The University learning context *seems* to be completely different. Instructors usually teach for a single semester, a time insufficient to establish a personal relationship. Performance evaluation is far more important, overshadowing other factors; standardized tests and procedures are used, focused on both general and specific topics. Higher levels of personal responsibility and self-organization are expected, so teacher control is generally limited.

However, in the specific case at hand, differences are not so well marked. We performed our experiments in high school courses (total: about 250 students) which are programming intensive, featuring around nine programming hours - labs included - per week for three full years. They cover basic and intermediate programming issues, including dynamic data structures, recursion, and databases for an average of 300 programming class hours per year, personal study not included. While we do not claim that this kind of education to software development is equivalent to a standard undergraduate level lab class in software development, it is undoubtedly comparable, on average compensating subject depth and personal motivation with more time spent in practical experiences. Our experiments on undergraduate students (total: about 90 students) confirm these impressions.

Not surprisingly, we found that our teaching strategies had to be adapted to the different educative levels. For example, students in high schools require learning activities on Agile to be repeated and, at least partially, integrated into standard teaching activities. Failing to do so inexorably results in limited long-term retention, as some interviews sadly demonstrated. Moreover, students must concentrate on Agile practices rather than on the overall development process; they are only able to handle a software project of limited scope and complexity, so setting up a full-fledged development environment (be it Agile or else) looks like an overkill.

Conversely, undergraduates are able to make the most out of one-shot activities; they are expected to reinforce their knowledge and skills with personal work, and most of them indeed do. They have sufficient capabilities and time to properly apply a standard Agile development cycle, especially in capstone projects. The problem in this case is the large amount of topics to cover: the instructor has the responsibility to select the topics that must be taught. In addition, undergraduates have a higher degree of freedom, so they cannot be forced to adopt a given method or practice. The

effective use of Agile by students depends on their personal and, for some part, on the charisma of the instructors.

8.6 Discussion

In 2006, Jeannette Wing's paper defined and popularized the concept of Computational Thinking [486], portrayed as a fundamental skill in *all* fields, not only in Computer Science. It is a way to approach complex problems, breaking them down in smaller problems (decomposition), taking into account how similar problems have been solved (pattern recognition), ignoring irrelevant information (abstraction), and producing a general, deterministic solution (algorithm).

Even after more than a decade, the impact of this idea is strong. Eventually, some governments realized that future citizens should be *creators* in the digital economy, not just consumers, and also become *active citizens* in a technology-driven world.

Computational Thinking needs to be properly learned and, therefore, is being inserted as a fundamental topic in school programs worldwide. This is a welcomed change away from old educational policies that equated computer literacy in schools to the ability of using productivity tools for word processing, presenting slide shows, rote learning of basic concepts. Though useful in the past, they are currently outdated and even possibly harmful. The US initiatives *"21st Century Skills"* [146] and curriculum redefinition, along with *"Europe's Key Skills for Lifelong Learning"* [106] should be viewed in this perspective.

However, these approaches might not be sufficient in the long run. Current educational approaches concentrate on coding (as an example, consider the *Hour of Coding* initiative), but this it not the end to it. Computational Thinking is made of complex, tacit knowledge, that overcomes limited resources and requires deep engagement, lots of deliberate practice, and expert guidance. Coding is one aspect, and not necessarily the most important one.

Tasks solved by software systems are becoming more complex by the day, and many of these in the real world could be classified as *wicked problems* [391]. There is no single "best solution" to many such problems, only Pareto-optimal ones which may change over time. In this situation, satisfying expectations and requirements becomes harder and harder as they are beyond the limit of solvability for any single programmer.

This is well known in the fields of Science and Business. The most common approach to trying to solve wicked problems in these fields is by forming teams including people with complementary backgrounds, trained to face problems and reach the goal – together. These new cooperative entities benefit from a high degree of independence and autonomy to deal with the assigned task; the idea is to solve a problem attacking it from different points of view.

Even if Computational Thinking has been defined as a problem-solving skill, and has benn taken as the basis for several ongoing activities, by itself alone it does not offer the variety of viewpoints required to solve difficult or wicked problems. Computational Thinking has traditionally been considered an individual skill, and taught as such. Teamwork and soft skills are generally not factored in, and even shunned as "cheating" in some introductory programming courses.

In our view, the general approach to Computational Thinking needs to be updated, by enhancing it with a complementary concept: Agile values and practices. The Agile Manifesto was published in 2001, just a few years before Wing's paper. In just 68 words, it proposed a quite original perspective on software development, recalling

values that clashed with the established culture of time, based on top-down hierarchies, linear decision making and, in general, pursuing unsustainable management plans. The most significant change introduced by the Agile movement is the paramount importance assigned to face-to-face communication and social interaction, superseding the internal organizational rigidity, documentation, contracts, roles, and more [387].

Including some Agile principles and learning-as-execute experiences in training for Computational Thinking is beneficial. We name *Cooperative Thinking* this Agile extension of Computational Thinking, and define it as follows:

"Cooperative Thinking is the ability to
describe, recognize, decompose, and computationally solve problems
teaming in a socially sustainable way"

This definition joins the basic values of both Computational Thinking and the Agile Manifesto.

Computational Thinking is based on the power of abstraction, problem recognition and decomposition, and algorithms. Agile principles include self-organizing teams, interaction and communication, and shared responsibility. Both Computational Thinking and Agile value the concepts of evolution and reflection of problems and solutions. Both approaches share the idea of problem solving by incremental practices based on learning by trial and error. Moreover, our definition of Cooperative Thinking underlines *sustainability*, since "solutions" as such have little impact, if not related to the available resources.

In sum, Computational Thinking is *the* individual skill to solve problems in an effective way. We found that Agile values are central not only for developers but also for educating individuals. Cooperative Thinking adds a variety of points of view required to solve really demanding and complex tasks. Enhancing Computational Thinking with Agile values and principles allows to exploit the power of a team of diverse backgrounds towards a common goal. Being mentally flexible, understanding the others' points of view and synthesizing a common solution are crucial skills for teaming developers.

8.7 Conclusions

In this chapter we explored Cooperative Thinking, a concept that expands Computational Thinking embracing Agile values. The proposal is graphically summarized in Fig. 8.2.

Cooperative Thinking is the extension of Computational Thinking with Agile Values. We considered the skill breakdown proposed for Computational Thinking by *Computing at School* [115] and grouped the skills into three broad categories: Problem solving, Evolution, and Reflection. Correspondingly, we considered Kent Beck's XP values and practices list [**BecAnd04**] as representative of Agile values and practices in general; list items were also grouped in three categories: Social Skills, Evolution, and Reflection.

Cooperative Thinking is a complex skill to acquire and master, but in our view, is the way to go to obtain teaming individuals able to tackle and resolve the challenges and questions that the future will present them.

We examined four different learning models, each with a different balance of traditional, Agile, and Cooperative learning, showing the impact they had on students in developing Cooperative Thinking. Specifically, Individual learning is strongly related

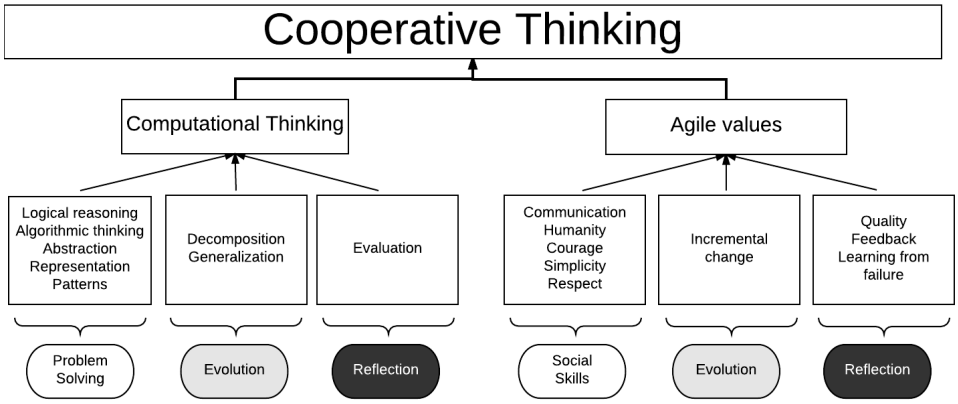


FIGURE 8.2: Cooperative Thinking, Computational Thinking and Agile values breakdown (according to *Computing at School* [115] and [43])

to Problem Solving, Social Skills to Self-directed group learning; all other aspects have a varying degree of relationship to the different models.

Our experiments showed a significant effect on the learning outcomes. Cooperative Thinkers will enjoy an edge on the job marketplace, making them more flexible, socially aware, and more able to handle future challenges, be they related to software development or not.

In order to educate students to Cooperative Thinking, we suggest that a mix of learning strategies be used, in order to expose students to Agile practices and values and develop teaming skills without forgetting basic Computational Thinking skills, such as abstraction. While we do not claim the superiority of Agile practices as such, we do observe their effectiveness as *enablers* of Cooperative Thinking, since they promote interaction, force efficient resource handling, and are strongly goal-oriented, substantially more than individual learning.

We propose to define and evaluate innovative educational programs promoting Cooperative Thinking. Mixed methods assessments for educational construct validation with Structural Equation Modeling as also fine granular performance indicator for Pareto-optimal solutions need to be validated. However, finding the exact blend of teaching strategies will be the real challenge for the Software Engineering community; this is exactly what we are investigating now, both at K-12 and undergraduate level.

Another line of research that we intend to pursue concerns the constructs which constitute Cooperative Thinking, especially concerning teaming [133]. For instance, the dynamic structure of teams is interesting: we have seen in our experiments that in pair programming asymmetry of competences is quite effective. In teams including more people, say four or five students, we intend to study the emergence of mentors as facilitators rather than leaders, and the impact of such figures on self-organization of teams.

Chapter 9

An Empirical Validation of Cooperative Thinking

9.1 Introduction

New skills are required for the future workforce to get employed in the Computer Science (CS) domain [477]. Several technological trends support novel requirements - mobile Internet and cloud technology, advances in Big Data, advanced robotics and autonomous transport, artificial intelligence and machine learning, advanced manufacturing and 3D printing and High Performance Computing, new materials, biotechnology and genomics, just to cite a few [477]. Future workers will need to think differently, to solve their working problems: those solved by software systems are becoming more complex by the day. Some problems in the real world can be classified as *wicked problems* which usually do not have an unique solution but many Pareto-optimal ones [391]. In other words, these problems outline trade-off situations, where the notion of Pareto optimality is applied to the selection of alternatives: each option is first assessed, and then a subset of those options is identified, with the property that no other option can outperform any of the chosen options.

Accordingly, the education system needs to train students on such new challenges. Novel initiatives were promoted by institutions in several countries, like for instance the US *"21st century skills"* [146] and *"Europe's Key skills for Lifelong Learning"* [106] initiatives, that prompted the redefinition of Computer Science curricula:

> *[...] to empower all [...] students to learn Computer Science and be equipped with the computational thinking skills they need to be creators in the digital economy, not just consumers, and to be active citizens in our technology-driven world. Our economy is rapidly shifting, and both educators and business leaders are increasingly recognizing that Computer Science is a "new basic" skill necessary for economic opportunity and social mobility.* [1]

The idea of a "new basic skill", according to this view, derives from the fact that computational proficiency became a traversal skill for all domains, complementing the soft skill areas. Modern education theories, such as Constructionism [351], promote critical thinking as opposed to mere memorization; teaching practices such as Cooperative Learning [231] and Problem-based learning [222] also introduce organizational and social skills in the educational process.

However, an analysis about the educational constructs used in CT and AV is missing in literature. Single constructs were presented, like Computational Thinking (CT)

[1] https://obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all. Accessed on 22.01.2018.

[486] for the Computer Science domain, in general, and Agile Values (AV) [43] for the Software Engineering one in particular. Surprisingly, a study about a comprehensive educational implementation of such constructs is lacking.

CT and AV represent complementary skills of Computer Science education for software development [...]: respectively, the individual ability to produce computationally efficient code, and the social ability to interact with both peers and stakeholders to deliver valuable software. Nevertheless, CT and AV have also practical implications in the broader Computer Science domain. The rise of complexity (and wicked problems) is not only a problem of Software Engineering, but engages all Computer Science areas. The interdisciplinary interaction between different hardware and software components, along with a context–dependent knowledge is a common scenario for most areas. As an illustrative example, IoT is moving to new paradigms due to raising complexity of computing (e.g., fog computing, context-aware computing) [319]. Here, the role of teams are crucial to address these topics, since they are both interdisciplinary and complex.

We argue that these two core skills are part of the higher level skill of *Cooperative Thinking* (CooT), which is, in our view, *the ability to describe, recognize, decompose problems and computationally solve them in teams in a socially sustainable way* [...].

We have developed the concept of CooT working with both high school and university students. Our initial idea was to exploit an agile approach to let teams to solve problems requiring Computational Thinking [...]. We started with teams composed of pairs, then scaled to self-organizing groups up to six students. We realized that *socially sustainability* is important: in particular we found that heterogeneous groups are more effective than homogeneous groups [...]. We noticed that such groups were able to handle complex problems more effectively, due to their ability to team up through peer education and communication. Especially for software developers, communication structures are essential to understand the way they design software: this is called the Conway law [107]. Since communication impacts the way they design software systems, it is necessary educating future developers to manage properly their organization of work (i.e., dealing with customers, rely upon fellow developers, be able to discuss algorithms, etc.). It should be socially sustainable, since a developer should be able not only to deliver her specific task (e.g., developing some piece of code), she should also interact effectively with her social context (e.g., internal and external project's stakeholders, laws and regulations). Educating students to deal responsibly with their social context means to make them aware that a socially sustainable work organization is important to solve complex problems. We are less interested to educate solo developers who provide fast algorithmic solutions, regardless of their social communication structures. Indeed, social sustainability is a new element which is additional to both Computational Thinking and Agile Values.

CooT is not a specific technical skill, as such. It focuses on cooperative problem solving of technical contents. So, CooT is not just the sum of two constructs, rather it "explains" other crucial educational constructs, like for instance continuous learning and social adaptability. We will discuss in Section 9.6 how CooT influences these and other educational constructs. To evaluate this assumption and test the model we have developed, we used Partial Least Squares Structural Equation Modeling (PLS–SEM). This research method enables researchers to assess if the relationships among different theoretical constructs are statistically significant in the surveyed population.

This chapter is organized as follows. In Section 9.2 we present the related literature. Subsequently, in Section 9.3 we discuss our research model with the underlying hypothesis. Then, we describe our research methodology in Section 9.4, along with a brief explanation of PLS. Afterwards, we validate the results obtained with PLS in

Section 9.5. The analysis of our findings with the study limitations is in Section 9.6. Finally, we outline future works and our conclusions in Section 9.7.

9.2 Related Work

There is a growing belief that complex problem solving, critical thinking, creativity, people management, and coordinating with others will become the most important job skills by 2020 [477]. According to the World Economic Forum, future companies will actively search for employees who can master "capacities used to solve novel, ill-defined problems in complex, real-world settings" and "motivate, develop and direct people as they work, identifying the best people for the job, also adjusting actions in relation to others' actions" [477]. So, skills to think in a computational friendly way and to solve them in a social and sustainable manner are both required. Apparently, CT and AV skills are strictly connected for companies, as suggested by the World Economic Forum [477].

Since 1945, several scholars have been theorizing *ante litteram* about Computational Thinking [368, 291]. The idea of "algorithm" became popular after 1960 when Katz suggested that automated processes would spread well beyond the Computer Science domain and would influence all fields [246].

In 2006, Jeannette Wing's paper introduced the concept of Computational Thinking [486], portrayed as a fundamental skill in *all* fields, not only in Computer Science. It is a way to approach complex problems, breaking them down in smaller problems (decomposition), taking into account how similar problems have been solved (pattern recognition), ignoring irrelevant information (abstraction), and producing a general, deterministic solution (algorithm). Today, governments are realizing its importance, and update school programs worldwide (like the US initiative *"21st century skills"* [146]).

However, more and more scholars argue whether the CT concept is too vague to have a real effect [126]. Denning claims that CT is too vaguely defined and, most important in an educational context, its *evaluation* is very difficult to have practical effects [126]. This same idea can be found in the CS Teaching community. [34] and [214], for example, try to decompose the CT idea itself, in order to have an operative definition. [198] notes that computing education has been too slow moving from the computing programming model to a more general one. [57] even wonders if the CT concept is at all useful in Computer Science, since it puts too much importance on abstract ideas. We also noted that, barring some limited work [215, 457, 17], there is not much research on CT and learning styles.

Though the Agile development is eventually going mainstream in the professional world, *teaching* the Agile methodology is still relatively uncommon, especially at the K-12 level; there are a few exceptions [436, 265]. A Waterfall-like development model is often the only development strategy taught in universities [266]. Moreso, it is usually limited to an introductory level and rarely tested firsthand. In practice, Agile is learned "on the field", often after attending *ad hoc* seminars. Interest in Agile is however rising, and curricula are being updated to reflect this [436, 265]. An interesting and complete proposal has been advanced by [304], where the "Agile Constructionist Mentoring Methodology" and its year-long implementation in high school is presented. It considers all aspects of software development, with a strong pedagogical support.

In general, CS skills, like programming, are considered a personal skill and taught as such. Not many researchers were challenged by this idea, with few exceptions [84].

We noted however that the approach is hardly systematic, and no general consensus exists on how to proceed along this line.

Generally speaking, the traditional educational paradigm is not tailored to educate people to handle complex issues or *wicked problems* [80]. PISA-like evaluations are meaningless to determine the educational system's efficiency in this respect, since they consider the individual performance of students. So, the gap between students' formal educational background and real life wicked problems and the related complex task becomes larger as the level of predictability decreases and uncertainty increases [381].

Some studies tackled the idea that hard skills expertise should be complemented with soft skills, possibly introducing active and cooperative learning to CS [231]. For example, in [392], a long list of so-called soft skills expertise are paired with various developer's roles. In [84] the problem is well analyzed, but arguably the proposed solution is not comprehensive. [305] presents an example of how to promote cooperation within a software project; however generalizing the proposed scheme seems difficult. There is some field testing of Team-based learning [309] applied to CS courses [280]; as however this approach requires a full and radical change of teaching methodology it is not much widespread.

Although the scientific literature has made a substantial contribution to our understanding of Computational Thinking and Agile Values, knowledge in this area, with respect to Cooperative Thinking, remains fragmented.

We have recently introduced the idea of an overarching competence for team coders [...]. This stream of research is based on several experiments [...]. suggesting that effective coding teamwork in educational environments leads to improved learning outcomes and even to software of better quality. Nevertheless, good teamwork is not sufficient, *per se*, to solve complex tasks - individual problem solving competencies are also needed. In previous works, we found that the best outcomes were provided in cases where both such competences (i.e., teamwork and problem solving skills) were effectively implemented [...]. Recently, these problems have been addressed by theoretical contributions. Indeed, [...] defined a conceptual model for Cooperative Thinking, providing a theoretical support.

Substantial questions remain open, like which is the best way to educate CS students to manage both teaming and software development skills, or the best educational practices to use in this regard. In response to these questions, we proposed a first validation of CooT, based on the conceptual model proposed in [...], and grounding it in empirical evidence.

9.3 Research Model and Hypotheses

Based on the prior discussion, we forward our basic thesis. Future workers will need a new set of skills to be competitive on tomorrow's job market. *Ad hoc* educational curricula need to be developed to prevent skill shortage. Apparently, CT and AV alone are not sufficient to educate students to solve wicked problems [476]. The development of a new overarching competence may lead students to describe, recognize, decompose problems and computationally solve them in teams in a socially sustainable way. This competence, which we named Cooperative Thinking, is not just the sum of the two underlying constructs of CT and AV. We propose to consider it as a social dimension of Computer Science Education.

For the sake of this chapter, we used the definition of Complex Problem Solving to identify the most relevant skills, as suggested by the [477].

This is an exploratory study to assess whenever formalized constructs have a significant relationship with each other. As a SEM study, constructs are grounded in literature or experience [191]. Therefore, we are hypothesizing relationships which have a theoretical explanation but were never assessed, which is an important novelty contribution of this chapter.

In the next subsections we are motivating our hypotheses, supported by [...].

9.3.1 Effect of Computational Thinking on Cooperative Thinking

As explained in Section 9.2, in order to enhance the new construct Cooperative Thinking, some individual Computational Thinking skills need to be developed to interact in a constructive way within the group, to suggest useful insights. Following [486], several frameworks have been proposed to operationalize it in an educational system [469, 468, 470]. The general idea is to train students to think in a computational–friendly way to improve their problem–solving skills. As such, it is a pivotal individual skill–set that any future worker will bring to its team. Team performance is strictly related to quality of its individual assets [35]. Therefore, the quality of the developed CT skills will affect positively future teams performance.

According to this background, we formulate our first hypothesis:

H_1: *Computational Thinking positively influences Cooperative Thinking*

9.3.2 Effect of Agile Values on Cooperative Thinking

While Computational Thinking is the specific skill useful to individuals to solve problems, Agile Values educate people to team together. Agile Values offer a variety of points of view useful to solve difficult or wicked problems. Usually there is no single "best solution" to such problems, but several Pareto-optimal ones, whose value moreover may change over time — as is the case in the field of Science and Business [82].

With particular regard to Software Engineering, the design of a complex system whose requirements are unstable is a typical wicked problem [497]. Satisfying unpredictable customer's expectations and ephemeral requirements is beyond the limit of solvability for any single programmer.

Delivering *valuable* software *on time* has been one of the major efforts of software development methodologies in the last years [132]. Although the definition of "on time" may look clear (since it is related to a deadline), it is strictly correlated to "valuable", which is a more vague definition. With reference to the ISO 25010:2011 standard on software quality, the customer may perceive as valuable aspects related to the Quality in Use dimension. Nevertheless, a software with high Quality in Use but a low e.g., maintainability (which is related to the Product Quality) could not be really defined "valuable". The aspect of maintainability may be related to poor refactoring due to time constraints.

In this (trivial) example, it is clear that value and time are two sides of one coin. Mastering such challenges requires a specific skill–set.

The Agile Manifesto proposed a new perspective on software development, based on values that clashed with the established culture of time, based on multi-level hierarchies, top-down decision making and, in general, accepting the given methods without voicing dissent or criticism [9]. The most significant change invoked by the Agile movement is the paramount relevance assigned to *communication and social interaction*, superseding any internal organizational rigidity, documentation, contracts, roles, and more.

This led to the formalization of important concepts (such as changing requirements, self-organizing teams, personal responsibility, ...) and programming practices (pair programming, test-first development, continuous integration, ...). The Agile approach has proven in several contexts its usefulness, and it is now an established development model and its adoption is steadily growing [289].

Consequently, Agile Values are an important skill–set for Cooperative Thinking, leading to our second hypothesis:

H_2: Agile Values positively influence Cooperative Thinking

9.3.3 Effect of Cooperative Thinking on Complex Problem Solving

As proposed with H_1 & H_2, the construct Cooperative Thinking is mainly explainable with Computation Thinking and Agile Values. Nevertheless, we do not believe that it is just the sum of these constructs. Rather it is a useful proxy to develop further fundamental skills.

The intuition is that some of crucial future skills can not be taught with an old–fashioned curriculum. The most significant future skill for future worker in 2020 is, according to the World Economic Forum, *Complex Problem Solving* [477]. According to its definition it is "Developed capacities used to solve novel, ill-defined problems in complex, real-world settings". In other words, it is another way to define wicked problems.

From a pedagogical perspective we started questioning ourselves how to train our best students to manage wicked problems. With regard to Computational Thinking and Agile Values we realized that, separately, they are not sufficient. CT deals with individual capabilities and is deeply routed in the traditional educational system of "solo" learners. On the other hand, AV *per se*, are not enough to deal with such problems. Good social interaction is a valuable driver but not the asset to solve wicked issues.

The idea of Cooperative Thinking, as defined in Section 9.2, is that of a construct which is able to teach students to tackle Complex Problem Solving as a proxy of wicked problems. Therefore, our last hypothesis is:

H_3: Cooperative Thinking positively influences Complex Problem Solving

The relationships among our three hypotheses can be represented as in Figure9.1.

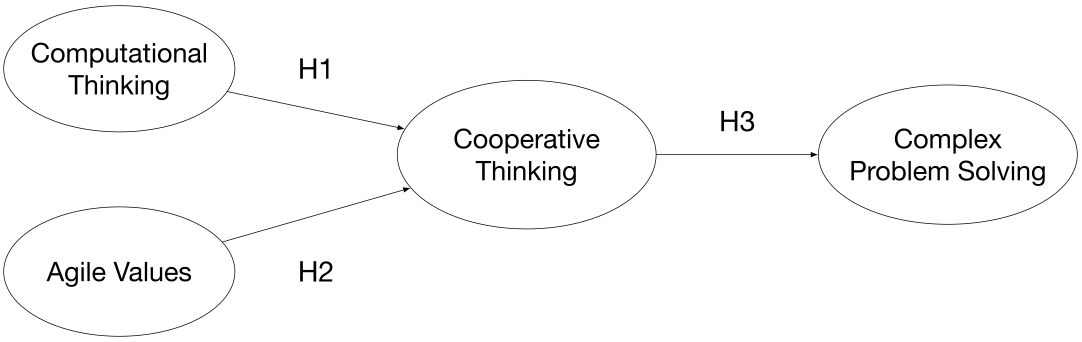


FIGURE 9.1: Theoretical framework and hypotheses

9.4 Research Design

Structural Equation Modeling is strongly influenced by Popper's post-positivist view, according to which social observations should be treated as entities like physical phenomena [369]. The researcher is detached from the observed constructs, since social science inquiry should be objective and hypotheses should be empirically validated to justify them. Typically research outcomes are generalizable, independently from time and context [332].

As this is an exploratory study we are here interested to test the significance of the proposed model. For this reason, post-positivism is the best suited meta-theoretical stance, since we are dealing with the falsification (i.e., significance verification) of our hypotheses. As researchers, we obviously have our epistemological bias, which usually remain hidden or implicit, even if they deeply influence our research [428]. Therefore, empiric (i.e., statistic) procedures are of greatest importance to mitigate researcher's biases [369].

9.4.1 Research Questions

We are interested to find out these two issues: a) is CooT grounded in empirical evidence, and b) does it explain other new constructs? This leads us to our first research question:

- RQ_1: Is *Cooperative Thinking* grounded as a new overarching theoretical construct of Computational Thinking and Agile Values?

Our second research question regards the "explanation" power of our construct:

- RQ_2: Is *Cooperative Thinking* a significant construct, to teach students how to deal with wicked problems?

This research journey let emerge some explicit dimensions which were initially implicit. Thus, beyond the proposal of a new construct which should be considered for curricular purposes, we validate it through a well established statistical method.

9.4.2 Partial Least Square path modeling

The use of PLS–SEM for the validation of latent unobserved variables with multiple observed indicators [93] is an emerging research trend within the Computer Science Education domain [293, 422, 177, 423, 160]. Other research communities have even a longer tradition with PLS–SEM and made several advances for theoretical model testing, in Management [220], Information Systems Research [128] and Organizational Behavior [208]. "SEM has become *de rigueur* in validating instruments and testing linkages between constructs" [169, p. 6], since it allows to distinguish between measurement and structural models, taking also measurement error into account. SEM distinguish itself between two families: the first one are covariance-based techniques (LISREL – CB-SEM); the second one are variance-based techniques i.e., among which partial least squares (PLS) path modeling is the most used one [201]. So, CB-SEM estimates model parameters to minimize the estimated and sample covariance matrices differences; while PLS–SEM estimates model parameters to maximize the variance of endogenous constructs. Therefore, CB assumes multivariate normality with high sample sizes and PLS works with small sample sizes, since it makes no distributional assumptions. Accordingly, model convergence is, in PLS the point at which no

substantial difference happens from one iteration to the next one; while in CB it is the increase or decrease in the function value beyond a certain threshold. The PLS technique is used to test causal relations, maximizing the explained variance of the dependent latent constructs. This enables to exploring cause–effect relations between latent constructs. It offers several advantages compared to CB, beyond those already stressed. Especially for complex models CB seldom converges, especially while dealing with small sample sizes or non-normal data; this is not the case of PLS. Operational research scholars consider PLS as a "silver bullet" for estimating causal models in many theoretical models and empirical data situations [190]. Indeed, it is flexible in the construction of unobserved latent variables and modeling relations among different predictor criteria and variables [92].

9.4.3 Scale Development

As any SEM study, the choice of scale was developed with the greatest care. Following also the example of [293], items were based on the existing literature and measured with a 7-point Likert scale. For any construct some existing frameworks were used to frame the items. In particular, we used for *Computational Thinking* the framework proposed by Computing at School, a subdivision of the British Computer Society [115]. For *Agile Values*, Kent Beck formalized the construct in [43]. *Complex Problem Solving* has been defined by the World Economic Forum in his pivotal report of future skills need [477]. Finally, Cooperative Thinking is the result of our studies about the education of Agile student developers to enhance their Computational Thinking capabilities [...]. The items list with the related literature is presented in A.1.1. Items were developed independently by the authors and refined iteratively until full consensus was reached. After that, a pre-test with five potential-target respondents (i.e., graduate students) was conducted to test the usability of the survey, its rationale, and also the wording. Usability was assessed positively, while minor rationale and wording issues emerged and were consequently fixed.

9.4.4 Data Collection

Firstly, we ran an *a priori* power test [155], to define the minimum sample size for a linear multiple regression F–test, which is a good approximation for a PLS analysis. With an effect size of 15%, and 10% significance, the minimum sample is 82. Then, a stratified convenience sampling technique was used.

To validate the latent variable, grounded in the conceptual model of [...], we used informants which have been already exposed to both Agile practices and Computational Thinking training. This procedure supports the idea that Cooperation Thinking is derived from the combination of AV and CT. Thus, to enhance the construct validity, we targeted students who have been exposed to both Agile practices and CT education along their studies. According to that, the improvement of the reliability of our formative and exogenous constructs enhanced the endogenous and reflective constructs. Strata were designed accordingly, focusing on undergraduate and graduate students of European Universities (missing for review). To enhance the generalizability of the study we also included a significant strata of High School students which were also exposed both to CT and AV during their education. To any subgroup was assigned an ID code for strata definition.

The respondents' rate was 70%, since the survey was directly administered during class by teaching personnel.

	%	#
Population		
Grad. & Undergrad. students	63%	74
High School students	37%	44
Programming experience		
Less than 1 year	8%	10
2-3 years	47%	55
4-6 years	26%	31
7-10 years	4%	5
11-20 years	8%	10
21-35 years	3%	4
More than 35 years	3%	3
Complete software projects		
1	12%	14
2-4	42%	50
5-10	30%	35
11-20	4%	5
20+	11%	13
Agile methods experience		
Daily	12%	14
Used in some projects	43%	51
Did some experiment	18%	21
I studied it	26%	31
Largest team participated in		
0-2	8%	9
3-5	31%	36
6-8	24%	28
9-12	8%	9
13+	9%	11

TABLE 9.1: Demographics

Finally, demographics variables relevant to the context and strata were controlled and represented in Table 9.1. Survey's questions are displayed in A.1.2 for the sake of reproducibility. In total we had 118 respondents, well above the minimum requirement. Undergraduate and Graduate students were 74 (63%), while High School students were 44 (37%). We collected several factors, like the programming experience, completed software projects, Agile method experience, and large team participation to identify sample's skill–set. These are useful indicators for Computational Thinking and Agile Values.

9.5 Results

To compute our model we used Smart PLS 3.0 [390] to estimate the path weighting scheme. Positively, our model converges after 10 iterations. We applied also non-parametric bootstrapping to obtain standard error's estimates [91, 147]. Also Blindfolding was used to calculate Stone-Geisser's Q square value, which represents

	AV	CPS	CT	CooT
AV_3	0,830			
AV_4	0,884			
AV_5	0,655			
CooT_1				0,701
CooT_2				0,693
CooT_3				0,865
CooT_4				0,709
CPS_1		0,801		
CPS_2		0,648		
CPS_3		0,891		
CT_1			0,745	
CT_2			0,756	
CT_5			0,756	
CT_6			0,794	

TABLE 9.2: Outer Loadings

an evaluation criterion for the cross-validated predictive relevance of the PLS path model [170, 439].

9.5.1 Measurement Model

All item loadings above the cut-off value of 0.65 were considered, as represented in Table 9.2, and were significant at $p<0{,}001$ (with the only exception of CPS with a $p<0{,}05$, since it is the highest construct). Following [191], those items below the cut-off value were rejected (i.e., AV 1, AV 2, AV 6, AV 7, CT 3, CT 4, CooT 5). The good average of items loading and a narrower range of difference for such an exploratory study, provide an adequate base for the items in measuring the underlying construct [191]. Items are not redundant, since the outer variance inflation factor (VIF) ranges between 1,165 and 1,832, well below the cut-off value of 5 [191]. Thus, we conclude to have an appropriate item reliability.

The construct reliability and validity is composed by the reliability of constructs, composite reliability and average variance extracted (AVE) [159]. To assess the construct reliability we used Cronbach's alpha, which measures the homogeneity of items in a construct based on the assumption that each item in the scale contributes equally to the latent construct. The composite reliability depends on the item loadings estimated in the measurement model to compute the measure of internal consistency [481]. According to [343] both Cronbach's alpha and composite reliability should have at least a value of 0,70 to be acceptable. Rho_a is another reliability measure developed by [131], according to which the most conservative critical value should be above 0,7. For AVE a value above 0,5 is desirable, since it reflects the variance captured by indicators. If this is the case, it means that the variance captured by indicators is greater than the measurement errors. As we can see in Table 9.3, all literature's requirements are met, also for formative constructs, which is rather uncommon [227].

We assess discriminant validity to analyze the relationships between latent variables with both Fornell–Lacker Criterion and Heterotrait–Monotrait Ratio of Correlations (HTMT) [389]. According to the Fornell–Lacker Criterion the square root of AVE must be greater than the correlation of the construct with all other constructs in the structural model [159]. In this way we can see if constructs do not share the same

Constructs	Cronbach's Alpha	rho_A	Composite Reliability	AVE
CooT	0,734	0,775	0,832	0,556
CT	0,762	0,763	0,848	0,582
CPS	0,721	0,904	0,827	0,618
AV	0,704	0,740	0,836	0,633

TABLE 9.3: Construct Reliability and Validity

	AV	CPS	CT	CooT
AV	0,796			
CPS	0,354	0,786		
CT	0,331	0,612	0,763	
CooT	0,570	0,285	0,397	0,745

TABLE 9.4: Fornell–Lacker Criterion

type of items and are so conceptually different from each other. As shown in Table 9.4, the lowest square root of AVE is 0,745 (CooT–CooT), which is greater than the highest correlation value of 0,612 (CPS–CT). With regard to HTMT, all values are below the most conservative threshold of 0,85 [200], as shown in Table 9.5.

Finally, we can conclude that the measurement model provides evidence of adequate reliability and validity for both reflective and formative constructs.

9.5.2 Structural Model

After the test of our measurement model we assess the validity and exploratory power of the structural model.

The first step is to test whenever the inner variance inflation factor values (VIF) are below the threshold value of 5 to discard redundant inner–model constructs [191]. We see that those values are between 1 and 1,23 so well below the critical value.

After that we measure path's significance through biased–corrected and accelerated bootstrapping. Since it is an exploratory study we assumed a two–tailed test with a significance level of 10%. As we can see from Table 9.6, all indicators comply with their respective critical values. In particular T–statistics are above 1,96 for all paths and the p–values are below the reference level of 0,1 (for 10% significance) and also below the more conservative value of 0,1 [191]. Therefore, we can conclude that all paths in the model are significant. This supports all our three hypothesis H_1, H_2, H_3.

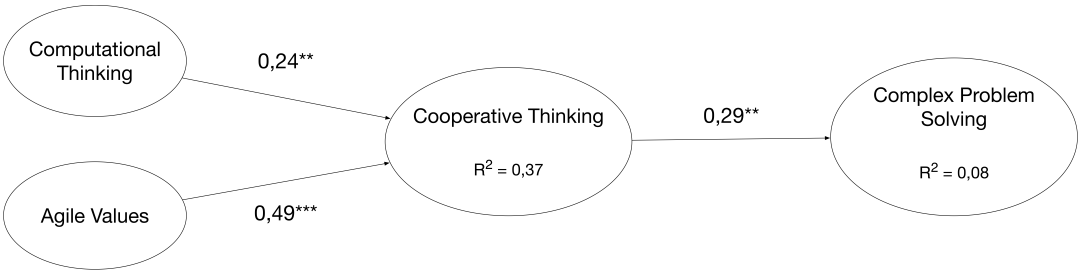
Passing now to the evaluation of the R-square values of the two endogenous variables we see that Computational Thinking and Agile Values explains very well the construct Coopertative Thinking with a value of 0,374. Interestingly, Complex Problem Solving has a relative low R-square value of 0,081 for several reasons [191]. The

	AV	CPS	CT	CooT
AV				
CPS	0,444			
CT	0,456	0,803		
CooT	0,764	0,340	0,513	

TABLE 9.5: Heterotrait-Monotrait Ratio of Correlations (HTMT)

Paths	Orig. Sample	Mean	St. Dev.	*T*	*p*
AV->CooT	0,492	0,498	0,066	7,474	0,000
CT->CooT	0,235	0,249	0,085	2,770	0,006
CooT->CPS	0,285	0,288	0,134	2,127	0,034

TABLE 9.6: Paths Coefficients



*** p<0,001, ** p<0,05, * p<0,1

FIGURE 9.2: Structural model with Path coefficients and p values

first one is statistic. Since CPS is derived by another endogenous construct, the statistical explanation power is mitigated by the mid-construct CooT. So, it is obvious to have a relatively lower value. The second is conceptual. Although CooT is a useful proxy for wicked problems, since the p-value is significant, CPS may not be enough. We should go back to the literature to define more constructs related to wicked problems. Doing so, we would have several constructs which are well explained by Cooperative Thinking.

To confirm those findings we look now at the f-square values. This metrics indicates how well each exogenous construct explains the endogenous ones. Here we have that the relationship AV->CooT has the highest value of 0,35 which suggests a very high effect, according to literature standards [191]. The relationship CT->CooT has a moderate effect, but still significant since it is above the threshold of 0,02 [191], with a value of 0,08. The same for the the relationship CooT->CPS with a value of 0,09.

Now we to test the predictive validity of the model, to see if the exogenous constructs explains significantly the endogenous ones [13]. To do so, we use run blindfolding with an omission distance of 7 to measure the Stone–Geisser's Q-square through Construct Crossvalidated Redundancy [439, 170]. Here, the Q-square should be bigger than 0 [191]. We have for both for CooT (Q^2: 0,177) and CPS (Q^2: 0,029) the match of this criterion.

Overall, we can conclude that our structural model, represented in Figure 9.2, is significant and predicts all tested constructs. Nevertheless, the low R^2 of CPS indicates that the model is not fully complete. Still, it is significant and is a solid ground to build new theory on. Therefore, we also conclude that CooT is a significant proxy to teach wicked problems. Significance and explanatory values of CooT suggest that AV and CT are good explanatory construct on which to build this new skill.

9.6 Discussion

Our structural model suggests a positive answer for both our Research Questions, according to these statistical considerations:

- RQ_1: the high R^2 of CooT indicates a high explanatory power of the new construct. This means that both CT and AV are significant components of this new overarching construct. Moreover, path coefficients of H_1 and H_2 are highly significant. So, they influence the new construct in a statistically significant way.

- RQ_2: CooT does explain in a significant way CPS, due to its path coefficient. Also H_3 is significant, considering path's p- and absolute value. Since the R^2 is not a relevant indicator in this case, for the above–mentioned reasons, we can state that it does well explain CPS, so wicked problems.

From this evidence we can conclude that both CT and AV are building constructs of CooT, which is able to explain independently a new construct, namely Complex Problem Solving.

Consistently with our research design, we outline now the educational implications of this study and its limitations.

9.6.1 Implications

Our findings support the idea that CT and AV reinforce each other to sustain the new construct of Cooperative Thinking. Now we outline some educational practices that could be included in current curricula to foster Cooperative Thinking.

These activities are all linked to the various components that constitute the Cooperative Thinking construct. All the proposed practices are deeply grounded in the pedagogical literature. Cooperative Thinking can be operationalized through established educational practices. The educational scope is to tackle key concepts of problem description, recognition, decomposition, in order to solve them computationally in teams, stressing social sustainability. This reinforces the theoretical ground of this construct, since it is both backed in literature and is empirically significant. Practices, as such, are not new; new is the educational scope we are interested in. Since the educational goal is to teach students to deal with complex and ill-defined problems for their future working life, a gradual approach should be introduced, according to the education grade. Students should experience in an incremental way wicked problems to learn useful patterns to reuse for future problems.

We propose the following categories of practices to foster CooT in everyday activities:

- **Complex Negotiation**: Students should be able to propose, discuss, and evaluate ideas and solutions, considering different point of views on stratified topics. Deriving from Agile negotiation [43] and negotiation pedagogy [25], this skill aims to develop adequate capabilities to deal with stratified issues and different opinions. Finding a group-wise sustainable way (i.e., Pareto-optimal) to devise a solution of a problem, taking into consideration a variety of useful (and useless) points of view is the aim of this skill. Key activities include: structured Brainstorming, Architectural design and code contests, Randoris and Code retreats [412].

- **Continuous Learning**: Both individuals and their groups should be ready to develop the skills needed to solve a given problem at hand. Education should be centered on enhancing the students' ability related to "reflection–in–action" [420], practicing continued learning and problem solving throughout their entire career. Activities such as Peer Learning and Exploratory learning are well suited to this task.

- **Group Awareness**: This indicates the capability to be part of a group. It covers knowledge and perception of behavioral, cognitive, and social context information within a group [60]. It requires reflective activities (such as [264] Lego Serious Play) and group games in order to develop a "team spirit" and promote the self-organizing skill of the team.

- **Group Organization**: It refers to the ability to develop software as a group, i.e. deliver a working product collaboratively. This goal can be achieved by regularly applying Group-oriented Project-based learning, starting with small, toy project and scaling to complex ones. It is grounded within the domain of peer learning, to generate productive instructional dialogues for joint problem solving, relying on intrinsic rather than extrinsic rewards, discouraging competition between students [120].

- **Social Adaptability**: It refers to the skill needed to handle groups' internal and external social dynamics. Especially for adolescents this kind of competence is a pivotal aspect of education; it will determine how future adults will be oriented to express social sensitivity [6]. Activities include role play, group exercises, project simulations, and even stress tests, as in [269].

These activities, collectively, address all aspects of a meaningful learning model [217], and promote Cooperative Thinking.

9.6.2 Limitations

As inherent of any scientific method [491], this study has several limitations.

The first issue is about the use of cross–sectional data (i.e., observation of the population through data collection of many subjects at the same point of time) for the empirical assessment of the model. Hence, results may reflect associations rather than causality between constructs. Moreover, it is not possible to predict if the causal relationship will change over time. However, a longitudinal study may overcome this limitation. Generally speaking, we tackled these issues through a sound theoretical derivation, which is the best way to minimize these limitations [191].

Secondly, we measured our constructs from a subjective perspective through a single–informant approach. So, the constructs represent students' perspective. Respondents may not have answered the question accurately or with some biases. For this reason the survey was anonymous and no grades were assigned for the participation at this research. Moreover, a sample size of 118 observation through different European environments minimized the common method bias [254].

Third, we used perceptual measures, rather than objective ones, asking students to state their level of agreement on literature–derived items. So, the measurements may not fully reflect the real world accurately due to potential respondent bias and random errors. Therefore, items were adapted from previous studies and literature and subject to various examinations for ensuring their quality. However, continuous item development and validation is needed to update the constructs.

Finally, the last limitation regards the sampling technique. We used a stratified convenience sampling technique, were strata were defined accordingly to the acquired skill–set. Students were already exposed to both CT and AV exogenous constructs along their educational career, in order to assess the level of endogeneity of CooT and CPS. In doing so, we asked European partner Universities we already collaborate with to administer the survey. Those Universities adopted curricula that fostered CT and AV and were therefore considered suitable targets for our strata definition. Our

research did not focus non–European educational environments; this may weaken the generalizability of our results, since the empirical results might not fully represent the constructs elsewhere. Cultural factors may have also played a role, which we did not considerate in this study. Generally, non-responses may have lead to sample selection bias if a systematic and unobservable difference exists between respondents and non-respondents [483].

All in all, we consider our limitations acceptable for this exploratory study, especially because we took several precautions to minimize them. As validated in Section 9.5, all statistical indicators suggest the conceptual validity of the model. Still, we are aware that this is a starting point, not an ending one; further research is needed to generalize the model and to better define its sub–dimensions.

9.7 Conclusions

With this chapter we validated the theoretical model of Cooperative Thinking to train teams of students to manage Computer Science problems. Computational Thinking and Agile Values are the pillars on which Cooperative Thinking is built. Nevertheless, it is not just the sum of these constructs. It is a new one, which educational curricula should deal with. In fact, Cooperative Thinking is able to tackle significantly Complex Problem Solving, which we used as a proxy construct of wicked problems.

To validate the proposed educational model we used Structural Equation modeling with Partial Least Squares. Exploiting this technique we were able to test the statistical significance of the relationships between constructs as also their explanatory power. Indeed, PLS–SEM has important potentials in Computer Science to test the significance of theoretical social constructs.

This study provided a model for our future empirical investigations on the new educational construct. Our future work will focus on both theoretical and pedagogical aspects.

Some generalization efforts need to be undertaken to consider Cooperative Thinking as a real universal competence. This study could be administered also in non-European countries. To uncover unobserved heterogeneity in the inner (structural) model a Finite Mixture Partial Least Squares (FIMIX-PLS) segmentation test should be run [189]. This will capture heterogeneity by estimating the probabilities of segment memberships for each observation and simultaneously estimate the path coefficients of all segments. Doing so, an improved understanding of constructs performance on different segments (i.e., groups of students) is possible. Thus, it is possible to tailor educational curricula, according to each segments' sensibility, according to respective differences (e.g., performance, culture, gender, age, students' level). Moreover, this can be supported by finer granular studies, based on students' composition, to analyze those pedagogical differences. Literature work is further needed to refine measurements and sub–dimensions of all constructs. We used Complex Problem Solving as proxy of wicked problem. However, this assumption needs further insights to be validated. In a possible extension of the model, wicked problems may be represented by other parent–constructs of CPS to make their representation more trustworthy.

From a pedagogical perspective, Cooperative Thinking practices and educational curricula need to be outlined in more depth respect to what we did in this chapter. Indeed, it is possible that an *ad hoc* curriculum on Cooperative Thinking will help students to improve model fitting. For instance, developing the proposed constructs of Complex Negotiation, Continuous Learning, Group Awareness, and Group Organization.

Chapter 10

Concluding Remarks

10.1 Discussion

There are many ways to engineer software in a sustainable fashion. Moreover, no one can pretend to have the 'silver bullet' to address such challenge.

Sustainability, intended as a socio-technical perspective over the software development and maintenance cycle, means to align technological solutions to the social context. It means to manage a variety of units of analysis of both technical (which is the most suited design, design patterns, coding standards, language, etc.) and social nature (coordination among developers, meeting organization's needs, usability aspects by end users, etc.). It is very hard to manage simultaneously these units of analysis, since they are quite heterogeneous and they might not always apply to each context. Again, there is no single solution to the problem. Rather, each IT manager should consider which are the most relevant sustainability issue for the specific organization and set accordingly Key Performance Indicators (KPIs).

What we provided with this work is a model to tackle socio-technical aspects when engineering software. At a high level, we identified three key factors to develop and maintain software in a sustainable way, namely Quality, Architecture, and Process. We do not advance a model, intended as a new standard, which practitioners should stick to. Fairly, we advise to consider these three key factors when engineering sustainable software. Indeed, as we have shown, when one of these factors is missing in the managerial process, and they emerge rather by chance (or in a random way), this leads unsustainable IT systems. The way the SQuAP model should be customized still relies on IT managers, since they have the full overview of the social context, as on technical specifications.

Nevertheless, we formalized the high-level SQuAP model into an OWL ontology, intended to serve as an assessment tool for practitioners. Similarly, a relevant Knowledge Engineering case study shows how Quality, Architecture, and Process aspects have been considerate to deliver mission-critical software. More administrative and legal aspects have also been analyzed, since they are daily concerns of IT managers, aiming to deliver sustainable systems. Managing Socio–Technical Software Engineering is a complex task, since it deals with heterogeneous units of analysis. Thus, we should also consider educational aspects, to train properly the next generation of IT professionals. Assuming that the degree of complexity will raise at a steady state along with software's pervasiveness, education will become also a key point to engineer software in a sustainable fashion. Accordingly, we found of pivotal importance to address also the way students should be trained to work in a cooperative way, understanding the surrounding environment. Hence, we proposed the Cooperative Thinking paradigm.

Surely, there are several ways to engineer software in a context-aware sustainable fashion. However, we believe to have provided both a high-theoretical, and

low-implementation perspectives on our MRQ, namely how to engineer software in a sustainable fashion. Nonetheless, we suggest that any possible different approach can be mapped in our model, at least at a high-level.

10.2 Conclusion

This dissertation analyzed several aspects of Socio–Technical Software Engineering. In particular, we elaborated on the idea of sustainability, intended as a context-aware management of software, covering both social and technical units of analysis; where, with management we meant the act of control, decision, and organization over the software development process.

Our aim was not to provided *ad hoc* solutions to single problems, rather we wanted to represent the debate within the community through a scoping study and address compelling practitioners concerns met during our research journey. As a result, we developed a model for sustainable information systems and formalized it as an OWL ontology.

Keeping high software quality for rapidly changing mission-critical requirements is a typical Socio–Technical Software Engineering problem. Thus, we described a real world case study where we tackled this issue through a Knowledge Engineering approach. This case study is exemplary of a typical situation where multiple mental models both of developers and different stakeholders have been aligned in a context of high volatility, where quality and security were a primary concern.

Similarly, the legal nature of cooperation has been considered. Law & Economics aspects were analyzed to align divergent interests of the different actors. Developers, as any human beings, are driven by incentives to be better-off. Similarly, this is the same behavior of organizations which hire developers or software houses to build their systems. Since the interests of these two actors are typically opposed, contracts are a working solution to mitigate such divergent views. For this reason, IT managers may greatly benefit from our work, which has been evaluated in a real world context, to set up collaborations which upholds both developers' interests and systems' quality and integrity. In addition to this work, we also provided relevant managerial insights related to software cloning and Intellectual Propriety Rights in general. In fact, to add new functionalities, a good idea is to integrate existing libraries or packages or reuse well-working pre-existing code. As already stated, the average reuse of code in large software systems is around 20%-30%, up to 50%. It is considered a good coding practice, but it also generates new dependencies which have to be properly managed. Nevertheless, our goal was to address questions like, on what extend may I reuse some others code? Which type of cloning is legally accepted? Can I reverse engineer some others code? In particular, we did not consider what the Software Engineering community thinks about those crucial issues but courts in the US and Europe. Ethical opinions are not relevant in a court trial. Therefore, we surveyed courts' orientation through the existing case law. This should help both IT managers and developers to adapt their IPR strategies. Interestingly, we found two different positions from European Union's courts and that of the United States. In the US, courts have a waiving attitude and decide over software's copyright protection case by case. While the European Court of Justice appears more liberal in terms of degree of legal protection of software's copyright, stating that cloning of "principles" or "ideas" (semantic clones) can not be an infringement of copyright, since "principles" or "ideas" are not copyrightable.

Finally, we studied education aspects of cooperation, which we consider pivotal to incrementally address socio-technical problems. The main insight is that if developers are not supported to acquire collaboration skills, they will be less prone to collaborate. Conversely, to improve socio-technical congruence, a cooperation-based development approach, named Cooperative Thinking is a viable solution. Accordingly, we developed both a theoretical and pedagogical framework which can be used by instructors to develop Cooperative Thinking skills. We grounded Cooperative Thinking from our previous contributions through a research synthesis and systematized it in the framework of Kolb's learning style inventory. Moreover, we validated the construct of Cooperative Thinking along with relevant teaching practices with Partial Least Squares Structural Equation modeling.

To conclude, our investigation did not provide a single solution to Socio–Technical Software Engineering problems. Rather, we proposed a perspective on Socio–Technical Software Engineering, which focuses on Quality, Architecture, and Process. We found out, and discussed it along the chapters, that engineering software, taking care of socio-technical aspects, means to focus on Quality–Architecture–Process aspects; and more in particular on their relations and connections. Without a view on these three elements it is likely that socio-technical congruence will dramatically decrease, with gloomy scenarios on the maintenance and evolution on such systems. This idea emerged in a clear way in Chapter 3, where substantial software quality concerns emerged due to loosely congruence of the different modules and poor attention on Quality–Architecture–Process issues.

Quality, impacts on the code. Poor software quality means to harm future maintenance and evolution of your system. It depends both on the way people work and on the design of such system.

Architecture, impacts on the design of the system. Consciously or unconsciously omitted architectural choices drives to tremendous system's layering with poor future maintenance and evolution capabilities. It is caused by poor software quality, which does not consider dependencies, and by people, who do not coordinate properly to hinder it.

Process, impacts on the way people build a system. If people do not cooperate with stakeholders to deliver valuable functionalities to the business, or do not coordinate among them selves to develop a coherent and consistent system. It is related to cooperation and code ownership practices; thus, effective communication among the different actor will be reflected in the system's design at an agreed quality level.

Building sustainable software means to manage these three key factors.

10.3 Future works

In this research area, there are several future directions. We will focus on two of them, which are the focus of this investigation: socio-technical aspects of Software Engineering, and the transferability of the Quality–Architecture–Process paradigm.

The first open-ended research journey will rely on interdisciplinary and cross-fertilization from Software Engineering areas, as well as others (e.g., psychology, management, engineering). It is a growing debate, where different contributions are needed to support sustainable Software Engineering practices. At the end of each chapter, some of these directions have been outlined.

The second direction we see regards the transferability of the model. We are interested to see if our model can be applicable to other contexts, or domains. If this model seems to work well in the Software Engineering domain, also other related

disciplines could benefit from this perspective. Potentially, any domain which relies socio-technical contributions is a target. Indeed, when there is a process to manage, a structure to implement, which works only if a certain quality is provided, the model could work. Data Science could be an example. It is a new and rising discipline which involves all three key factors. Data quality is highest priority, since bad data quality could highly biases results. The value extraction process need also to be properly manage by a process. Finally, the architecture of the most suitable model to respond to Initial Hypotheses have to be implemented.

In sum, this dissertation supported the vision that most software problems are of socio-technical nature. Although we narrowed our research to the Software Engineering domain, other research areas could benefit from this perspective.

Appendix A

Research Materials

A.1 Questionnaire

A.1.1 Constructs

Construct [source]	*Labels*	*Items*	*Questions*
Cooperative Thinking [311]	COOT_1	Complex negotiation	During design, I like to discuss with people who have different ideas, in order to develop the best solution.
	COOT_2	Continuous learning	Programming in team taught me something I didn't know.
	COOT_3	Group awareness	I like to be part of a software developing team.
	COOT_4	Group organization	When I work in team, results are better than when I work alone.
	COOT_5	Social sensitivity	During development, I work fine even with teammates with whom I have personal difficulties.
Agile Values [43]	AV_1	Timeboxing and estimation	I can estimate precisely the time needed to complete a developing task.
	AV_2	Simple solution	It is important to find a solution, regardless how, also if not generally applicable.
	AV_3	Programming practices	I use Agile practices during software development
	AV_4	Agile SDLC	I prefer Agile methods to traditional ones.
	AV_5	On-site customer	When I work in a team, I join frequently conversations with teammates or clients/stakeholders.
	AV_6	Face-to-face communication	I prefer direct, face-to-face, communication to emails or messages.
	AV_7	Courage	During a discussion with teammates, I am able to well defend my point of view.
Computational Thinking [115]	CT_1	Logical reasoning	I get good results in logical-mathematical tests and exercises.
	CT_2	Algorithmic thinking	I can usually decompose a problem in precise and sequential steps.
	CT_3	Generalization	I discard details not essential to solving design problems.
	CT_4	Evaluation	I like to modify a working solution to improve it, even risking to waste a lot of time.
	CT_5	Patterns	I can easily identify and evaluate recurring patterns or behaviors.
	CT_6	Decomposition	I can always decompose a complex Problem into simpler ones.
Complex Problem Solving [477]	CPS_1	Curiosity	I'm good at working on problems I never tackled before.
	CPS_2	Creativity	I can solve ill-defined problems.
	CPS_3	Tenacity	I like to solve real, complex problems.

Table A.1: Items list

Question	Type	Options
Institution name	Open	
Occupation	Closed	High School Student, University Student
Programming Experience	Closed	1y, 2-3y, 4-6y, 7-10y, 11-20y, 21-35y, 35+y
Completed SW Projects	Closed	0,1,2-4, 5-10, 11-20, 20+
Agile Experience	Closed	None, Studied, Experimented, Some Project, Daily Use
Largest team size participated in	Open	

Table A.2: Demographics questionnaire

A.1.2 Demographic Information Questionnaire

We present the questions used to assess the answering population, along with answer choices (if present).

Bibliography

[1] meta . *Oxford Dictionary Of English*. 3th. Oxford University Press, 2013.

[2] European Parliament and of the Council. *Directive 2007/64/EC*. http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02007L0064-20091207. 2007.

[3] European Parliament and of the Council. *Directive 2015/2366/EU*. http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32015L2366. 2015.

[4] G. Abowd et al. *Recommended Best Industrial Practice for Software Architecture Evaluation*. Tech. rep. Software Engineering Institute, 1997.

[5] A. Abualkishik et al. "A study on the statistical convertibility of IFPUG Function Point, COSMIC Function Point and Simple Function Point". In: *Information and Software Technology* 86 (2017), pp. 1–19.

[6] G. Adams. "Social competence during adolescence: Social sensitivity, locus of control, empathy, and peer popularity". In: *Journal of Youth and Adolescence* 12.3 (1983), pp. 203–211.

[7] P. Adler and R. Cole. *Designed for learning: A tale of two auto plants*. Sloan Management Review, 1995.

[8] A. T. M. Aerts et al. "Architectures in context: on the evolution of business, application software, and ICT platform architectures". In: *Information & Management* 41.6 (2004), pp. 781–794.

[9] AgileAlliance. *Agile manifesto*. 2001.

[10] N. Ahmad, A. Rextin, and U. Kulsoom. "Perspectives on usability guidelines for smartphone applications: An empirical investigation and systematic literature review". In: *Information and Software Technology* 94 (2018), pp. 130–149.

[11] S. Aier, B. Gleichauf, and R. Winter. "Understanding Enterprise Architecture Management Design – An Empirical Analysis". In: *Proc. 10th Int. Conf. Wirtschaftsinformatik*. 2011.

[12] G. Akerlof. "The market for "lemons": quality uncertainty and the market mechanism". In: *The Quarterly Journal of Economics* (1970), pp. 488–500.

[13] S. Akter, J. D'Ambra, and P. Ray. "An evaluation of PLS based complex models: the roles of power analysis, predictive relevance and GoF index". In: *Proceedings of the Americas Conference on Information Systems*. 2011, pp. 1–7.

[14] B. Al-Ani et al. "Continuous coordination within the context of cooperative and human aspects of software engineering". In: *International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. ACM. 2008, pp. 1–4.

[15] D. S. Alberts, J. J. Garstka, and F. P. Stein. *Network Centric Warfare: Developing and Leveraging Information Superiority*. Tech. rep. DTIC Document, 2000.

[16] A. Albrecht and J. Gaffney. "Software function, source lines of code, and development effort prediction: a software science validation". In: *IEEE Transactions on Software Engineering* 9.6 (1983), pp. 639–648.

[17] J. Allert. "Learning style and factors contributing to success in an introductory computer science course". In: *Advanced Learning Technologies, 2004. Proceedings. IEEE International Conference on*. IEEE. 2004, pp. 385–389.

[18] T. Amabile, C. Fisher, and J. Pillemer. "IDEO?s Culture of Helping". In: *Harvard Business Review* 92.1-2 (2014), pp. 54–61.

[19] G. Antunes et al. "Using ontologies to integrate multiple enterprise architecture domains". In: *International Conference on Business Information Systems*. Springer. 2013, pp. 61–72.

[20] A. April and F. Coallier. "Q. Bell Canada, "Trillium: A model for the assessment of Telecom software system development and maintenance capability"". In: *Proc. Software Engineering Standards Sym.* 1995.

[21] C. Arumugam and B. Kaliamourthy. "Global Software development: An approach to design and evaluate the risk factors for global practitioners." In: *International Conference on Software Engineering and Knowledge Engineering*. 2016, pp. 565–568.

[22] R. Atkinson. "Project management: cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria". In: *International Journal of Project Management* 17.6 (1999), pp. 337–342.

[23] S. Atkinson and G. Benefield. "Software Development: Why the Traditional Contract Model Is Not Fit for Purpose". In: *Proc. HICSS46, Software Track*. Hawaii: IEEE Computer Society Press, 2013, pp. 330–339.

[24] D. Avison, R. Baskerville, and M. Myers. "Controlling action research projects". In: *Information Technology & People* 14.1 (2001), pp. 28–45.

[25] K. Avruch. "Culture and negotiation pedagogy". In: *Negotiation Journal* 16.4 (2000), pp. 339–346.

[26] M. A. Babar, A. W. Brown, and I. Mistrík. *Agile Software Architecture: Aligning Agile Processes and Software Architectures*. Newnes, 2013.

[27] M. A. Babar, L. Zhu, and R. Jeffery. "A framework for classifying and comparing software architecture evaluation methods". In: *Proc. Australian Software Engineering Conference*. IEEE. 2004, pp. 309–318.

[28] R. Badham, C. Clegg, and T. Wall. "Socio-technical theory". In: *Handbook of Ergonomics*. Wiley, 2000.

[29] B. S. Baker. "On Finding Duplication and Near-duplication in Large Software Systems". In: *Proceedings of the Second Working Conference on Reverse Engineering*. WCRE '95. Washington, DC, USA: IEEE Computer Society, 1995, pp. 86–95. ISBN: 0-8186-7111-4.

[30] L. Balasubramanian and E. Mnkandla. "An evaluation to determine the extent and level of Agile Software Development Methodology adoption and implementation in the Botswana Software Development Industry". In: *Advances in Computing and Communication Engineering (ICACCE), 2016 International Conference on*. IEEE. 2016, pp. 320–325.

[31] V. Balijepally, S. Nerur, and R. Mahapatra. "IT value of software development: A multi-theoretic perspective". In: *Americas Conference on Information Systems*. IGI Global, 2009, pp. 96–110.

[32] R. D. Banker, I. Bardhan, and O. Asdemir. "Understanding the impact of collaboration software on product design and development". In: *Information Systems Research* 17.4 (2006), pp. 352–373.

[33] H. Baraki et al. "Interdisciplinary design patterns for socially aware computing". In: *International Conference on Software Engineering (ICSE)*. Vol. 2. IEEE. 2015, pp. 477–486.

[34] V. Barr and C. Stephenson. "Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community?" In: *ACM Inroads* 2.1 (2011), pp. 48–54.

[35] M. R Barrick et al. "Relating member ability and personality to work-team processes and team effectiveness." In: *Journal of Applied Psychology* 83.3 (1998), pp. 377–391.

[36] V. R. Basili. *Software modeling and measurement: the Goal/Question/Metric paradigm*. Tech. rep. 1992.

[37] R. Baskerville and M. D. Myers. "Special issue on action research in information systems: Making IS research relevant to practice: Foreword". In: *MIS Quarterly* (2004), pp. 329–335.

[38] L. Bass, P. Clemens, and R. Kazman. *Software Architecture in Practice*. 3rd ed. Addison-Wesley, 2012.

[39] L. Bass, I. Weber, and L. Zhu. *DevOps: A Software Architect's Perspective*. Addison-Wesley, 2015.

[40] G. Baxter and I. Sommerville. "Socio-technical systems: From design methods to systems engineering". In: *Interacting with Computers* 23.1 (2011), pp. 4–17.

[41] J. Bearden. *Command and Control Enabling the Expeditionary Aerospace Force*. Tech. rep. DTIC Document, 2000.

[42] K. Beck. *Test Driven Development By Example*. Addison-Wesley, Boston, 2003.

[43] K. Beck and C. Andres. *Extreme programming explained: embrace change - 2nd edition*. Addison–Wesley, 2004.

[44] S. Bellomo, I. Gorton, and R. Kazman. "Toward agile architecture: Insights from 15 years of ATAM data". In: *IEEE Software* 32.5 (2015), pp. 38–45.

[45] B. Bendik, P. Nielsen, and B. Munkvold. "Four integration patterns: IS development as stepwise adaptation of technology and organisation". In: *European Conference of Information Systems*. 2005, pp. 26–29.

[46] L. Benedicenti et al. "Applying scrum to the army: a case study". In: *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM. 2016, pp. 725–727.

[47] P. Bengtsson et al. "Architecture-level modifiability analysis (ALMA)". In: *Journal of Systems and Software* 69.1 (2004), pp. 129–147.

[48] A. Benlian and I. Haffke. "Does mutuality matter? Examining the bilateral nature and effects of CEO–CIO mutual understanding". In: *The Journal of Strategic Information Systems* 25.2 (2016), pp. 104–126.

[49] J. A. Bergstra and P. Klint. "About "trivial" software patents: The IsNot case". In: *Science of Computer Programming* 64.3 (2007), pp. 264–285. ISSN: 0167-6423.

[50] S. Berman. "Digital transformation: opportunities to create new business models". In: *Strategy & Leadership* 40.2 (2012), pp. 16–24.

[51] S. Betz et al. "An evolutionary perspective on socio-technical congruence: The rubber band effect". In: *International Workshop on Replication in Empirical Software Engineering Research (RESER)*. IEEE. 2013, pp. 15–24.

[52] N. Bicocchi, D. Fontana, and F. Zambonelli. "A self-aware, reconfigurable architecture for context awareness". In: *Symposium on Computers and Communication (ISCC)*. IEEE. 2014, pp. 1–7.

[53] I. Bider and H. Otto. "Modeling a global software development project as a complex socio-technical system to facilitate risk management and improve the project structure". In: *International Conference on Global Software Engineering (ICGSE)*. IEEE. 2015, pp. 1–12.

[54] I. Bider and O. Söderberg. "Becoming Agile in a Non-disruptive Way-Is It Possible?" In: *International Conference on Enterprise Information Systems*. 2016, pp. 294–305.

[55] C. Bird et al. "Empirical software engineering at microsoft research". In: *Conference on Computer Supported Cooperative Work*. ACM. 2011, pp. 143–150.

[56] C. Bird et al. "Putting it all together: Using socio-technical networks to predict failures". In: *Proc. International Symposium on Software Reliability Engineering*. IEEE. 2009, pp. 109–119.

[57] A. Blackwell, L. L. Church, and T. R. T. Green. "The abstract is 'an enemy': Alternative perspectives to Computational Thinking". In: *Proc. 20th Annual Workshop of the Psychology of Programming Interest Group*. Vol. 8. 2008, pp. 34–43.

[58] E. Blomqvist et al. "Experimenting with eXtreme Design". In: (Lisbon, Portugal). Springer, 2010, pp. 120–134.

[59] S. C Blumenthal. *Management information systems; a framework for planning and development*. Tech. rep. 1969.

[60] D. Bodemer and J. Dehler. "Group awareness in CSCL environments". In: *Computers in Human Behavior* 27.3 (2011), pp. 1043–1045.

[61] B. Boehm. "A spiral model of software development and enhancement". In: *IEEE Computer* 21.5 (1988), pp. 61–72.

[62] B. Boehm, C. Abts, and S. Chulani. "Software development cost estimation approaches–A survey". In: *Annals of Software Engineering* 10.1-4 (2000), pp. 177–205.

[63] B. Boehm and V. R. Basili. "Software Defect Reduction Top 10 List". In: *Computer* 34.1 (2001), pp. 135–137.

[64] B. Boehm, J. R. Brown, and M. Lipow. "Quantitative evaluation of software quality". In: *Proc. 2nd Int. Conf. on Software Engineering (ICSE)*. ACM/IEEE. 1976, pp. 592–605.

[65] W. Boh and D. Yellin. "Using enterprise architecture standards in managing information technology". In: *Journal of Management Information Systems* 23.3 (2006), pp. 163–207.

[66] M. Book, V. Gruhn, and R. Striemer. "adVANTAGE: A fair pricing model for agile software development contracting". In: *Agile Processes in Software Engineering and Extreme Programming*. Ed. by C. Wohlin. Malmo, Sweden: Springer, 2012, pp. 193–200.

[67] M. Book, V. Gruhn, and R. Striemer. *Tamed Agility*. Springer, 2016.

[68] A. Borici et al. "Proxiscientia: Toward real-time visualization of task and developer dependencies in collaborating software development teams". In: *International Workshop on Co-operative and Human Aspects of Software Engineering*. IEEE Press. 2012, pp. 5–11.

[69] V. Boucharas et al. "The contribution of enterprise architecture to the achievement of organizational goals: Establishing the enterprise architecture benefits framework". In: *Department of Information and Computing Sciences, Utrecht University, Utrecht* (2010).

[70] M. Bourimi et al. "AFFINE for enforcing earlier consideration of NFRs and human factors when building socio-technical systems following agile methodologies". In: *International Conference on Human-Centred Software Engineering*. Springer. 2010, pp. 182–189.

[71] P. Bourque et al. *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE, 2014.

[72] P. Boxer. "Building organizational agility into large-scale software-reliant environments". In: *International Systems Conference*. 2009.

[73] A. C. Boynton, B. Victor, and P. J. "New competitive strategies: Challenges to organizations and information technology". In: *IBM Systems Journal* 32.1 (1993), pp. 40–64.

[74] J. Brancheau, B. Janz, and J. Wetherbe. "Key Issues in Information Systems Management: 1994-95 SIM Delphi Results". In: *MIS Quarterly* 20.2 (1996), pp. 225–242.

[75] P. Bresciani and P. Donzelli. "The Agent at the Center of the Requirements Engineering Process". In: *International Workshop on Software Engineering for Large-Scale Multi-Agent Systems*. Springer. 2003, pp. 1–18.

[76] J. Brier, L. Rapanotti, and J. Hall. "Problem-based analysis of organisational change: a real-world example". In: *International Workshop on Advances and Applications of Problem Frames*. ACM. 2006, pp. 13–18.

[77] F. P Brooks. *The mythical man-month. Essays on software engineering*. Addison-Wesley, 1982.

[78] S. Brown. *500 tips on group learning*. Routledge, 2014.

[79] W. Brown et al. *AntiPatterns: refactoring software, architectures, and projects in crisis*. John Wiley & Sons, Inc., 1998.

[80] R. Buchanan. "Wicked problems in design thinking". In: *Design issues* 8.2 (1992), pp. 5–21.

[81] C. Calero, F. Ruiz, and M. Piattini. *Ontologies for software engineering and software technology*. Springer, 2006.

[82] J. C Camillus. "Strategy as a wicked problem". In: *Harvard Business Review* 86.5 (2008), p. 98.

[83] G. Campbell and P. Papapetrou. *SonarQube in action*. Manning Publications Co., 2013.

[84] L. Carter. "Ideas for Adding Soft Skills Education to Service Learning and Capstone Courses for Computer Science Students". In: *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*. SIGCSE '11. Dallas, TX, USA: ACM, 2011, pp. 517–522. ISBN: 978-1-4503-0500-6.

[85] R. Castro et al. "An Ontology Model to Support the Automated Evaluation of Software". In: *International Conference on Software Engineering and Knowledge Engineering*. Knowledge Systems Institute Graduate School, 2010.

[86] M. Cataldo, J. Herbsleb, and K. Carley. "Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity". In: *International Symposium on Empirical Software Engineering and Measurement*. ACM. 2008, pp. 2–11.

[87] M. Cataldo et al. "Identification of coordination requirements: implications for the Design of collaboration and awareness tools". In: *Conference on Computer Supported Cooperative Work*. ACM. 2006, pp. 353–362.

[88] C. Cevenini et al. "Privacy Through Anonymisation in Large-Scale Socio-Technical Systems: Multi-lingual Contact Centres Across the EU". In: *International Conference on Internet Science*. Springer. 2016, pp. 291–305.

[89] P. Checkland. *Systems thinking, systems practice*. John Wiley, 1981.

[90] C. Cherryholmes. "Notes on pragmatism and scientific realism". In: *Educational researcher* 21.6 (1992), pp. 13–17.

[91] W. W. Chin. "Issues and Opinion on Structural Equation Modeling". In: *MIS Quarterly* 22.1 (1998).

[92] W. Chin and P R. Newsted. *Structural Equation Modeling Analysis with Small Samples Using Partial Least Square*. Sage, 1996.

[93] W. W. Chin. "The partial least squares approach to structural equation modeling". In: *Modern Methods for Business Research* 295.2 (1998), pp. 295–336.

[94] T. Chow and D. Cao. "A survey study of critical success factors in agile software projects". In: *Journal of Systems and Software* 81.6 (2008), pp. 961–971.

[95] M. B. Chrissis, M. Konrad, and S. Shrum. *CMMI guidlines for process integration and product improvement*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[96] C. Churchman. *The Design of Inquiring Systems Basic Concepts of Systems and Organization*. Basic Books, 1971.

[97] P. Ciancarini, F. Poggi, and D. Russo. "Big Data Quality: a Roadmap for Open Data". In: *2nd IEEE International Conference on Big Data Service (BigDataService)*. IEEE. 2016, pp. 210–215.

[98] P. Ciancarini and V. Presutti. "Towards ontology driven software design". In: *Radical Innovations of Software and Systems Engineering in the Future*. Springer, 2004, pp. 122–136.

[99] P. Ciancarini et al. "A Guided Tour of the Legal Implications of Software Cloning". In: *Proceedings of the 38th International Conference on Software Engineering Companion*. ICSE '16. ACM. 2016, pp. 563–572.

[100] P. Ciancarini et al. "Agile Knowledge Engineering for Mission Critical Software Requirements". In: *Knowledge Engineering and Software Engineering - Methods, tools, and case studies*. Ed. by G. Nalepa and J. Baumeister. Springer-Verlag, Berlin, 2017, pp. 1–21.

[101] P. Ciancarini et al., eds. *Proc. 5th Int. Conf. on Software Engineering for Defense Applications*. Vol. 717. Advances in Intelligent Systems and Computing. Springer, 2018.

[102] P. Ciancarini et al. "Reverse engineering: a European IPR perspective". In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. ACM. 2016, pp. 1498–1503.

[103] P. Ciancarini and G. P. Favini. "Detecting clones in game-playing software". In: *Entertainment Computing* 1.1 (2009), pp. 9–15. ISSN: 1875-9521.

[104] J. Cleland-Huang et al. "The twin peaks of requirements and architecture". In: *IEEE Software* 30.2 (2013), pp. 24–29.

[105] P. Clements, R. Kazman, and M. Klein. *Evaluating Software Architectures*. Addison-Wesley, Boston, 2002.

[106] E. Communities. "Key competences for lifelong learning: European Reference Framework". In: (2007).

[107] M. Conway. "How do committees invent". In: *Datamation* 14.4 (1968), pp. 28–31.

[108] H. Cooper. "Scientific guidelines for conducting integrative research reviews". In: *Review of Educational Research* 52.2 (1982), pp. 291–302.

[109] H. Cooper, L. Hedges, and J. Valentine. *The handbook of research synthesis and meta-analysis*. Sage, 2009.

[110] F. R. Cotugno and A. Messina. "Adapting Scrum to the Italian Army: Methods and (Open) Tools". In: *IFIP International Conference on Open Source Systems*. Springer. 2014, pp. 61–69.

[111] K. Craik. *The nature of exploration*. 1943.

[112] J. Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage, 2013.

[113] J. Creswell, V. Clark, and L. Plano. *Designing and conducting Mixed Methods research*. Wiley, 2007.

[114] M. Crofts, B. Fraunholz, and M. Warren. "Using the Sociotechnical Approach in Global Software Developments: Is the Theory Relevant today?" In: *Australasian Conference on Information Systems* (2008), pp. 250–260.

[115] A. Csizmadia et al. *Computational thinking: A guide for teachers*. 2015.

[116] M. A. Cusumano. *The Business of Software: What every manager, programmer, and entrepreneur must know to thrive and survive in good times and bad*. Simon and Schuster, 2004.

[117] N. C. Dalkey, B. B. Brown, and S. Cochran. *The Delphi method: An experimental study of group opinion*. Vol. 3. Rand Corporation Santa Monica, CA, 1969.

[118] F. Dalpiaz, P. Giorgini, and J. Mylopoulos. "Adaptive socio-technical systems: a requirements-based approach". In: *Requirements Engineering* 18.1 (2013), pp. 1–24.

[119] D. Damian et al. "The role of domain knowledge and cross-functional communication in socio-technical coordination". In: *International Conference on Software Engineering (ICSE)*. IEEE. 2013, pp. 442–451.

[120] W. Damon and E. Phelps. “Critical distinctions among three approaches to peer education”. In: *International Journal of Educational Research* 13.1 (1989), pp. 9–19.

[121] D. F. Dansereau. “Cooperative learning strategies”. In: *Learning and study strategies: Issues in assessment, instruction, and evaluation* (1988), pp. 103–120.

[122] J. S. David, D. Schuff, and R. St Louis. “Managing your total IT cost of ownership”. In: *Communications of the ACM* 45.1 (2002), pp. 101–106.

[123] C. Deephouse et al. “Software processes and project performance”. In: *Journal of Management Information Systems* 12.3 (1995), pp. 187–205.

[124] W. DeLone and E. McLean. “Information systems success: The quest for the dependent variable”. In: *Information Systems Research* 3.1 (1992), pp. 60–95.

[125] W. DeLone and E. McLean. “The DeLone and McLean model of information systems success: a ten-year update”. In: *Journal of Management Information Systems* 19.4 (2003), pp. 9–30.

[126] P. J. Denning. “Remaining trouble spots with computational thinking”. In: *Communications of the ACM* 60.6 (2017), pp. 33–39.

[127] J. Dewey. *Logic: the theory of inquiry*. Holt, 1938.

[128] J. Dibbern et al. “Information systems outsourcing: a survey and analysis of the literature”. In: *ACM Sigmis Database* 35.4 (2004), pp. 6–102.

[129] G. W Dickson. “Management information-decision systems: A new era ahead?” In: *Business Horizons* 11.6 (1968), pp. 17–26.

[130] V. Dignum and Y. Tan. “Multi agent simulation for control and autonomy in complex socio-technical systems”. In: *International Conference on Networking, Sensing and Control (ICNSC)*. IEEE. 2011, pp. 62–67.

[131] T. K. Dijkstra and J. Henseler. “Consistent and asymptotically normal PLS estimators for linear structural equations”. In: *Computational Statistics & Data Analysis* 81 (2015), pp. 10–23.

[132] T. Dingsøyr and C. Lassenius. “Emerging themes in agile software development: Introduction to the special section on continuous value delivery”. In: *Information and Software Technology* 77 (2016), pp. 56–60.

[133] T. Dingsøyr et al. “Team Performance in Software Development: Research Results versus Agile Principles”. In: *IEEE Software* 33.4 (2016), pp. 106–110.

[134] L. Dobrica and E. Niemela. “A survey on software architecture analysis methods”. In: *IEEE Transactions on Software Engineering* 28.7 (2002), pp. 638–653.

[135] E. Doke and N. Swanson. “Decision variables for selecting prototyping in information systems development: A Delphi study of MIS managers”. In: *Information & Management* 29.4 (1995), pp. 173–182.

[136] A. Dorling. “SPICE: Software process improvement and capability determination”. In: *Software Quality Journal* 2.4 (1993), pp. 209–224.

[137] C. Dorn, G. Edwards, and N. Medvidovic. “Analyzing design tradeoffs in large-scale socio-technical systems through simulation of dynamic collaboration patterns”. In: *Confederated International Conferences On the Move to Meaningful Internet Systems*. Springer. 2012, pp. 362–379.

[138] C. Dorn and R. Taylor. "Coupling software architecture and human architecture for collaboration-aware system adaptation". In: *International Conference on Software Engineering*. IEEE Press. 2013, pp. 53–62.

[139] R. Dos Santos and C. Werner. "On the Impact of Software Ecosystems in Requirements Communication and Management." In: *Requirements Engineering @ Brazil*. Citeseer. 2013.

[140] T. Drozdowski et al. "India's Rise as a Software Power: Governmental Policy Factors". In: *Portland International Conference on Management of Engineering and Technology*. IEEE. 2007, pp. 2811–2819.

[141] K. Eason. *Information technology and organisational change*. CRC Press, 2014.

[142] S. Easterbrook et al. "Experiences using lightweight formal methods for requirements modeling". In: *IEEE Transactions on Software Engineering* 24.1 (1998), pp. 4–14.

[143] C. Ebert and M. Paasivaara. "Scaling Agile". In: *IEEE Software* 6 (2017), pp. 98–103.

[144] A. Edmonson. *Teaming to Innovate*. Wiley, 2013.

[145] A. Edmonson. "Wicked Problem Solvers". In: *Harvard Business Review* 94.June (2016), p. 52.

[146] T. G. of Education Reform. *21st century skills*. http://edglossary.org/21st-century-skills/. 2016.

[147] B. Efron and R. J. Tibshirani. *An introduction to the bootstrap*. CRC Press, 1994.

[148] K. Ehrlich and K. Chang. "Leveraging expertise in global software teams: Going outside boundaries". In: *International Conference on Global Software Engineering*. IEEE. 2006, pp. 149–158.

[149] K. E. Emam, W. Melo, and J.-N. Drouin. *SPICE: The theory and practice of software process improvement and capability determination*. IEEE, 1997.

[150] D. Emery and R. Hilliard. "Every architecture description needs a framework: Expressing architecture frameworks using ISO/IEC 42010". In: *Conference on Software Architecture*. IEEE. 2009, pp. 31–40.

[151] E Engström and P. Runeson. "Software product line testing–a systematic mapping study". In: *Information and Software Technology* 53.1 (2011), pp. 2–13.

[152] A. Espinosa et al. "Team knowledge and coordination in geographically distributed software development". In: *Journal of Management Information Systems* 24.1 (2007), pp. 135–169.

[153] J Etezadi-Amoli and A. Farhoomand. "A structural model of end user computing satisfaction and user performance". In: *Information & Management* 30.2 (1996), pp. 65–73.

[154] D. Fabiano et al. "Applying Tropos to Socio-Technical System Design and Runtime Configuration". In: *Italian Workshop dagli Oggetti agli Agenti*. 2008.

[155] F. Faul et al. "Statistical power analyses using G* Power 3.1: Tests for correlation and regression analyses". In: *Behavior Research Methods* 41.4 (2009), pp. 1149–1160.

[156] D. Feeny, M. Lacity, and L. Willcocks. "Taking the measure of outsourcing providers". In: *MIT Sloan Management Review* 46.3 (2005), p. 41.

[157] D. F. Feeny, B. R. Edwards, and K. M. Simpson. "Understanding the CEO/CIO relationship". In: *MIS Quarterly* (1992), pp. 435–448.

[158] F. Ferrucci, C. Gravino, and L. Lavazza. "Simple function points for effort estimation: a further assessment". In: *Proc. 31st ACM Symposium on Applied Computing*. 2016, pp. 1428–1433.

[159] C. Fornell and D. F. Larcker. "Evaluating structural equation models with unobservable variables and measurement error". In: *Journal of Marketing Research* (1981), pp. 39–50.

[160] E. Fraj-Andrés, L. Lucia-Palacios, and R. Pérez-López. "How extroversion affects student attitude toward the combined use of a wiki and video recording of group presentations". In: *Computers & Education* 119 (2018), pp. 31 –43.

[161] R. France and B. Rumpe. "Model-driven Development of Complex Software: A Research Roadmap". In: *Future of Software Engineering*. FOSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 37–54. ISBN: 0-7695-2829-5.

[162] B. Gallina, E. Sefer, and A. Refsdal. "Towards safety risk assessment of socio-technical systems via failure logic analysis". In: *International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE. 2014, pp. 287–292.

[163] A. Gangemi. "Norms and plans as unification criteria for social collectives". In: *Autonomous Agents and Multi-Agent Systems* 17.1 (2008), pp. 70–112. DOI: 10.1007/s10458-008-9038-9. URL: https://doi.org/10.1007/s10458-008-9038-9.

[164] A. Gangemi and P. Mika. "Understanding the semantic web through descriptions and situations". In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2003, pp. 689–706.

[165] A. Gangemi and V. Presutti. "Ontology Design Patterns". In: Springer, 2009, pp. 221–243.

[166] E. Gansner and S. North. "An open graph visualization system and its applications to software engineering". In: *Software: practice and experience* 30.11 (2000), pp. 1203–1233.

[167] D. Garlan and D. E Perry. "Introduction to the special issue on software architecture". In: *IEEE Transaction on Software Engineering* 21.4 (1995), pp. 269–274.

[168] S. Gazzerro et al. "Capturing User Needs for Agile Software Development". In: *Proceedings of 4th International Conference in Software Engineering for Defence Applications*. Springer. 2016, pp. 307–319.

[169] D. Gefen, D. Straub, and M.-C. Boudreau. "Structural equation modeling and regression: Guidelines for research practice". In: *Communications of the AIS* 4.1 (2000), p. 7.

[170] S. Geisser. "A predictive approach to the random effect model". In: *Biometrika* 61.1 (1974), pp. 101–107.

[171] J. Georgas and A. Sarma. "STCML: an extensible XML-based language for socio-technical modeling". In: *International Workshop on Cooperative and Human Aspects of Software Engineering*. ACM. 2011, pp. 61–64.

[172] D. M. Germán and A. E. Hassan. "License integration patterns: Addressing license mismatches in component-based development". In: *2009 IEEE 31st International Conference on Software Engineering* (2009), pp. 188–198.

[173] B. G Glaser. *Basics of grounded theory analysis: Emergence vs forcing*. Sociology Press, 1992.

[174] B. G Glaser. *Theoretical sensitivity: Advances in the methodology of grounded theory*. Sociology Press, 1978.

[175] G. Glass. "Primary, secondary, and meta-analysis of research". In: *Educational Researcher* 5.10 (1976), pp. 3–8.

[176] R. Glass, I. Vessey, and V. Ramesh. "Research in software engineering: an analysis of the literature". In: *Information and Software Technology* 44.8 (2002), pp. 491–506.

[177] S. Goggins and W. Xing. "Building models explaining student participation behavior in asynchronous online discussion". In: *Computers & Education* 94 (2016), pp. 241 –251.

[178] A. Gomez-Perez, M. Fernández-López, and O. Corcho. *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Springer Science & Business Media, 2006.

[179] C. Gonzalez-Perez et al. "An Ontology for ISO software engineering standards: 2) Proof of concept and application". In: *Computer Standards & Interfaces* 48 (2016), pp. 112–123.

[180] N. Gorla and S. C. Lin. "Determinants of software quality: A survey of information systems project managers". In: *Information and Software Technology* 52.6 (2010), pp. 602–610.

[181] N. Gorla, T. Somers, and B. Wong. "Organizational impact of system quality, information quality, and service quality". In: *The Journal of Strategic Information Systems* 19.3 (2010), pp. 207–228.

[182] A. Gregoriades and A. Sutcliffe. "Scenario-based assessment of nonfunctional requirements". In: *IEEE Transactions on Software Engineering* 31.5 (2005), pp. 392–409.

[183] H.-G. Gross, M. Melideo, and A. Sillitti. "Self-certification and Trust in Component Procurement". In: *Sci. Comput. Program.* 56.1-2 (Apr. 2005), pp. 141–156. ISSN: 0167-6423.

[184] M. Grüninger and M. S. Fox. "The role of competency questions in enterprise engineering". In: *Benchmarking–Theory and practice*. Springer, 1995, pp. 22–31.

[185] E. Guba. "Criteria for assessing the trustworthiness of naturalistic inquiries". In: *Educational Communication and Technology Journal* 29.2 (1981), pp. 75–91.

[186] L. M. Guglielmino and P. J. Guglielmino. "Practical experience with self-directed learning in business and industry human resource development". In: *New Directions for Adult and Continuing Education* 1994.64 (1994), pp. 39–46.

[187] J. Gulden, D. van der Linden, and B. Aysolmaz. "A Research Agenda on Visualizations in Information Systems Engineering". In: *International Conference on Evaluation of Novel Software Approaches to Software Engineering*. SciTePress. 2016.

[188] U. G. Gupta and R. E. Clarke. "Theory and applications of the Delphi technique: A bibliography (1975–1994)". In: *Technological forecasting and social change* 53.2 (1996), pp. 185–211.

[189] C. Hahn et al. “Capturing Customer Heterogeneity Using A Finite Mixture Pls Approach”. In: *Schmalenbach Business Review* 54.3 (2002), pp. 243–269.

[190] J. F. Hair, C. M. Ringle, and M. Sarstedt. “PLS-SEM: Indeed a silver bullet”. In: *Journal of Marketing theory and Practice* 19.2 (2011), pp. 139–152.

[191] J. F. Hair Jr et al. *A primer on partial least squares structural equation modeling (PLS-SEM)*. Sage Publications, 2016.

[192] J. Hall and L. Rapanotti. “A design theory for software engineering”. In: *Information and Software Technology* 87 (2017), pp. 46–61.

[193] K. Harrison-Broninski and J. Korhonen. “Collaboration infrastructure for the learning organization”. In: *International Conference on Business Information Systems*. Springer. 2012, pp. 120–131.

[194] D. Harvie and A. Agah. “Targeted Scrum: Applying Mission Command to Agile Software Development”. In: *IEEE Transactions on Software Engineering* 42.5 (2016), pp. 476–489.

[195] J. Hassine and M. Alshayeb. “Measurement of Actor External Dependencies in GRL Models.” In: *iStar*. Citeseer. 2014.

[196] S. Hayne and C. Pollard. “A comparative analysis of critical issues facing Canadian information systems personnel: A national and global perspective”. In: *Information & Management* 38.2 (2000), pp. 73–86.

[197] C. Heath et al. “Unpacking collaboration: the interactional organisation of trading in a city dealing room”. In: *Computer Supported Cooperative Work* 3.2 (1994), pp. 147–165.

[198] P. B. Henderson. “Ubiquitous Computational Thinking”. In: *IEEE Computer* 42.10 (2009).

[199] B. Henderson-Sellers et al. “An ontology for ISO software engineering standards”. In: *Computer Standards & Interfaces* 36.3 (2014), pp. 563–576.

[200] J. Henseler, C. M. Ringle, and M. Sarstedt. “A new criterion for assessing discriminant validity in variance-based structural equation modeling”. In: *Journal of the Academy of Marketing Science* 43.1 (2015), pp. 115–135.

[201] J. Henseler, C. M. Ringle, and R. R. Sinkovics. “The use of partial least squares path modeling in international marketing”. In: *New challenges to international marketing*. Emerald Group Publishing Limited, 2009, pp. 277–319.

[202] J. Herbsleb. “Building a socio-technical theory of coordination: why and how (outstanding research award)”. In: *Foundations of Software Engineering*. ACM. 2016, pp. 2–10.

[203] J. Herbsleb. “Global software engineering: The future of socio-technical coordination”. In: *Future of Software Engineering*. IEEE. 2007, pp. 188–198.

[204] J. Herbsleb and R. Grinter. “Architectures, coordination, and distance: Conway’s law and beyond”. In: *IEEE Software* 16.5 (1999), pp. 63–70.

[205] H. J. van den Herik et al. “Plagiarism in game programming competitions”. In: *Journal of Entertainment Computing* 5 (3 2014), pp. 173–187.

[206] B. Hermalin, A. Katz, and R. Craswell. “The Law and Economics of Contracts”. In: *Handbook of Law and Economics*. Ed. by M. Polinsky and S. Shavell. Elsevier, 2007, pp. 3–138.

[207] T. Herrmann et al. "Concepts for usable patterns of groupware applications". In: *International Conference on Supporting Group Work*. ACM. 2003, pp. 349–358.

[208] C. A. Higgins, L. E. Duxbury, and R. H. Irving. "Work-family conflict in the dual-career family". In: *Organizational Behavior and Human Decision Processes* 51.1 (1992), pp. 51–75.

[209] J. Highsmith and M. Fowler. "The Agile manifesto". In: *Software Development Magazine* 9.8 (2001), pp. 29–30.

[210] R. Hirschheim and H. K Klein. "Tracing the history of the information systems field". In: *The Oxford handbook of management information systems: Critical perspectives and new directions* (2011), pp. 16–61.

[211] P. Hitzler et al. "Towards a simple but useful ontology design pattern representation language". In: *Proc. of the WOP2017*. CEUR-ws, 2017.

[212] E. Hollnagel and D. Woods. *Joint cognitive systems: Foundations of cognitive systems engineering*. CRC Press, 2005.

[213] M. Horspool and M. Humphreys. *European Union Law*. Oxford University Press, 2012.

[214] A. Hoskey and S. Zhang. "Computational Thinking: what does it really mean for the K-16 computer science education community". In: *Journal of Computing Sciences in Colleges* 32.3 (2017), pp. 129–135.

[215] R. A. Howard, C. A. Carver, and W. D. Lane. "Felder's learning styles, Bloom's taxonomy, and the Kolb learning cycle: tying it all together in the CS2 course". In: *ACM SIGCSE Bulletin*. Vol. 28. ACM. 1996, pp. 227–231.

[216] J. Howison and K. Crowston. "Collaboration through open superposition". In: *MIS Quarterly* 38.1 (2014), pp. 29–50.

[217] J. L. Howland, D. H. Jonassen, and R. M. Marra. *Meaningful learning with technology*. Pearson Upper Saddle River, 2012.

[218] Y. Hu et al. "A geo-ontology design pattern for semantic trajectories". In: *International Conference on Spatial Information Theory*. Springer. 2013, pp. 438–456.

[219] S. Hudert, S. Konig, and T. Eymann. "A Proposal for a Life Cycle Model for Electronic Service Markets". In: *Conference on Commerce and Enterprise Computing (CEC)*. IEEE. 2011, pp. 371–378.

[220] J. Hulland. "Use of partial least squares (PLS) in strategic management research: A review of four recent studies". In: *Strategic Management Journal* (1999), pp. 195–204.

[221] W. S. Humphrey. "Characterizing the software process: a maturity framework". In: *IEEE Software* 5.2 (1988), pp. 73–79.

[222] W. Hung, D. H. Jonassen, R. Liu, et al. "Problem-based learning". In: *Handbook of research on educational communications and technology* 3 (2008), pp. 485–506.

[223] K. N. Hylton, ed. *Antitrust Law and Economics*. Edward Elgar Publishing, 2010. URL: https://EconPapers.repec.org/RePEc:elg:eebook:13001.

[224] N. Itzik, I. Reinhartz-Berger, and Y. Wand. "Variability Analysis of Requirements: Considering Behavioral Differences and Reflecting Stakeholders". In: *IEEE Transactions on Software Engineering* 42.7 (2016), pp. 687–706.

[225] G. Jackson. "Methods for integrative reviews". In: *Review of Educational Research* 50.3 (1980), pp. 438–460.

[226] I. Jacobson, I. Spence, and E. Seidewitz. "Industrial-scale agile: from craft to engineering". In: *Communications of the ACM* 59.12 (2016), pp. 63–71.

[227] C. B. Jarvis, S. B. MacKenzie, and P. M. Podsakoff. "A critical review of construct indicators and measurement model misspecification in marketing and consumer research". In: *Journal of Consumer Research* 30.2 (2003), pp. 199–218.

[228] L. Jiang, K. Carley, and A. Eberlein. "Assessing team performance from a socio-technical congruence perspective". In: *International Conference on Software and System Process*. IEEE Press. 2012, pp. 160–169.

[229] D. Johnson and R. Johnson. *Learning together and alone: Cooperative, competitive, and individualistic learning*. Prentice-Hall, 1987.

[230] D. Johnson, R. Johnson, and K. Smith. *Active learning: Cooperation in the college classroom*. ERIC, 1998.

[231] D. Johnson et al. *Cooperative learning in the classroom*. ERIC, 1994.

[232] J. H. Johnson. "Substring matching for clone detection and change tracking". In: *Proceedings 10 International Conference on Software Maintenance*. 1994, pp. 120–126.

[233] M. Johnson. *Should My Kid Learn to Code?* http://googleforeducation.blogspot.gr/2015/07/should-my-kid-learn-to-code.html. 2015.

[234] P. N. Johnson-Laird. *Mental models: Towards a cognitive science of language, inference, and consciousness*. 6. Harvard University Press, 1983.

[235] S. Joy and D. A. Kolb. "Are there cultural differences in learning style?" In: *International Journal of Intercultural Relations* 33.1 (2009), pp. 69–85.

[236] R. Judd. "Use of Delphi methods in Higher Education". In: *Technological Forecasting and Social Change* 4.2 (1972), pp. 173–186.

[237] M. Jun and S. Cai. "The key determinants of internet banking service quality: a content analysis". In: *International Journal of Bank Marketing* 19.7 (2001), pp. 276–291.

[238] H. W. Jung. "Validating the external quality subcharacteristics of software products according to ISO/IEC 9126". In: *Computer Standards & Interfaces* 29.6 (2007), pp. 653–661.

[239] H. W. Jung, S. G. Kim, and C. S. Chung. "Measuring software product quality: A survey of ISO/IEC 9126". In: *IEEE Software* 21.5 (2004), pp. 88–92.

[240] E. Kabaale et al. "An Axiom Based Metamodel for Software Process Formalisation: An Ontology Approach". In: *International Conference on Software Process Improvement and Capability Determination*. Springer. 2017, pp. 226–240.

[241] S. H. Kaisler, F. Armour, and M. Valivullah. "Enterprise architecting: Critical problems". In: *Proc. 38th Annual Hawaii Int. Conf. on System Sciences (HICSS)*. IEEE. 2005, 224b.

[242] A. K. Kakar. "Assessing Self-Organization in Agile Software Development Teams". In: *Journal of Computer Information Systems* 57.3 (2017), pp. 208–217.

[243] K. Kamaruddin, N. Yusop, and M. Ali. "Using activity theory in analyzing requirements for mobile phone application". In: *Malaysian Conference in Software Engineering (MySEC)*. IEEE. 2011, pp. 7–13.

[244] C. J. Kapser and M. W. Godfrey. ""Cloning considered harmful" considered harmful: patterns of cloning in software". In: *Empirical Software Engineering* 13.6 (2008), p. 645. ISSN: 1573-7616.

[245] C. J. Kapser and M. W. Godfrey. "Supporting the analysis of clones in software systems". In: *Journal of Software Maintenance and Evolution: Research and Practice* 18.2 (), pp. 61–82. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.327.

[246] D. L. Katz. "Conference report on the use of computers in engineering classroom instruction". In: *Communications of the ACM* 3.10 (1960), pp. 522–527.

[247] A. Kayed et al. "Towards an ontology for software product quality attributes". In: *International Conference on Internet and Web Applications and Services*. IEEE. 2009, pp. 200–204.

[248] R. Kazman and H.-M. Chen. "The metropolis model and its implications for the engineering of software ecosystems". In: *Workshop on Future of Software Engineering Research*. ACM. 2010, pp. 187–190.

[249] R. Kazman et al. "SAAM: A method for analyzing the properties of software architectures". In: *Proc. Int. Conf. on Software Engineering (ICSE)*. ACM/IEEE. 1994, pp. 81–90.

[250] P. G. Keen. "MIS research: Current status, trends and needs". In: *Information systems education: Recommendations and implementation* (1987), pp. 1–13.

[251] R. Khadka et al. "How do professionals perceive legacy systems and software modernization?" In: *Proc. of the 36th Int. Conf. on Software Engineering*. ACM/IEEE. 2014, pp. 36–47.

[252] T. Kilamo, M. Leppänen, and T. Mikkonen. "The social developer: now, then, and tomorrow". In: *International Workshop on Social Software Engineering*. ACM. 2015, pp. 41–48.

[253] B.-J. Kim and S.-W. Lee. "Analytical study of cognitive layered approach for understanding security requirements using problem domain ontology". In: *Asia-Pacific Software Engineering Conference (APSEC)*. IEEE. 2016, pp. 97–104.

[254] D. Kim and E. Cavusgil. "The impact of supply chain integration on brand equity". In: *Journal of Business & Industrial Marketing* 24.7 (2009), pp. 496–505.

[255] H. Kim, D.-H. Shin, and D. Lee. "A socio-technical analysis of software policy in Korea: Towards a central role for building ICT ecosystems". In: *Telecommunications Policy* 39.11 (2015), pp. 944–956.

[256] M. Kim et al. "An ethnographic study of copy and paste programming practices in OOPL". English (US). In: *Proceedings – 2004 International Symposium on Empirical Software Engineering, ISESE 2004*. Dec. 2004, pp. 83–92. ISBN: 0769521657.

[257] B. Kitchenham, D. Budgen, and P. Brereton. "Using mapping studies as the basis for further research–a participant-observer case study". In: *Information and Software Technology* 53.6 (2011), pp. 638–651.

[258] B. A. Kitchenham, D. Budgen, and P. Brereton. *Evidence-based software engineering and systematic reviews*. Vol. 4. CRC Press, 2015.

[259] H. Knublauch. "Ramblings on Agile methodologies and ontology-driven software development". In: *Workshop on Semantic Web Enabled Software Engineering (SWESE)*. Galway, Ireland, 2005.

[260] H. Knublauch et al. “The Protégé OWL plugin: An open development environment for semantic web applications”. In: *International Semantic Web Conference*. Springer. 2004, pp. 229–243.

[261] D. Kolb. *Learning Style Inventory technical manual*. McBer Boston, MA, 1976.

[262] R. Koschke. “Frontiers of Software Clone Management”. In: *Proceedings of the 2008 Frontiers of Software Maintenance, FoSM 2008*. Nov. 2008, pp. 119–128.

[263] M. Krafft, K. Stol, and B. Fitzgerald. “How Do Free/Open Source Developers Pick Their Tools?: A Delphi Study of the Debian Project”. In: *Proc. 38th Int. Conf. on Software Engineering (ICSE)*. ACM/IEEE. 2016, pp. 232–241.

[264] P. Kristiansen and R. Rasmussen. *Building a better business using the Lego serious play method*. John Wiley & Sons, 2014.

[265] M. Kropp and A. Meier. “New sustainable teaching approaches in software engineering education”. In: *Proc. IEEE Global Engineering Education Conference (EDUCON)*. 2014, pp. 1019–1022.

[266] M. Kropp and A. Meier. “Teaching agile software development at university level: Values, management, and craftsmanship”. In: *Proc. 26th IEEE Conf. on Software Engineering Education and Training (CSEE&T)*. 2013, pp. 179–188.

[267] V. Krotov. “Bridging the CIO-CEO gap: It takes two to tango”. In: *Business Horizons* 58.3 (2015), pp. 275–283.

[268] A. Kuhn. “Immediate Search in the IDE as an Example of Socio-Technical Congruence in Search-driven development”. In: *Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation*. ACM. 2010, pp. 25–28.

[269] M. Kuhrmann and J. Münch. “When teams go crazy: An environment to experience group dynamics in software project management courses”. In: *Proceedings of the International Conference on Software Engineering*. ACM. 2016, pp. 412–421.

[270] K. Kumar and T. Prabhakar. “Pattern-oriented knowledge model for architecture design”. In: *Conference on Pattern Languages of Programs*. ACM. 2010, p. 23.

[271] M. Kumar, N. Ajmeri, and S. Ghaisas. “Towards Knowledge Assisted Agile Requirements Evolution”. In: *Proc.e 2Nd Int. Workshop on Recommendation Systems for Software Engineering*. RSSE ’10. Cape Town, South Africa: ACM, 2010, pp. 16–20.

[272] I. Kwan and D. Damian. “Extending socio-technical congruence with awareness relationships”. In: *International Workshop on Social Software Engineering*. ACM. 2011, pp. 23–30.

[273] I. Kwan, A. Schroter, and D. Damian. “Does socio-technical congruence have an effect on software build success? a study of coordination in a software project”. In: *IEEE Transactions on Software Engineering* 37.3 (2011), pp. 307–324.

[274] M. Lange, J. Mendling, and J. Recker. “An empirical analysis of the factors and measures of Enterprise Architecture Management success”. In: *European Journal of Information Systems* 25.5 (2016), pp. 411–431.

[275] B. Langefors. *Theoretical analysis of information systems*. Tech. rep. 1973.

[276] M. Lanza and S. Ducasse. “Polymetric views-a lightweight visual approach to reverse engineering”. In: *IEEE Transactions on Software Engineering* 29.9 (2003), pp. 782–795.

[277] M. Lapham, M. Bandor, and E. Wrubel. *Agile Methods and Request for Change (RFC): Observations from DoD Acquisition Programs*. Tech. rep. CMU-SEI-13-TN-31. Software Engineering Institute, Carnegie Mellon University, 2014.

[278] M. Lapham et al. *Agile Methods: Selected DoD Management and Acquisition Concerns*. Tech. rep. CMU-SEI-11-TN-2. Software Engineering Institute, Carnegie Mellon University, 2011.

[279] M. Lapham et al. *RFP Patterns and Techniques for Successful Agile Contracting*. Tech. rep. CMU-SEI-13-SR-25. Software Engineering Institute, Carnegie Mellon University, 2016.

[280] P. Lasserre and C. Szostak. "Effects of team-based learning on a CS1 course". In: *Proc. 16th Annual Joint Conf. on Innovation and technology in Computer Science Education*. ACM. 2011, pp. 133–137.

[281] L. Lavazza and R. Meli. "An evaluation of simple function point as a replacement of IFPUG function point". In: *Proc. Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement*. IEEE. 2014, pp. 196–206.

[282] T. Lavoie, M. Eilers-Smith, and E. Merlo. "Challenging Cloning Related Problems with GPU-based Algorithms". In: *Proceedings of the 4th International Workshop on Software Clones*. IWSC '10. Cape Town, South Africa: ACM, 2010, pp. 25–32. ISBN: 978-1-60558-980-0.

[283] P. Lavrakas. *Encyclopedia of survey research methods*. Sage, 2008.

[284] D. Leffingwell. *SAFe® 4.0 Reference Guide: Scaled Agile Framework® for Lean Software and Systems Engineering*. Addison-Wesley Professional, 2016.

[285] C. Lentzsch et al. "Integrating a Practice Perspective to Privacy by Design". In: *International Conference on Human Aspects of Information Security, Privacy, and Trust*. Springer. 2017, pp. 691–702.

[286] J. Letouzey and M. Ilkiewicz. "Managing technical debt with the SQALE method". In: *IEEE Software* 29.6 (2012), pp. 44–51.

[287] Z. Li, P. Liang, and P. Avgeriou. "Architectural technical debt identification based on architecture decisions and change scenarios". In: *Proc. 12th Work. Int. Conf on Software Architecture (WICSA)*. IEEE. 2015, pp. 65–74.

[288] L. Liao, Y. Qu, and H. Leung. "A software process ontology and its application". In: *Workshop on Semantic Web Enabled Software Engineering*. 2005, pp. 6–10.

[289] Y. Lindsjørn et al. "Teamwork quality and project success in software development: A survey of agile development teams". In: *Journal of Systems and Software* 122 (2016), pp. 274–286.

[290] R. Lock and I. Sommerville. "Modelling and Analysis of Socio-Technical System of Systems". In: *International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE. 2010, pp. 224–232.

[291] M. Lodi, S. Martini, and E. Nardelli. "Abbiamo davvero bisogno del pensiero computazionale?" In: *Mondo Digitale* 72 (2017), pp. 1–15.

[292] C. López et al. "Visualization and comparison of architecture rationale with semantic web technologies". In: *Journal of Systems and Software* 82.8 (2009), pp. 1198–1210.

[293] E. Y. Lu et al. "Wireless Internet and student-centered learning: A Partial Least-Squares model". In: *Computers & Education* 49.2 (2007), pp. 530 –544.

[294] G. Lucassen et al. “Improving agile requirements: the quality user story framework and tool”. In: *Requirements Engineering* 21.3 (2016), pp. 383–403.

[295] J. Luftman and H. S. Zadeh. “Key information technology and management issues 2010–11: an international study”. In: *Journal of Information Technology* 26.3 (2011), pp. 193–204.

[296] K. Lyytinen, L. Mathiassen, and J. Ropponen. “Attention shaping and software risk—a categorical analysis of four classical risk management approaches”. In: *Information Systems Research* 9.3 (1998), pp. 233–255.

[297] J. Machado et al. “OntoSoft Process: Towards an agile process for ontology-based software”. In: *49th Hawaii International Conference on System Sciences (HICSS)*. IEEE. 2016, pp. 5813–5822.

[298] B. MacKellar. “Analyzing coordination among students in a software engineering project course”. In: *Conference on Software Engineering Education and Training (CSEE&T)*. IEEE. 2013, pp. 279–283.

[299] D. Martin et al. “Cooperative work in software testing”. In: *International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. ACM. 2008, pp. 93–96.

[300] A. Martini and J. Bosch. “The danger of architectural technical debt: Contagious debt and vicious circles”. In: *Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE. 2015, pp. 1–10.

[301] P. A. McNutt. *Law, economics and antitrust: towards a new perspective*. Edward Elgar, 2005.

[302] G. H. Mead. “The social self”. In: *The Journal of Philosophy, Psychology and Scientific Methods* 10.14 (1913), pp. 374–380.

[303] N. Medvidovic and R. N. Taylor. “Software architecture: foundations, theory, and practice”. In: *Proc. Int. Conf. Software Engineering (ICSE)*. ACM/IEEE. 2010, pp. 471–472.

[304] O. Meerbaum-Salant and O. O. Hazzan. “An Agile Constructionist Mentoring Methodology for Software Projects in the High School”. In: *ACM Transactions on Computing Education* 9.4 (2010), n4.

[305] A. Meier, M. Kropp, and G. Perellano. “Experience Report of Teaching Agile Collaboration and Values: Agile Software Development in Large Student Teams”. In: *Proc. 29th IEEE Conf. on Software Engineering Education and Training (CSEE&T))*. 2016, pp. 76–80.

[306] A Meneely and L. Williams. “Socio-technical developer networks: Should we trust our measurements?” In: *International Conference on Software Engineering*. ACM. 2011, pp. 281–290.

[307] A. Messina et al. “A New Agile Paradigm for Mission-Critical Software Development”. In: *The Journal of Defense Software Engineering (CrossTalk)* 6 (2016), pp. 25–30.

[308] metamodel. *Oxford Dictionary Of English*. 3th. Oxford University Press, 2013.

[309] L. K. Michaelsen, A. B. Knight, and L. D. Fink. *Team-based learning: A transformative use of small groups*. Greenwood publishing group, 2002.

[310] J. Miller and B. A Doyle. “Measuring the effectiveness of computer-based information systems in the financial services sector”. In: *MIS Quarterly* (1987), pp. 107–124.

[311] M. Missiroli, D. Russo, and P. Ciancarini. "Cooperative Thinking, or: Computational Thinking Meets Agile". In: *Proceedings of the Conference on Software Engineering Education and Training*. IEEE. 2017, pp. 187–191.

[312] M. Missiroli, D. Russo, and P. Ciancarini. "Learning Agile software development in high school: an investigation". In: *Proc. 38th Int. Conf. on Software Engineering*. ICSE '16. ACM. 2016, pp. 293–302.

[313] M. Missiroli, D. Russo, and P. Ciancarini. "Una didattica Agile per la programmazione". In: *Mondo Digitale* 15.64 (2016).

[314] M. Missiroli, D. Russo, and P. Ciancarini. "Agile for millennials: a comparative study". In: *Proceedings of the 1st International Workshop on Software Engineering Curricula for Millennials*. IEEE. 2017, pp. 47–53.

[315] M. Missiroli, D. Russo, and P. Ciancarini. "Teaching Test-First Programming: assessment and solutions". In: *COMPSAC, 2017*. IEEE, 2017.

[316] I. Mistrik et al., eds. *Relating System Quality and Software Architecture*. Morgan Kaufmann, 2014.

[317] S. N. Mohanty. "Models and measurements for quality assessment of software". In: *ACM Computing Surveys* 11.3 (1979), pp. 251–275.

[318] A. Monden et al. "Software quality analysis by code clones in industrial legacy software". In: *Proceedings Eighth IEEE Symposium on Software Metrics*. 2002, pp. 87–94.

[319] F. Montori, L. Bedogni, and L. Bononi. "A collaborative Internet of Things architecture for smart cities and environmental monitoring". In: *IEEE Internet of Things Journal* 5.2 (2018), pp. 592–605.

[320] D. Moody. "Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions". In: *Data & Knowledge Engineering* 55.3 (2005), pp. 243–276.

[321] I. Morales-Ramirez et al. "Revealing the obvious?: A retrospective artefact analysis for an ambient assisted-living project". In: *International Workshop on Empirical Requirements Engineering (EmpiRE)*. IEEE. 2012, pp. 41–48.

[322] K. Mordal-Manet et al. "The squale model–A practice-based industrial quality model". In: *Proc. Int. Conf. on Software Maintenance (ICSM)*. IEEE. 2009, pp. 531–534.

[323] S. Motogna et al. "Improving software quality using an ontology-based approach". In: *OTM Confederated International Conferences*. Springer. 2015, pp. 456–465.

[324] C. Moustakas. *Phenomenological research methods*. Sage, 1994.

[325] P. Mulligan. "Specification of a capability-based IT classification framework". In: *Information and Management* 39.8 (2002), pp. 647–658.

[326] E. Mumford. "Computer systems and work design: Problems of philosophy and vision". In: *Personnel Review* 3.2 (1974), pp. 40–49.

[327] E. Mumford. *Designing human systems*. 1983.

[328] E. Mumford. "The story of socio-technical design: Reflections on its successes, failures and potential". In: *Information Systems Journal* 16.4 (2006), pp. 317–342.

[329] G. Murphy. *The Need for Context in Software Engineering*. Keynote at the International Conference on Automated Software Engineering. 2018.

[330] R. Myerson. "Justice, Institutions, and Multiple Equilibria". In: *The Chicago Journal of International Law* 5 (2004), p. 91.

[331] N. Nagappan, B. Murphy, and V. Basili. "The influence of organizational structure on software quality". In: *Proc. International Conference on Software Engineering*. IEEE. 2008, pp. 521–530.

[332] T. Nagel. *The view from nowhere*. Oxford University Press, 1986.

[333] K. Nakakoji, K. Yamada, and E. Giaccardi. "Understanding the Nature of Collaboration in Open-Source Software Development". In: *Asia-Pacific Software Engineering Conference*. IEEE. 2005, pp. 827–834.

[334] L. Nardin et al. "Classifying sanctions and designing a conceptual sanctioning process model for socio-technical systems". In: *The Knowledge Engineering Review* 31.2 (2016), pp. 142–166.

[335] R. Nelson and S. Winter. *An evolutionary theory of economic change*. Harvard University Press, 2009.

[336] K. Nidiffer, S. Miller, and D. Carney. *Agile Methods in Air Force Sustainment: Status and Outlook*. Tech. rep. CMU-SEI-14-TN-9. Software Engineering Institute, Carnegie Mellon University, 2014.

[337] K. Nidiffer, S. Miller, and D. Carney. *Potential Use of Agile Methods in Selected DoD Acquisitions: Requirements Development and Management*. Tech. rep. CMU-SEI-13-TN-6. Software Engineering Institute, Carnegie Mellon University, 2014.

[338] S. Nidumolu. "The effect of coordination and uncertainty on software project performance: residual performance risk as an intervening variable". In: *Information Systems Research* 6.3 (1995), pp. 191–219.

[339] S. Nidumolu and G. W. Knotts. "The effects of customizability and reusability on perceived process and competitive performance of software firms". In: *MIS Quarterly* 22.2 (1998), pp. 105–137.

[340] S. Nidumolu and M. Subramani. "The matrix of control: Combining process and structure approaches to managing software development". In: *Journal of Management Information Systems* 20.3 (2003), pp. 159–196.

[341] J. Noll, S. Beecham, and I. Richardson. "Global software development and collaboration: barriers and solutions". In: *ACM Inroads* 1.3 (2010), pp. 66–78.

[342] A. Norta et al. "An agent-oriented method for designing large socio-technical service-ecosystems". In: *World Congress on Services (SERVICES)*. IEEE. 2014, pp. 242–249.

[343] J. Nunnally. *Psychometric methods*. McGraw-Hill, 1978.

[344] B. Nuseibeh. "Weaving together requirements and architectures". In: *IEEE Computer* 34.3 (2001), pp. 115–119.

[345] OECD. "Stimulating digital innovation for growth and inclusiveness". In: (2016).

[346] A. Ogunyemi et al. "HCI practices in the Nigerian software industry". In: *Human-Computer Interaction*. Springer. 2015, pp. 479–488.

[347] C. Okoli and S. Pawlowski. "The Delphi method as a research tool: an example, design considerations and applications". In: *Information & management* 42.1 (2004), pp. 15–29.

[348] G. Oliva et al. "Evolving the system's core: a case study on the identification and characterization of key developers in Apache Ant". In: *Computing and Informatics* 34.3 (2015), pp. 678–724.

[349] A. Opelt et al. *Agile Contracts*. Wiley, 2013.

[350] S. Palmquist et al. *Parallel Worlds: Agile and Waterfall Differences and Similarities*. Tech. rep. CMU-SEI-13-TN-21. Software Engineering Institute, Carnegie Mellon University, 2014.

[351] S. Papert and I. Harel. *Situating constructionism*. Vol. 36. Ablex Publishing Corporation, 1991, pp. 1–11.

[352] C. Pardo et al. "An ontology for the harmonization of multiple standards and models". In: *Computer Standards & Interfaces* 34.1 (2012), pp. 48–59.

[353] R. Parthasarthy and S. Sethi. "The impact of flexible automation on business strategy and organizational structure". In: *Academy of Management Review* 17.1 (1992), pp. 86–111.

[354] O. Paruma-Pabón et al. "Finding relationships between socio-technical aspects and personality traits by mining developer e-mails". In: *International Workshop on Cooperative and Human Aspects of Software Engineering*. ACM. 2016, pp. 8–14.

[355] S. Pasupathy, A. Asad, and P. Y. Teng. *Rethinking K–20 Education Transformation for a New Age*. https://www.atkearney.com/about-us/social-impact/related-publications-detail/-/asset_publisher/EVxmHENiBa8V/content/rethinking-k-20-education-transformation-for-a-new-age/10192.

[356] M. C. Paulk et al. "Capability maturity model, version 1.1". In: *IEEE Software* 10.4 (1993), pp. 18–27.

[357] S. Pedell et al. "Substantiating agent-based quality goals for understanding socio-technical systems". In: *International Conference on Autonomous Agents and Multiagent Systems*. Springer. 2011, pp. 80–95.

[358] C. S. Peirce. "The architectonic construction of pragmatism". In: *Collected Papers of Charles Sanders Pierce* 5 (1905), pp. 3–6.

[359] M. D. Penta et al. "An exploratory study of the evolution of software licensing". In: *ICSE (1)*. ACM, 2010, pp. 145–154.

[360] J. Peppard and J. Ward. *The strategic management of information systems: Building a digital strategy*. Wiley, 2016.

[361] K. Petersen, S. Vakkalanka, and L. Kuzniarz. "Guidelines for conducting systematic mapping studies in software engineering: An update". In: *Information and Software Technology* 64 (2015), pp. 1–18.

[362] K. Petersen et al. "Systematic Mapping Studies in Software Engineering." In: *International Conference on Evaluation and Assessment in Software Engineering*. Vol. 8. 2008, pp. 68–77.

[363] E. Petrinja, A. Sillitti, and G. Succi. "Comparing OpenBRR, QSOS, and OMM Assessment Models". In: *Open Source Software: New Horizons*. Ed. by P. Ågerfalk et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 224–238. ISBN: 978-3-642-13244-5.

[364] S. Petter, W. DeLone, and E. McLean. "Measuring information systems success: models, dimensions, measures, and interrelationships". In: *European Journal of Information Systems* 17.3 (2008), pp. 236–263.

[365] E. Pilios. "Contracting practices in traditional and agile software development". PhD thesis. University of Leiden, NL, 2015.

[366] L. F. Pitt, R. T. Watson, and C. B. Kavan. "Service Quality: A Measure of Information Systems Effectiveness." In: *MIS Quarterly* 19.2 (1995), pp. 173–187.

[367] M. Polanyi. *The Tacit Dimension*. 1966.

[368] G. Polya. *How to solve it: A new aspect of mathematical method*. Princeton University Press, 1957.

[369] K. Popper. *The logic of scientific discovery*. Routledge, 2005.

[370] J. F. Porac and H. Thomas. "Taxonomic Mental Models in Competitor Definition". In: *The Academy of Management Review* 15.2 (1990), pp. 224–240.

[371] M. Porter and J. E Heppelmann. "How smart, connected products are transforming companies". In: *Harvard Business Review* 93.10 (2015), pp. 96–114.

[372] M. Porter and V. E. Millar. "How information gives you competitive advantage". In: *Harvard Business Review* 63.4 (1985), pp. 149–160.

[373] R. Posner. "Gratuitous Promises in Economics and Law". In: *Journal of Legal Studies* 6.2 (1977), pp. 411–426.

[374] T. C. Powell and A. Dent-Micallef. "Information technology as competitive advantage: The role of human, business, and technology resources". In: *Strategic Management Journal* (1997), pp. 375–405.

[375] C. K. Prahalad and M. S. Krishnan. "The new meaning of quality in the information age." In: *Harvard Business Review* 77.5 (1998), pp. 109–18.

[376] P. Predonzani, A. Sillitti, and T. Vernazza. "Components and data-flow applied to the integration of Web services". In: *IECON'01. 27th Annual Conference of the IEEE Industrial Electronics Society (Cat. No. 37243)*. Vol. 3. 2001, pp. 2204–2207.

[377] R. Pressman. *Software Engineering: a Practitioner's Approach*. McGrawHill, 2014.

[378] V. Presutti and A. Gangemi. "Dolce+D&S Ultralite and its main ontology design patterns". In: *Ontology Engineering with Ontology Design Patterns - Foundations and Applications*. Ed. by P. Hitzler et al. Vol. 25. Studies on the Semantic Web. IOS Press, 2016, pp. 81–103.

[379] V. Presutti et al. "The role of Ontology Design Patterns in Linked Data projects". In: Springer, 2016, pp. 113–121.

[380] D. Radjenović et al. "Software fault prediction metrics: A systematic literature review". In: *Information and Software Technology* 55.8 (2013), pp. 1397–1418.

[381] M. Raskino and G. Waller. *Digital to the Core: Remastering Leadership for Your Industry, Your Enterprise, and Yourself*. Routledge, 2016.

[382] D. Rattan, R. Bhatia, and M. Singh. "Software clone detection: A systematic review". In: *Information and Software Technology* 55.7 (2013), pp. 1165–1199. ISSN: 0950-5849.

[383] D. Reifer. "Industry software cost, quality and productivity benchmarks". In: *The DoD SoftwareTech News* 7.2 (2004), pp. 3–19.

[384] T. Remencius, A. Sillitti, and G. Succi. "Assessment of Software Developed by a Third-party". In: *Inf. Sci.* 328.C (Jan. 2016), pp. 237–249. ISSN: 0020-0255.

[385] I. Richardson et al. "A process framework for global software engineering teams". In: *Information and Software Technology* 54.11 (2012), pp. 1175–1191.

[386] M. Rieger, S. Ducasse, and M. Lanza. "Insights into System-Wide Code Duplication". In: *Proceedings of the 11th Working Conference on Reverse Engineering*. WCRE '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 100–109. ISBN: 0-7695-2243-2.

[387] D. Rigby, J. Sutherland, and H. Takeuchi. "Embracing Agile". In: *Harvard Business Review* May.May (2016).

[388] S. Riis. "What makes a chess program original? Revisiting the Rybka case". In: *Entertainment Computing* 5 (Aug. 2014), pp. 189–204.

[389] C. M. Ringle, M. Sarstedt, and D. Straub. "A critical look at the use of PLS-SEM in MIS Quarterly". In: *MIS Quarterly* 36.1 (2012), pp. iii–xiv.

[390] C. M. Ringle, S. Wende, and J.-M. Becker. "SmartPLS 3". In: *Boenningstedt: SmartPLS GmbH* (2015).

[391] H. Rittel and M. M. Webber. "2.3 planning problems are wicked". In: *Polity* 4 (1973), pp. 155–169.

[392] J. G. Rivera-Ibarra, J. Rodríguez-Jacobo, and M. A. Serrano-Vargas. "Competency framework for software engineers". In: *Proc. 23rd IEEE Conf. on Software Engineering Education and Training (CSEE&T)*. 2010, pp. 33–40.

[393] P. Rodriguez et al. "Continuous deployment of software intensive products and services: A systematic mapping study". In: *Journal of Systems and Software* 123 (2017), pp. 263–291.

[394] J. Rooksby, J. Hunt, and X. Wang. "The theory and practice of Randori coding dojos". In: *Proc. 15th Int. Conf. on Agile Software Development (XP2014)*. Ed. by G. Cantone and M. Marchesi. Vol. 179. Lecture Notes in Business Information Processing. Springer, 2014, pp. 251–259.

[395] K. Rostami et al. "Architecture-based assessment and planning of change requests". In: *Proc. 11th Int. Conf. on Quality of Software Architectures (QoSA)*. ACM. 2015, pp. 21–30.

[396] K. Rostami et al. "Architecture-Based Change Impact Analysis in Information Systems and Business Processes". In: *Proc. Int. Conf. on Software Architecture (ICSA)*. IEEE. 2017, pp. 179–188.

[397] C. K. Roy and J. R. Cordy. *A Survey on Software Clone Detection Research*. Tech. rep. Report 2007–541. Ontario, Canada: Queen's School of Computing Tech., 2007.

[398] W. Royce. "Managing the development of large software systems: concepts and techniques". In: *Proc. Int. Conf. Software Engineering (ICSE)*. ACM/IEEE. 1987, pp. 328–338.

[399] K. S. Rubin. *Essential Scrum: a practical guide to the most popular agile process*. Addison-Wesley, 2012.

[400] P. Runeson, and M. Höst. "Guidelines for conducting and reporting case study research in software engineering". In: *Empirical Software Engineering* 14.2 (2008), pp. 131–164. ISSN: 1573-7616.

[401] D. Russo. "Benefits of Open Source Software in Defense Environments". In: *Proceedings of 4th International Conference in Software Engineering for Defence Applications*. Vol. 422. Springer, Advances in Intelligent Systems and Computing. 2016, pp. 123–131.

[402] D. Russo and P. Ciancarini. "A Proposal for an Antifragile Software Manifesto". In: *Procedia Computer Science* 83 (2016). The 7th Int. Conf. on Ambient Systems, Networks and Technologies (ANT 2016), pp. 982–987.

[403] D. Russo, V. Lomonaco, and P. Ciancarini. "A Machine Learning Approach for Continuous Development". In: *Proceedings of 5th International Conference in Software Engineering for Defence Applications*. Springer, 2018, pp. 109–119.

[404] D. Russo et al. "A Meta-Model for Information Systems Quality: A Mixed Study of the Financial Sector". In: *ACM Transactions on Management Information Systems* 9.3 (2018), p. 11.

[405] D. Russo et al. "Software Quality Concerns in the Italian Bank Sector: the Emergence of a Meta-Quality Dimension". In: *Proc. 39th Int. Conf. on Software Engineering*. ICSE '17. ACM/IEEE. 2017.

[406] D. Russo and P. Ciancarini. *Towards Antifragile Architectures*. Tech. rep. 2017.

[407] C. Santana et al. "Using Function Points in Agile Projects". In: *Agile Processes in Software Engineering and Extreme Programming*. Vol. 77. Lecture Notes in Business Information Processing. Springer, May 2011, pp. 176–191.

[408] R. Santos et al. "Supporting negotiation and socialization for component markets in software ecosystems context". In: *Latin American Computing Conference (CLEI)*. IEEE. 2016, pp. 1–12.

[409] R. dos Santos and C. Werner. "Revisiting the concept of components in software engineering from a software ecosystem perspective". In: *European Conference on Software Architecture*. ACM. 2010, pp. 135–142.

[410] A. Sarma, J. Herbsleb, and A. Van Der Hoek. "Challenges in measuring, understanding, and achieving social-technical congruence". In: *Socio-Technical Congruence Workshop*. 2008.

[411] A. Sarma et al. "Tesseract: Interactive visual exploration of socio-technical relationships in software development". In: *International Conference on Software Engineering*. IEEE Computer Society. 2009, pp. 23–33.

[412] D. T. Sato, H. Corbucci, and M. V. Bravo. "Coding dojo: An environment for learning and sharing agile practices". In: *Proceedings of the Agile Conference*. IEEE. 2008, pp. 459–464.

[413] S. Sawyer. "Software development teams". In: *Communications of the ACM* 47.12 (2004), pp. 95–99.

[414] S. Sawyer and H. Annabi. "Methods as theories: evidence and arguments for theorizing on software development". In: *Social Inclusion: Societal and Organizational Implications for Information Systems*. Springer, 2006, pp. 397–411.

[415] W. Scacchi. "Free/open source software development: recent research results and emerging opportunities". In: *Foundations of Software Engineering*. ACM. 2007, pp. 459–468.

[416] C. Schmidt and P. Buxmann. "Outcomes and success factors of enterprise IT architecture management: empirical insight from the international financial services industry". In: *European Journal of Information Systems* 20.2 (2011), pp. 168–185.

[417] R. Schmidt. "Managing Delphi Surveys Using Nonparametric Statistical Techniques". In: *Decision Sciences* 28.3 (1997), pp. 763–774.

[418] R. Schmidt et al. "Identifying software project risks: An international Delphi study". In: *Journal of Management Information Systems* 17.4 (2001), pp. 5–36.

[419] K. Schneider et al. "Enhancing security requirements engineering by organizational learning". In: *Requirements Engineering* 17.1 (2012), pp. 35–56.

[420] D. Schön. *Educating the reflective practitioner: Toward a new design for teaching and learning in the professions*. Jossey-Bass, 1987.

[421] K. Schwaber. *Agile Project Management With Scrum*. Microsoft Press, 2004.

[422] L. O. Seman, R. Hausmann, and E. A. Bezerra. "On the students' perceptions of the knowledge formation when submitted to a Project-Based Learning environment using web applications". In: *Computers & Education* 117 (2018), pp. 16 –30.

[423] M. Shakroum, K. W. Wong, and C. C. Fung. "The influence of Gesture-Based Learning System (GBLS) on Learning Outcomes". In: *Computers & Education* 117 (2018), pp. 75 –101.

[424] M. Shaw and D. Garlan. *Software architecture: perspectives on an emerging discipline*. Prentice Hall Englewood Cliffs, 1996.

[425] R. Simpson and T. Storer. "Formalising Responsibility Modelling for Automatic Analysis". In: *Workshop on Enterprise and Organizational Modeling and Simulation*. Springer. 2015, pp. 125–140.

[426] R. Singh. "International Standard ISO/IEC 12207 software life cycle processes". In: *Software Process Improvement and Practice* 2.1 (1996), pp. 35–50.

[427] R. Slavin. "Cooperative learning". In: *Learning and Cognition in Education* (2011), pp. 160–166.

[428] B. Slife and R. Williams. *What's Behind the Research?: Discovering Hidden Assumptions in the Behavioral Sciences*. Sage, 1995.

[429] c. de Souza and D. Redmiles. "An empirical study of software developers' management of dependencies and changes". In: *International Conference on Software Engineering*. IEEE. 2008, pp. 241–250.

[430] C. de Souza et al. "Supporting collaborative software development through the visualization of socio-technical dependencies". In: *International Conference on Supporting Group Work*. ACM. 2007, pp. 147–156.

[431] G. Soydan and M. Kokar. "An OWL ontology for representing the CMMI-SW model". In: *Workshop on Semantic Web Enabled Software Engineering*. 2006.

[432] D. I. S.p.A. *Payments Service Directive 2 (PSD2): Il nostro approccio*. Tech. rep. Deloitte Consulting, 2016.

[433] C. of the Joint Chiefs of Staff. *Interoperability and Supportability of Information Technology and National Security Systems*. Tech. rep. CJCSI 6212.01E. Department of Defence (United States of America), 2008.

[434] StandishGroup. *The CHAOS report*. 2016. URL: http://www.standishgroup.com/outline.

[435] K. Stanovich. "Matthew effects in reading: Some consequences of individual differences in the acquisition of literacy". In: *Reading Research Quarterly* (1986), pp. 360–407.

[436] J.-P. Steghöfer et al. "Teaching Agile: addressing the conflict between project delivery and application of Agile methods". In: *Proc. 38th Int. Conf. on Software Engineering (ICSE)*. ACM. 2016, pp. 303–312.

[437] L. Sterling, P. Ciancarini, and T. Turnidge. "On the Animation of Not Executable Specifications by Prolog". In: *Int. Journal on Software Engineering and Knowledge Engineering* 6.1 (1996), pp. 63–88.

[438] G. Stobart. *The Expert learner*. McGraw-Hill Education (UK), 2014.

[439] M. Stone. "Cross-validatory choice and assessment of statistical predictions". In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1974), pp. 111–147.

[440] M.-A. Storey et al. "The (r) evolution of social media in software engineering". In: *Future of Software Engineering*. ACM. 2014, pp. 100–116.

[441] A. Strauss and J. M Corbin. *Grounded theory in practice*. Sage, 1997.

[442] G. Succi and M. Ronchetti. "Legal issues regarding software use and reuse within the European Union legislation". In: *Journal of Computing and Information Technology* 4 (Jan. 1996), pp. 179–186.

[443] L. Suchman. *Plans and situated actions: The problem of human-machine communication*. Cambridge University Press, 1987.

[444] T. Sunazuka, M. Azuma, and N. Yamagishi. "Software quality assessment technology". In: *Proc. 8th Int. Conf. on Software Engineering (ICSE)*. ACM/IEEE. 1985, pp. 142–148.

[445] A. Sutcliffe and A. Gregoriades. "Validating functional system requirements with scenarios". In: *International Conference on Requirements Engineering*. IEEE. 2002, pp. 181–188.

[446] J. Sutherland. "Agile can scale: Inventing and reinventing Scrum in five companies". In: *Cutter IT Journal* 14.12 (2001), pp. 5–11.

[447] D. Svanæs and J. Gulliksen. "Understanding the context of design: towards tactical user centered design". In: *Nordic Conference on Human-Computer Interaction*. ACM. 2008, pp. 353–362.

[448] M. Syeed and I. Hammouda. "Socio-technical congruence in OSS projects: Exploring Conway's law in FreeBSD". In: *International Conference on Open Source Systems*. Springer. 2013, pp. 109–126.

[449] M. M. Syeed et al. "Socio-technical congruence in the ruby ecosystem". In: *International Symposium on Open Collaboration*. ACM. 2014, p. 2.

[450] C. Symons. "Function Point Analysis: Difficulties and Improvements". In: *IEEE Transactions on Software Engineering* 14.1 (1988), pp. 2–11.

[451] D. Tamburri et al. "Social debt in software engineering: insights from industry". In: *Journal of Internet Services and Applications* 6.1 (2015), p. 10.

[452] K. Taveter, H. Du, and M. Huhns. "Engineering societal information systems by agent-oriented modeling". In: *Journal of Ambient Intelligence and Smart Environments* 4.3 (2012), pp. 227–252.

[453] J. Taylor. "Designing an organization and an information system for" Central Stores": A study in participative socio-technical analysis and design." In: *Systems Object Solutions* 2.2 (1982), pp. 67–76.

[454] C. P. Team. "Capability Maturity Model® Integration (CMMI), Version 1.1– Continuous Representation". In: (2002).

[455] D. Teichroew. "A survey of languages for stating requirements for computer-based information systems". In: *Proc. Fall Joint Computer Conference*. ACM. 1972, pp. 1203–1224.

[456] C. Thamrongchote and W. Vatanawood. "Business process ontology for defining user story". In: *IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*. Japan, 2016, pp. 1–4.

[457] L. Thomas et al. "Learning styles and performance in the introductory programming sequence". In: *ACM SIGCSE Bulletin*. Vol. 34. ACM. 2002, pp. 33–37.

[458] J. Tian. "Quality-evaluation models and measurements". In: *IEEE Software* 21.3 (2004), pp. 84–91.

[459] E. Trainer, A. Kalyanasundaram, and J. Herbsleb. "e-mentoring for software engineering: a socio-technical perspective". In: *International Conference on Software Engineering*. IEEE. 2017, pp. 107–116.

[460] E. Trist and K. Bamforth. "Some social and psychological consequences of the longwall method of coal-getting: An examination of the psychological situation and defences of a work group in relation to the social structure and technological content of the work system". In: *Human Relations* 4.1 (1951), pp. 3–38.

[461] M. Unterkalmsteiner et al. "Evaluation and measurement of software process improvement–a systematic literature review". In: *IEEE Transactions on Software Engineering* 38.2 (2012), pp. 398–424.

[462] M. Uschold and M. Gruninger. "Ontologies: Principles, methods and applications". In: *The Knowledge Engineering Review* 11.2 (1996), pp. 93–136.

[463] G. Valetto et al. "Using software repositories to investigate socio-technical congruence in development projects". In: *International Workshop on Mining Software Repositories (MSR)*. IEEE. 2007, p. 25.

[464] N. Venkatraman. "IT-enabled business transformation: from automation to business scope redefinition". In: *Sloan Management Review* 35.2 (1994), p. 73.

[465] VersionOne. *11th Annual State of Agile Survey*. 2016. URL: http://stateofagile.versionone.com/..

[466] L. Von Hellens. "Information systems quality versus software quality a discussion from a managerial, an organisational and an engineering viewpoint". In: *Information and Software Technology* 39.12 (1997), pp. 801–808.

[467] A. Vv. *Regulatory Technical Standards on strong customer authentication and secure communication under PSD2*. Final Draft. European Banking Authority/RTS/2017/02, 2017.

[468] Vv.Aa. *Computational Thinking: A Guide for Teachers by the British Computer Society*. 2015. URL: http://community.computingatschool.org.uk/files/6695/original.pdf.

[469] Vv.Aa. *ISTE Standards for Students by the International Society for Technology in Education*. 2016. URL: http://www.iste.org/standards/standards/for-students-2016.

[470] Vv.Aa. *Operational Definition of Computational Thinking by the ACM Computer Science Teachers Association*. 2011.

[471] L. Vygotsky. "Zone of proximal development". In: *Mind in society: The development of higher psychological processes* 5291 (1987), p. 157.

[472] S. Wagner and F. Deissenboeck. "An integrated approach to quality modelling". In: *Proc. 5th Int. Workshop on Software Quality*. IEEE Computer Society. 2007, pp. 1–6.

[473] S. Wagner et al. “Operationalised product quality models and assessment: The Quamoco approach”. In: *Information and Software Technology* 62 (2015), pp. 101–123.

[474] S. Wagner et al. “The Quamoco product quality modelling and assessment approach”. In: *Proc. 34th Int. Conf. on Software Engineering*. ICSE. 2012, pp. 1133–1142.

[475] J. Wang and A. Sarma. “Which bug should I fix: helping new developers onboard a new project”. In: *International Workshop on Cooperative and Human Aspects of Software Engineering*. ACM. 2011, pp. 76–79.

[476] E. P. Weber and A. M. Khademian. “Wicked problems, knowledge challenges, and collaborative capacity builders in network settings”. In: *Public administration review* 68.2 (2008), pp. 334–349.

[477] WEF. *The Future of Jobs: Employment, Skills and Workforce Strategy for the Fourth Industrial Revolution*. 2016. URL: http://www3.weforum.org/docs/WEF_Future_of_Jobs.pdf.

[478] S.-F. Wen and S. Kowalski. “A Case Study: Heartbleed Vulnerability Management and Swedish Municipalities”. In: *International Conference on Human Aspects of Information Security, Privacy, and Trust*. Springer. 2017, pp. 414–431.

[479] S.-H. Wen. “Software security in open source development: A systematic literature review”. In: *Conference of Open Innovations Association (FRUCT)*. IEEE. 2017, pp. 364–373.

[480] M. Wermelinger, Y. Yu, and M. Strohmaier. “Using formal concept analysis to construct and visualise hierarchies of socio-technical relations”. In: *International Conference on Software Engineering*. IEEE. 2009, pp. 327–330.

[481] C. E. Werts, R. L. Linn, and K. G. Jöreskog. “Intraclass reliability estimates: Testing structural assumptions”. In: *Educational and Psychological Measurement* 34.1 (1974), pp. 25–33.

[482] P. White, A. Rowland, and I. Pesis-Katz. “Peer-led team learning model in a graduate-level nursing course”. In: *The Journal of Nursing Education* 51.8 (2012), pp. 471–475.

[483] J. C. Whitehead, P. A. Groothuis, and G. C. Blomquist. “Testing for nonresponse and sample selection bias in contingent valuation: Analysis of a combination phone/mail survey”. In: *Economics Letters* 41.2 (1993), pp. 215–220.

[484] E. Whitworth and R. Biddle. “The social nature of agile teams”. In: *Agile conference (AGILE)*. IEEE. 2007, pp. 26–36.

[485] R. Wieringa et al. “Requirements engineering paper classification and evaluation criteria: a proposal and a discussion”. In: *Requirements Engineering* 11.1 (2006), pp. 102–107.

[486] J. Wing. “Computational thinking”. In: *Communications of the ACM* 49.3 (2006), pp. 33–35.

[487] R. Winter, C. Legner, and K. Fischbach. “Introduction to the special issue on enterprise architecture management”. In: *Information Systems and e-Business Management* 12.1 (2014), pp. 1–4.

[488] B. Wixom and H. Watson. “An empirical investigation of the factors affecting data warehousing success”. In: *MIS Quarterly* (2001), pp. 17–41.

[489] B. H. Wixom and P. A. Todd. "A theoretical integration of user satisfaction and technology acceptance". In: *Information Systems Research* 16.1 (2005), pp. 85–102.

[490] C. Wohlin. "Guidelines for snowballing in systematic literature studies and a replication in software engineering". In: *International Conference on Evaluation and Assessment in Software Engineering*. ACM. 2014, p. 38.

[491] C. Wohlin et al. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[492] E. Wrubel and J. Gross. *Contracting for Agile Software Development in the Department of Defense: An Introduction*. Tech. rep. CMU-SEI-15-TN-06. Software Engineering Institute, Carnegie Mellon University, 2015.

[493] E. Wrubel et al. *Agile Software Teams: How They Engage with Systems Engineering on DoD Acquisition Programs*. Tech. rep. CMU-SEI-14-TN-13. Software Engineering Institute, Carnegie Mellon University, 2014.

[494] A. Yadav et al. "Computational Thinking as an Emerging Competence Domain". In: *Competence-based Vocational and Professional Education*. Ed. by M. Mulder. Vol. 23. Technical and Vocational Education and Training: Issues, Concerns and Prospects. Springer, 2017, pp. 1051–1067.

[495] C. Yang, P. Liang, and P. Avgeriou. "A systematic mapping study on the combination of software architecture and agile development". In: *Journal of Systems and Software* 111 (2016), pp. 157–184.

[496] Y. Ye, Y. Yamamoto, and K. Nakakoji. "A socio-technical framework for supporting programmers". In: *Foundations of Software Engineering*. ACM. 2007, pp. 351–360.

[497] R. T Yeh. "System development as a wicked problem". In: *International Journal of Software Engineering and Knowledge Engineering* 1.02 (1991), pp. 117–130.

[498] C.-P. Yu et al. "The roots of executive information system development risks". In: *Information and Software Technology* 68 (2015), pp. 34–44.

[499] Y. Zhao, J. Dong, and T. Peng. "Ontology classification for semantic-web-based software engineering". In: *IEEE Transactions on Services Computing* 2.4 (2009), pp. 303–317.

[500] Y. Zhao. *World class learners: Educating creative and entrepreneurial students*. Corwin Press, 2012.

[501] M. Zhou and A. Mockus. "Does the initial environment impact the future of developers?" In: *International Conference on Software Engineering*. ACM. 2011, pp. 271–280.

[502] S. Znamenskij. "Effect driven Evolution: Information Systems Architecture for Large Dynamic Organizations". In: *Conference on Manufacturing Modelling, Management, and Control International Federation of Automatic Control*. Vol. 46. 9. Elsevier, 2013, pp. 1061–1066.