Regulation in Software Engineering

by

Maryi Arciniegas-Mendez
B.Sc., Icesi University, 2009

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Regulation in Software Engineering

by

Maryi Arciniegas-Mendez
B.Sc., Icesi University, 2009

Supervisory Committee

Dr. Margaret-Anne Storey, Co-supervisor
(Department of Computer Science)

Dr. Allyson F. Hadwin, Co-supervisor
(Department of Educational Psychology & Leadership Studies)

Supervisory Committee

Dr. Margaret-Anne Storey, Co-supervisor
(Department of Computer Science)

Dr. Allyson F. Hadwin, Co-supervisor
(Department of Educational Psychology & Leadership Studies)

ABSTRACT

Collaboration has become an integral part of software engineering. The widespread availability and adoption of social channels has led to a culture where developers participate and collaborate more frequently with one another. While collaboration in software engineering has been studied extensively, models and frameworks do not adequately capture how development team members "regulate" themselves, one another, and their projects. I borrow the term "regulate" from the learning sciences to refer to mindful processes developers engage in to determine what tasks they need to complete and who should be involved, what their goals are relative to those tasks, how they should meet their goals, what domain knowledge needs to be manipulated, and why they use a particular approach or tool.

This research starts by borrowing constructs from the theory of regulated learning in the learning science domain, adapting and extending them as a model of collaboration for software engineering: the Model of Regulation. This model was composed to capture how individuals self-regulate their tasks, knowledge and motivation, how they regulate one another, and how they achieve a shared understanding of project goals and required tasks. The model provides a vocabulary for comparing and analyzing collaboration tools and processes. In this thesis, I present the Model of Regulation as a new and complementary theoretical model of collaboration for software engineering and showcase its potential by using the model to analyze features of a collaborative tool, gain insights into an open-source software development community and to create an instrument that investigates about collaboration practices and tool support in units of collaboration (e.g., group, project, community).

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank:

Geovany Trejos, for believing in me, for his unconditional support to my ideas and for sharing his positive energy that always kept me going in the tough times.

My supervisors, Dr. Margaret-Anne Storey and Dr. Allyson F. Hadwin, for giving me the opportunity to join amazing research labs, for their support, for their enthusiasm and for their on-time suggestions and advices that helped me improve this work.

Alexey Zagalsky, for his patience, for his support and guidance, and for his always disposition to help.

Lorena Castaneda, Carlos Gomez, Tania Ferman, Carly Lebeuf for their friendship, support and the talks in the right moment.

Members of the CHISEL lab, for sharing their knowledge with me and for their thoughtful feedback in my research adventure.

Members of the TIE lab, for helping me understand the secrets of the learning sciences, offer me a different perspective to understand my research and for their friendly companion.

Dedication

To my family, you live in my hearth and in everything I do.

Chapter 1

Introduction

Software engineering is a highly collaborative activity that involves the shared expertise and effort of multiple stakeholders. In general, collaboration refers to an endeavor where different people come together to produce something better than any participant could conceive of or produce alone [59]. And while software development emphasizes the production of software assets, it is necessary to also recognize that the way individuals and teams *learn* to create, capture, collaborate, and manipulate domain artifacts (such as source code or specialized development tools) is an important contribution in its own right. The formation of communities of practice characterized by knowledge sharing and learning, and the emergence of socially enabled tools and channels in the social era has led to a paradigm shift in software development [98] and the creation of a highly tuned participatory culture. Thus, modern software engineering is collaborative, global, and large in scale (e.g., the *Ruby on Rails* project has more than 2700 contributors[1]).

The tools used in software development can significantly impact the success of a project and influences how developers learn and work together. Practitioners and researchers have proposed the creation of or improvements to novel tools such as task

[1] `https://github.com/rails/rails`

trackers, configuration management tools, and lightweight messaging tools [64, 44]. Recent studies [99] have found that developers use a rich and complex constellation of communication and social tools in addition to their development tools. However, some researchers argue that the tools developers use still do not adequately support collaboration—collaboration features have been added to many tools as complementary components when they should be at the core of the tool's functionality [28].

Gaining an understanding of collaboration has been a long standing challenge in software engineering practice and research. Over the years, several theories, frameworks and models of collaboration have emerged with the purpose of improving the productivity of individuals and teams (e.g., [31], [51], [2], [66]). Although each of these approached adopts a different perspective to understand collaboration, they do not adequately capture how development team members "regulate" themselves, one another, and their projects. I borrow the term "regulate" from the learning sciences to refer to mindful processes developers engage in to determine what tasks they need to complete and who should be involved, what their goals are relative to those tasks, how they should meet their goals, what domain knowledge needs to be manipulated, and why they use a particular approach or tool. An overview of current models of collaboration in software engineering is presented in Section 3.4

Interestingly, the evolution of collaboration in software engineering into knowledge building communities allow us to identify developers as active learners and explore theories from the learning science domain. In this work, I emphasize that software engineering involves *dynamic informal learning* where participants, guided by their various interests, engage in task coordination and the co-construction of knowledge—how this multidimensional phenomena takes place needs to be investigated. Building on theories of self-regulated learning (SRL) researchers in the learning science domain have explored social modes of regulation, leading to the theory

of regulated learning in collaborative settings. Using the key concept of *Regulation* as a metacognitive base that allows for strategic decisions in the face of change, the theory of regulated learning describes how learners appropriate knowledge by individual processes, interactions with their social context, and group discussions. In this thesis, my research goal is to explore how a model based on the principle of regulation can be used to understand collaboration in software engineering.

To create a model of collaboration based on the principle of regulation, I borrowed some of the constructs defined within the theory of regulated learning [46, 45, 112, 113], extending and adapting the social modes of regulation used in the learning sciences to theoretically model and explain collaboration in software engineering environments. The result is the Model of Regulation for software engineering. Further, by using the model, (a) I describe the affordances and constraints for collaboration in GitHub (see Section 4.3 Chapter 4), (b) described collaboration practices of the open source community created by the Neo4J project (see Chapter 5), and (c) created an instrument to profile collaboration (see Chapter 6).

The Model of Regulation is a descriptive model that allow us to capture how individuals self-regulate their tasks, knowledge and motivation, how they regulate one another, and how they achieve a shared understanding of project goals and required tasks. Further, this model brings awareness about regulation activities, reveals insights on tool needs, and provides a shared vocabulary for researchers and practitioners that was previously lacking.

In this thesis, I extend existing research by (1) introducing a new descriptive model of collaboration: The Model of Regulation, which adds new vocabulary to describe collaboration processes and analyze tool support. Also, (2) I use the vocabulary of the model to describe the support for collaboration provided by the software development tool GitHub. Later, (3) I use the Model of Regulation to describe collaboration and

interaction in the open source community created by the Neo4J project and (4) use the model to develop an instrument that allows one to profile collaboration practices and tool support for any unit of collaboration.

This thesis is organized as follows. The methodology used in this work is documented in Chapter 2. Chapter 3 presents an overview of the main models and frameworks of collaboration, and explores the formation of communities of practice in the domain and as well as knowledge building communities. Chapter 4 introduces the background on the theory of regulated learning from the learning science domain, presents the Model of Regulation for software engineering and describes how it can be used to describe tool support by presenting an analysis of GitHub features. Chapter 5 presents a case study on the open source software development community of Neo4J and illustrates how the Model of Regulation can be used to analyze collaboration in the community and the regulation of the users. Chapter 6 introduces the instrument developed based on the model to assist in investigating collaboration practices of individuals and larger units of collaboration and presents the results from applying it to two software projects. Finally, future work is presented in Chapter 7.

Chapter 2

Methodology

The methodology used for the research presented in this thesis consisted of 5 major steps. As an initial goal, I wanted to evaluate the benefits to collaboration brought by collaborative tools. For that, (1) I conducted a literature review on current models and frameworks of collaboration in software engineering. This review revealed limitations on the current approaches as none of them had the mechanisms to allow me to reach my initial goal (see Chapter 3). This finding motivated me to look to other knowledge domains for other models to describe collaboration as well as evaluate tool support.

As a result, I examined the learning science domain where collaboration has been studied extensively. In particular, I studied the theory of regulated learning which describes strategic actions in the face of change, as well as the work of Hadwin and colleagues on the regulated learning theory in collaborative contexts [46, 45, 112, 113]. Then, (2) I built on Hadwin and colleagues modes of regulation, extending and refining them for software engineering to produce: *The Model of Regulation* (see Chapter 4).

To explore how this model could be used to describe collaborative tools in terms of their benefits to collaboration, (3) I conducted an analysis of GitHub. (4) I then conducted a case study on an open source software development community, and

using the model's affordances, explored how collaboration was experienced. Finally, (5) I developed an instrument based on the model to profile collaboration on software development projects.

It is important to note that my interpretation of the model for software engineering was iteratively improved throughout these studies, leading to a rich descriptive model for productive collaboration. A summary of the methodology is presented in Figure 2.1.

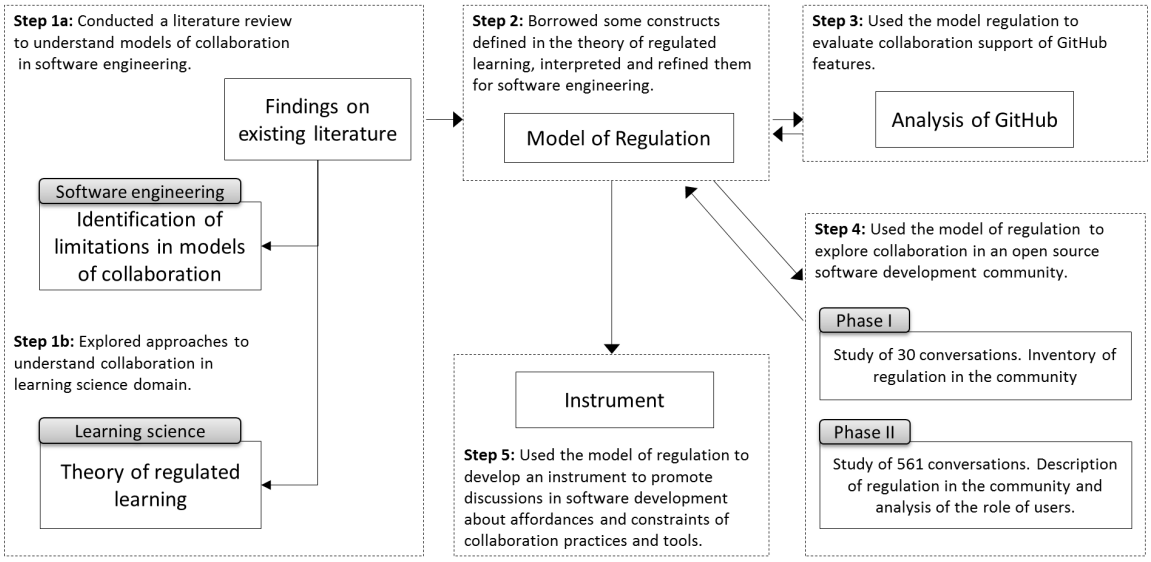

Figure 2.1: Methodology Overview. Each step of the methodology is presented with a short description and the main findings or outcomes (inner boxes).

2.1 Research questions and objective

The purpose of this thesis is to **explore how the model of regulation can be used to understand collaboration in software engineering**. My first research question is:

RQ1: Can the model of regulation be used to evaluate collaborative tools in terms of their benefits to collaboration?

To address RQ1, I analyzed some of the most used features of a collaborative development platform —GitHub—, using the model's vocabulary to describe the collaboration support offered by the tool (see Chapter 4.3). This analysis motivated further research and defined my next research question:

RQ2: How can the model of regulation be used to explain collaboration in open source software development communities?

To answer RQ2, I conducted a two-phase case study on the Neo4J open source software development community. In the first phase (exploratory), I collected a random sample of 30 conversations to help me refine my interpretation of the model's constructs for the software engineering domain. In the second phase (descriptive), I collected 561 conversations.This phase produced more and richer data which allowed me to further refine the model (see Chapter 5).

Finally, my last research question is:

RQ3: Can the model of regulation be used to promote discussions in software engineering about constraints and affordances of collaboration practices and tool support?

To answer RQ3, I developed a questionnaire based on the model of regulation, used it as a guideline for conducting interviews to developers currently involved in collaborative software development and analyzed and compared their answers (see Chapter 6).

2.2 Literature review

My understanding of collaboration in software engineering began with an extensive literature review of the existing models and frameworks within the domain, including the approaches from Computer-Supported Collaborative Work (CSCW) research,

Global Software Development (GSD), and Open Source Software (OSS). I reviewed the state of the art of collaborative software development and tool support. I explored concepts like *communities of practice* and *knowledge building communities* and interpreted them as a consequence of the participatory culture experienced in modern software engineering. In particular, my literature review allowed me to understand how this participatory culture changes the way collaboration happens today. Key findings from my literature review are explored in Chapter 3.

Findings from my literature review in software engineering showed current models and frameworks do not adequately capture how development team members determine what tasks they need to complete and who should be involved, what their goals are relative to those tasks, how they should meet their goals, what domain knowledge needs to be manipulated, and why they use a particular approach or tool. Furthermore, it motivated me to explore the learning science domain, which has also explored challenges in collaboration. Specifically, a review of the literature on collaborative learning from the field of the learning sciences is presented in Chapter 4. The goal of this chapter is to reveal the relevance of this literature for understanding collaboration in modern software development.

Chapter 3

Collaboration in software engineering

This chapter clarifies what is considered theory, framework and model within this thesis, presents the evolution of collaborative software development research, and overviews the main models and frameworks of collaboration relevant to software engineering. I describe modern collaborative software development and how it has created communities of practice in the domain. I also explore the concept of *communities of practice* and *knowledge building communities*.

3.1 Definition of relevant concepts

What constitutes a *theory*, a *model* or a *framework* in software engineering research is still in discussion [92], so this section briefly describes how those concepts are considered in this thesis.

In this work, a theory is considered as a mechanism by which one can generalize a phenomena [92]. *"A scientific theory identifies and defines a set of phenomena, and makes assertions about the nature of these phenomena and the relationships between them"* [29]. Moreover, theories define specific terms that allow researchers to organize,

structure, observe and measure facts and knowledge [29, 92].

A model is at a less abstract level than a theory and represents a set of variables connected through a set of relationships that describe a phenomena and suggest causality [74]. Models are considered to support theories by describing actions and expected outcomes [62]. Finally, frameworks are mechanisms that *"encapsulate and contain the explanation of the phenomena, issue or problem that is the focus of study"* [70]. Considering these definitions the terms 'model' and 'framework' are used interchangeably throughout this thesis.

3.2 Models and frameworks of collaboration in software engineering

In general, collaboration refers to an endeavor where different people come together to produce something better than any participant could conceive of or produce alone [59]. Although there have been many attempts to define collaboration, understanding collaboration and how it can be improved with processes and tools has been a long-standing challenge in software engineering practice and research. This is not surprising given the different contexts and viewpoints that can be used. MoCA [66] is one conceptual framework for characterizing the underlying context where a collaboration takes place, however, this focus on understanding the context (such as size of the group) does not bring insights on how a collaboration occurs nor how it can be improved.

In the following, I discuss some of the more popular models that have been or could be applied to the software engineering domain with the purpose of describing and improving collaboration. I review models that are technology-centric, followed by a review of models that focus on characterizing the collaboration processes and

participants involved. I then propose how the Model of Regulation can complement the existing models and why it is an important contribution.

3.2.1 Technology-centric approaches

Collaboration and its tool support has been studied extensively. The study of Computer-Supported Cooperative Work (CSCW) is an active research field with beginnings in 1984 [42]. Current CSCW research focuses on topics that explore both technologies and practices in collaborative settings. However, in the early days of CSCW, the emphasis was on tools needed to support collaboration rather than on understanding practices or developing theories [7]. Thus, it is not surprising that the first models were focused on classifying systems and tool features. In the beginnings of CSCW, the term *groupware* was coined and frequently used to describe collaborative technologies [31].

One of the most largely used taxonomies to classify groupware, which was proposed by the CSCW research community, is Johansen's time-space matrix [57]. He proposed that all collaboration technologies could be classified on two dimensions: time and space. The space dimension can be *collocated* or *distributed*, while the time dimension can represent *synchronous* or *asynchronous* communication. This matrix was later considered incomplete and some attempts were made to refine it. For example, Grudin [42] added one more state to the original dimensions and allowed for *unpredictable* time and space, while Dix *et al.* [25] expanded on the space dimension by proposing four states: *concurrent synchronized*, *mixed*, *serial*, and *unsynchronized*.

In the software engineering domain, Cook's 2004 work [16] reviewed existing collaboration tools at that time and classified them based on their support for *design*, *development*, *management*, and *inspection*. In 2007, Whitehead [110] introduced a tool classification with four categories: *model-based collaboration tools* (for creation

of models and artifacts for the different phases of the software development life cycle), *process support tools* (process-centered software development), *awareness tools* (informal information), and *collaboration infrastructure or collaborative development environments*. Similar to its predecessors, this framework emphasizes the function or utility of tools without considering the theoretical ground to evaluate the tool's true support for collaboration. Two years later, Martignoni [71], grouped collaboration tools according to *development tools*, *integrated development environments*, *software as a service*, and *consulting and services*, again emphasizing on the tool's functional role.

Many other researchers have also proposed the creation of or improvements to novel tools that support collaboration in software engineering—task trackers, configuration management tools, collaborative IDEs, and lightweight messaging tools are just some of the innovations that have emerged to support distributed development [64, 44].

Striving to understand collaboration through required tools was a good initial step, however, collaboration is a complex activity that cannot be improved by just upgrading technologies. It is not possible to achieve productive results in collaboration when what collaboration consist of is still undefined. Moreover, as software development became more distributed and dynamic, the community experienced a proliferation of technologies. Today, we have such a vast array of computer-based tools that it is hard to list and even more challenging to classify these tools given their hybrid and changing nature. Nowadays, understanding how developers collaborate by considering the tools they use is similar to trying to understand how a chef cooks by reviewing the tools used in a modern high-end kitchen.

The inadequacy of using technology-centric approaches to understand collaboration was noticed by researchers who then tried to understand collaboration from a

more process-centric rather than tool-centric approach. I describe these models next.

3.2.2 Towards characterizing collaboration processes and participants

As far back as 1991, Ellis *et al.* [31] investigated computer-supported group interactions and suggested three key areas that require attention when studying collaborative work: *communication*, *collaboration*, and *coordination*. Their analysis provided the basis for a widely used framework: the 3C Model. In this model, *communication* refers to the exchange of knowledge within a group and allows for the coordination of group tasks. *Coordination* refers to the awareness of and agreements made regarding tasks to be completed through team interactions, as well as any overhead (e.g., planning) that is necessary for the *collaboration* effort itself [31]. Although the model has been largely used to evaluate collaboration tools in software development (e.g., [110, 76, 97]), it represents one of the first approaches drawn from an analysis of collaboration processes.

Later, Gerosa *et al.* [34] suggested that *awareness* be added to the 3C Model. Awareness, as defined by Dourish and Bellotti [27], is *"an understanding of the activities of others, which provides a context for your own activity"*. Indeed, in software engineering Gutwin *et al.* [43] found that distributed developers need to maintain awareness of one another and the entire team, as well as gather more detailed knowledge of the people they plan to work with. DeSouza and Redmiles [23] present a more recent and complete analysis of the role of *'awareness'* in software development teams.

In software engineering, van der Hoek *et al.* [106] explored the process of coordination in software development teams as being the key for collaboration. They introduced a paradigm of *Continuous Coordination* (CC) for leveraging the benefits

from both formal (process-based) and informal (awareness-based) approaches. The underlying principle is that humans cannot and must not have their method of collaboration dictated, but instead they should be flexibly supported with formal and informal tools that they can use as they see fit. The work of DeFranco-Tommarello and Deek [24] goes further and analyzes group cognition and collaborative problem solving processes among developers using concepts from the social sciences. Robillard and Robillard [84] adopted another perspective to study collaboration and identified four types of work: *mandatory collaborative activities* (scheduled formal meetings), *called collaborative activities* (formal meetings to discuss solutions to particular problems), *ad-hoc collaborative activities* (all members simultaneously work on the same task), and *individual activities*. Further research on this classification has focused on ad-hoc collaboration [14, 15].

Cataldo *et al.* [11] emphasized the importance of task dependencies mirroring social dependencies in software projects (the need for socio-technical congruence). Sarma *et al.* [87] took a deeper look at collaboration and presented an adaptation of Maslow's motivational theory to describe the needs of developers. Their framework illustrates the hierarchical needs of software developers and set the basis for a classification system for collaborative tools.

Research thus far has explored collaboration while adopting different perspectives that identify the activities and processes required for successful collaboration (e.g., cooperation, awareness, coordination). However, the way by which those activities and processes are linked and activated in a collaborative activity or supported by tools is unclear.

Other research has focused on analyzing the processes to extrapolate *"principles"* aimed to improve software development practice and collaboration, —in fact, the model presented in this work is of this nature. One example of this research is

presented by the Agile manifesto [2, 32], whose methodology focuses on improving communication among all participants of the collaboration, and includes an indirect positive effect on coordination. As a result of following the principles described in the agile manifesto, agile teams are described as "self-organized units" able to adapt to changing contexts. The methodology has gained popularity among practitioners such that it has been customized in Agile methods for multiple scenarios, for example: Scrum, eXtreme Programming (XP), Crystal, Feature Driven Development (FDD), Dynamic Software Development Method (DSDM), and Adaptive Software Development. An important contribution of the methodology is that it promotes support for collaboration from the human perspective rather than from the tool perspective, however, the methodology misses out the important role of the individual and focuses mainly on group actions. Also, some researchers have developed their work based on Agile event driven environments. For example, Verginadis *et al.* [107] introduced the concept of collaboration pattern (Cpat) as a prescription to address a particular collaborative problem. In this work, researchers propose an architecture defined in terms of rules that suggests Cpats as a mechanism to facilitate collaboration in virtual organizations.

In a similar research, Watts Humphrey developed the Personal Software Process (PSP) to provide structure and increase quality in the individual work of a developer [49, 50]. Later, PSP skills became the foundation for Team Software Process (TSP), an adaptation of the framework aimed to provide guidance for development teams while keeping high productivity and quality [51]. One of the main strengths of this methodology is that it provides structured support for the development activity along with a large list of methods for quantifying and analyzing performance. However, the methodology and its documentation is too extensive and requires developers be trained and certified at both levels: first at PSP and later at TSP. What's more,

the methodology is unsuitable for most cases when one considers the time costs involved and the fact that all members of the team must be certified on both levels in order to implement TSP.

3.3 Motivation for using a "Regulation lens" to study collaboration in software development

Collaboration is a complex activity that involves multiple components. To describe and analyze collaboration, we must study the *tasks* involved (what is being done), the *participants* in the activity (who is interacting), the *methods and strategies* used to achieve the goals (how participants collaborate), and the *knowledge* that is manipulated, refined, and created by the collaborative activity. Moreover, we need to understand the *motivations* that drive the participants' intent (why participants do what they do).

Studying only a subset of these collaboration components is not comprehensive enough to understand the phenomena. For example, the Transactive Memory System (TMS) approach can be used to explain how people who perform activities together (e.g., family members, teammates, organizations) create a shared store of knowledge [108]. TMS effectively helps one make sense of how teams manipulate and create knowledge [118], [68], however, it emphasizes the cognitive aspects at the expense of other components of collaboration, such as the motivation for using certain tools or practices. Likewise, the 3C Model, the Continuous Coordination framework, and the PSP and TSP approaches presented earlier emphasize activities, methods, and participants, but do not study the knowledge shared and the participants' motivations. Similarly, other models of collaboration focus only on a subset of components, arriving at a partial understanding of collaboration.

One important aspect of collaboration that is not sufficiently considered by other models is *learning*, both in terms of the assets to be created and how tasks and collaboration activities are conducted. Learning—or the appropriation of knowledge—is enabled not only by *formal learning* where specific goals are set (e.g., taking courses, reading books), but also by *informal learning* where knowledge is acquired through impromptu processes and ongoing participation (e.g., learning by experience). Moreover, learning facilitates the emergence of regulation as it is through our past experiences and interpretation of social context that we are able to foresee the results of our current actions and make strategic decisions to achieve our goals. Software engineering projects are a particularly salient context for the emergence of regulation because they often involve multiple distributed stakeholders, their evolution is fluid and flexible, and they challenge conventional notions of a fixed or externally defined goal or product. For this reason, regulation is essential for success when managing multiple goals, integrating code development in cohesive ways, documenting intent and explaining progress, and maintaining awareness of personal and collective activity.

I propose that concepts from the learning sciences' theory of regulated learning offer a new and promising perspective for collaboration in software engineering. Regulation is multifaceted as it not only refers to the *tasks* involved and their *participants*, but it also refers to the *methods and strategies* used, the *knowledge* manipulated, and the *motivations* for the participants' actions. I argue that it is important to understand how individuals and teams regulate—plan, monitor, evaluate, and adapt—their activities in order to improve their processes and tools. Moreover, our studies of collaborative software development have shown that these models are inadequate for understanding how individuals and teams regulate and monitor each other's activities in a project. In order to adapt or define a model for current practices of collaborative software development, we need to first understand the key characteristics of modern

collaborative software development and their participants.

3.4 Modern collaborative software development

Modern software development is a social and collaborative process [65]. The recurrent collaborative activity, fueled by the need for different skills and perspectives to solving problems, has led to the formation of *communities of practice* [73] around diverse topics in software engineering.

Communities of practice, empowered by the *"inexpensive and low barriers to publish, as well as the rapidly spreading peer-to-peer, large-scale communications made possible by social media"*, have become an extensive part of software engineering [98]. The social structure created by the community supports the co-construction of knowledge for active participants and learning for all members, which today includes not only developers but also users from all domains, and stakeholders. Moreover, these communities are a source for consulting on and distributing software development practices, tools, and resources.

The widespread adoption of socially enabled tools and channels —characteristics of this social era— facilitates virtual communities of practice and empowers global software teams. Consequently, the availability of additional communication channels (e.g., micro-blogging) and a broad catalog of tools has increased the participatory culture among software developers and all participants of the community. More importantly, the diversity of these tools and features poses a challenge in understanding which may be the best composition or combination of tools that will enhance the *productivity* of collaborating developers and participants and motivate broader *participation*. However, models of collaboration were proposed before this social era and fail to provide an understanding of the broad collaboration experienced in this highly

dynamic scenario of modern software development.

There are two particular areas of software engineering that have flourished in this modern context: global software development and open-source software development. Next, the most relevant research in these areas is presented.

Global software development: As collaboration is especially challenging in distributed teams, any analysis needs to add into consideration constraints like timezone, culture, and language. In Global Software Development (GSD) there have been some efforts to create models and frameworks for this particular scenario. For example, Wiredu [114] adapted a conceptual framework from the organizational behavior domain, proposing four core dimensions to software development organizations: *process*, *people*, *information* and *technology*. The result is a theoretical framework aimed to provide a foundation for research on the coordination aspects of GSD. Another approach is presented by Redmiles *et al.* [83], who adapted the Continuous Coordination design principles to investigate how to improve and integrate formal and informal coordination mechanisms in software project tools. The work of Dafoulas *et al.* [21] identified the main factors that affect communication in distributed teams and illustrated how they are defined in terms of two variables: task and team complexity. Although this work was conducted with two pilot studies in simulated scenarios, the results are still relevant to the analysis of collaboration in software engineering. Also, Olson and Olson [77] present a list of recommendations for distributed teams aimed at three levels: the individual, the manager, and the organization.

Open source software development: In this case, researchers from the company VA Software, based on their experience enabling collaborative development within the SourceForge platform, used principles to describe the nature of collaboration in Open Source Software (OSS) projects. Then, they used these principles to identify common weaknesses in commercial projects that prevent the achievement of orga-

nizational goals [6]. Crowston *et al.* [18] investigated how to improve work team effectiveness in free and OSS projects through the application of Continuous Coordination principles [106] and the alignment of task dependencies, team structures, resources, tool support, and actors. In addition, Crowston *et al.* emphasized how additional work is sometimes required to deal with coordination issues. Also, through the study of four OSS communities, Scacchi [88] presented a list of eight kinds of informalisms key for developing in OSS and showed how these are different from traditional software development. Further, studies on free and OSS projects described the development process [89], as well as studied conflict resolution and strategies for collaboration promotion in the GNU enterprise project [30] and the Netbeans.org community [55].

3.5 Knowledge building communities

Communities of practice have been the focus of study in many domains. In the learning sciences, researchers refer to a *Knowledge Building Community* as a socio-constructivist pedagogical strategy *"that supports discourse and aims to advance the knowledge of the members collectively while still encouraging individual growth that will produce new experts and extend expertise within the community's domain"* [1]. The software developer community is a prime example of such a knowledge building community, and not surprisingly, its users are also part of this group.

In this world of rapid technological change, developers are required to demonstrate active learning skills and their connections to the communities is what allows them to stay up to date and informed of the stakeholder's current needs.

In this scenario of modern software development, research into learning in collaboration and its technological support from the learning science domain becomes

[1] `http://edutechwiki.unige.ch/en/Knowledge-building_community_model`

significantly more relevant for software engineering.

Researchers from the learning sciences developed a theory of regulated learning that includes considerations for learning while working in collaboration. The central piece of the theory is a concept called *Regulation*, which refers to individual and group strategic responses in the face of challenges. I borrowed some of the constructs of this theory of regulated learning as a base for an alternative framework for understanding collaboration. This new model is presented in the next chapter.

Chapter 4

The Model of Regulation

This chapter introduces the theory of regulated learning from the learning science domain. It describes the original constructs that underpin my collaboration model and presents the Model of Regulation for software engineering.

4.1 The theory of regulated learning from the learning science domain

The theory of *regulated learning* is an approach that *"seeks to explain how people improve their performance using a systematic or regular method of learning"* [123]. This theory models the learner's strategic control of their resources and studies their adaptive skills towards continuous improvement of their learning mechanisms.

In the learning science domain, early research considered self-regulated learning as a new approach to describe *"how students activate, alter, and sustain their learning practices using a variety of self-related processes"* [120]. And more recent approaches describe *Regulation* as an individual's strategic response to perceived challenges resulting from changes to their activities or when their goals and performance are misaligned [46, 45, 112, 113]. A strategic response involves monitoring progress, eval-

uating results, and adapting strategies if outcomes are not aligned with specific goals. Learners who engage in this regulatory practice become self-regulated learners [121]. Objects of regulation include behavior (e.g., strategic action, goals and plans), motivation (e.g., user motivation), cognition (e.g., task knowledge, self-knowledge, strategy knowledge) and emotion.

Early theories of regulated learning focused mainly on the strategic responses for individual learning (self-regulated learning, [8], [12], [81], [119], [90]). However, with the shift to learning in more social environments, the importance of other modes of regulation started to emerge. What's more, the recognition of the ability to work and learn from others as a critical skill in the 21st century [82] has made research about regulated learning in collaboration even more relevant. As a result, recent research has also actively studied social modes of regulation: *co-* and *socially shared* regulated learning.

In particular, the work of Hadwin and colleagues [46, 45, 112, 113] on the theory of regulated learning explicitly defined the role of the participant's interaction in the learning process. Her work on regulated learning in collaborative settings reveals insights about collaboration that have not yet been explored or considered by the models used in software engineering.

4.1.1 Hadwin's contributions to the theory of regulated learning

Providing opportunities to collaborate does not necessarily guarantee successful collaboration. Hadwin *et al.* [45] found that a group's collaborative potential may not be fulfilled due to a lack of *regulatory skills* such as time management, an efficient use of resources, and task distribution. They also found that many learners do not effectively regulate their individual and collective activities [52].

According to Hadwin's work, one needs to consider the following aspects of regulation:

1. Regulation is multifaceted—it is not just about behavior/action, it is also about perceptions of behavior, knowing, motivations, climate, etc.

2. Regulation assumes human agency—individuals and team members exercise their capacity to make choices about tasks, situations, and other factors (e.g. teammates). People and teams strive to achieve goals, whether those goals are transparently communicated or not.

3. Regulation is a cyclical adaptation—a set of contingencies (goals, plans, and actions) shift and evolve over time in response to people monitoring and evaluating themselves and others.

4. Regulation draws from our socio-historical past—we are never a blank slate. We bring knowledge and mental models of the task, tools, domain, and team to new work situations, and these beliefs continue to develop over time.

5. Regulation is adaptive—when challenging situations arise, people strive to adjust and optimize success, drawing from a range of internal and external resources (e.g., people, tools, technologies).

6. Regulation is socially situated—it involves a dynamic interplay between tasks, contexts, people, and communities that create and constrain opportunities for planning, doing, and reflecting on what we do.

Hadwin and colleagues identified three modes of regulated learning [46], four recursive processes for regulated learning [112, 113], and suggested how computer-based tools can be used to support regulated learning with individuals and groups [45]. These contributions are presented next.

Modes of regulated learning

In collaborative contexts, multiple modes of regulated learning are required: self-regulated learning, co-regulated learning, and socially shared regulated learning. *Self-regulated learning* (SRL) refers to the strategic actions performed by an individual in order to optimize the results of their own work. For instance, when a student keeps notes of their classes to remember concepts.

Co-regulated learning (CoRL) describes interactions between individuals and the social context in which their perceptions about the objects of regulation are shared. The purpose of co-regulation is to provide temporary metacognitive knowledge support for planning, monitoring, and evaluating, where an individual or group supports or influences the individual regulation processes of one or more members. [78, 47]. For example, when a participant suggests that other group members use a personal weekly planner to help give direction to their work. It should be noted that co-regulated learning can also be temporarily guided or supported (or even thwarted) by tools, technologies, and practices.

Lastly, *socially shared regulated learning* (SSRL), occurs when participants collectively regulate their activities [47] by co-constructing their understandings about tasks, setting shared goals, and defining the strategic use of resources [46]. When a learner suggests the implementation of a different strategy to approach a task and group members reply with comments that contribute to further development of the initiative, the group is experiencing an episode of shared regulation.

Processes of regulated learning

In Winne and Hadwin's model [112, 113], regulated learning unfolds over four recursive phases —these have been extended to include co- and socially shared regulation [46]. Figure 4.1 illustrates the processes of regulated learning, which are

described as follows:

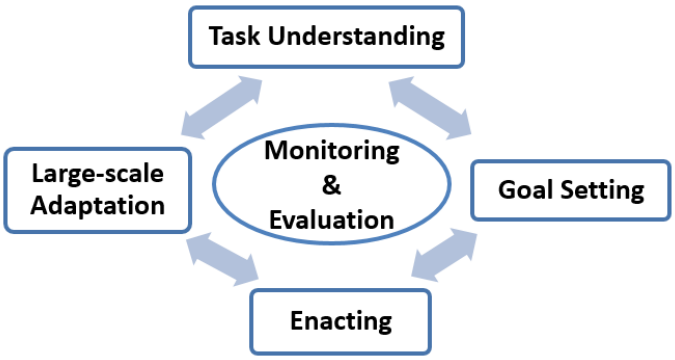

Figure 4.1: Processes of regulation from Hadwin's work

- *Task understanding:* (a) Defining perceptions of task requirements, purpose, and social context; (b) Knowledge and beliefs about one's own strengths and weaknesses and their responsibilities with respect to the task; (c) Reading and interpreting task assignments and discussing how to go about solving the problem [85].
- *Goal setting:* (a) Setting goals, task standards, and plans for approaching and contributing to a joint task (including standards about motivational and emotional states, task processes, and products); (b) Selecting appropriate strategies and allocating resources that affect performance [105].
- *Enacting:* (a) An indicator of strategy use; (b) Purposefully selecting and generating tools and strategies to help attain task goals and standards, maintain motivational engagement, and mediate socio-emotional challenges.
- *Large-scale adaptation:* (a) Making a purposeful change in planning, enacting, etc. when task perceptions, goals, plans, and strategies are not working for the current task or future tasks; (b) Persisting in the face of challenge.

As depicted in Figure 4.2, learners are metacognitively *monitoring and evaluating*

progress throughout the phases. This is central to regulated learning as it is by evaluating their own progress (self-regulation), one another's progress (co-regulation), and shared progress (socially shared regulation) that learners decide when and how to make changes in their learning.

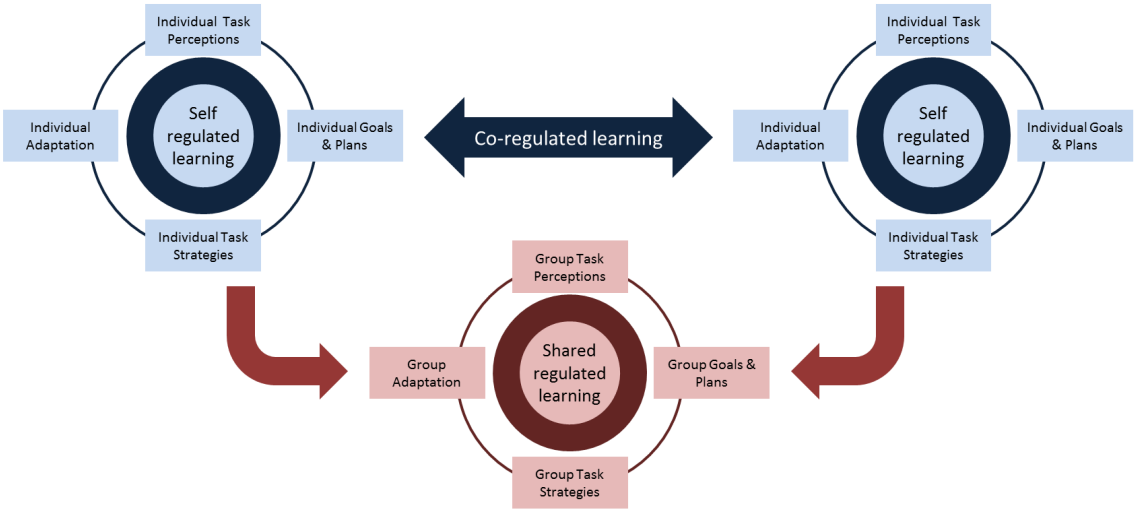

Figure 4.2: Modes and processes of regulated learning. Adapted from the work of Hadwin and colleagues [52].

Tool support for regulated learning

Information and communication systems bring an opportunity to support regulated learning as they have been proven to be useful for enhancing performance, increasing knowledge construction and learning, allowing for equal participation, and stimulating more complex and enriching discussions [53, 58, 61, 48, 95]. However, the lack of regulatory skills prevents the identification of good opportunities for regulation support in these systems [122]. Consequently, recent efforts have focused on highlighting the importance of targeted support for self-regulatory skills, co-regulation, and socially shared regulation [52], as well as the introduction of design principles for computer-based tools [111, 54].

Below I describe the different categories of tool support for regulated learning suggested by Hadwin *et al.* [45]. Note that in this classification system, the word *'tools'* is used in the broader sense to include all mechanisms —not just computer-based— that can provide support for regulation. Also, it should be noted that their contribution to support self-, co-, and socially shared regulation depends on the scope of the information and the way it is presented to the user.

Structuring support refers to approaches that aim to guide interactions by designing or scripting a situation before it begins [94]. Structuring support consists of roles, scripts, and prompts. For instance, instructors can define and assign *roles* to ensure certain activities are carried out (e.g., note taker, data collector, coordinator) [101, 93, 75]. *Scripts* can be designed to help learners even further by describing the specific responsibilities for each role (e.g., the note taker must update the group wiki after each meeting) [109], while *prompts* can provide direction. (e.g., a chat tool can prompt pre-set messages or questions to guide team interaction). Configured by instructors, prompts can ask about the agreements the group has made so far or suggest the team work on a particular task based on the amount of time they have been online.

Mirroring support refers to mechanisms that reflect individual or collective actions by gathering and summarizing data [94]. Depending on the scope of the information presented by the tool, it can promote self-, co- or socially shared regulation. In addition, mirroring support can be achieved with a visualization such as a pie or bar chart that shows individual or aggregated contributions [63, 67].

Awareness tools are similar mirroring support mechanisms, except that they allow for self-comparison or comparisons between group members [9, 10]. A timeline graph is a good example of this kind of tool —this visualization shows comparisons across time or people.

Guiding systems are programs that behave like a virtual presence interpreting data and providing instructions when issues in the collaboration process are detected [94]. For example, a guiding system that shows visualizations of ideas not picked up during a group discussion creates context for shared-regulation.

In previous sections, I described how Hadwin and colleagues, building on theories of self-regulated learning, explored the social modes of regulated learning and defined a new theory that explains the way learners appropriate knowledge in collaborative contexts. In this new theory, *regulation* is the central piece that triggers all processes so that the learner can ultimately acquire new knowledge. Interestingly, the learning science domain not only explores and studies formal learning, like the learning experienced in schools or universities, but also informal learning such as the kind experienced by everyone of us in our daily tasks (e.g., when using a new printer or formatting a table in LaTex for the first time). In this sense, a 'learner' can be identified in many other contexts outside formal education, and a 'theory of learning' can be applied to any context where individuals intentionally plan, enact and evaluate their performance with the purpose of improving their practice and results. In particular, the concept of regulation brings a new perspective for exploring collaboration in software engineering. Some of the constructs of the theory of regulated learning can be refined and extended to compose a model of collaboration for this particular domain.

In this thesis, I borrow the constructs created as the modes of regulation (self-, co- and socially shared regulation) and processes of regulation (Task Understanding, Goal Setting, Enacting and Adaptation) and refine their definitions and relationships for software engineering —this process led to the formation of a Model of Regulation for software engineering. The categorization for tool support (Structuring, Mirroring support, Awareness tools and Guiding systems) is used as is. In the next section,

I review regulation from the software engineering perspective as well as introduce the model and provide examples of regulation support in the software engineering domain.

4.2 The Model of Regulation

Modern software development is by default a collaborative activity that creates communities with demands of strong interactions between participants [3]. In a world of rapid technological change, developers must demonstrate active learning skills and be long-life learners, and as such, they have to rely on their community connections to stay up to date. Likewise, communities act as dynamics organism that adapt to contextual changes; skill is achieved by promoting and supporting knowledge sharing among members.

From the learning science domain, I found the concept of *Regulation* as a way to refer to individual and collective responses in the face of changes, either external (an activity is not giving the expected results) or internal (new ideas to approach the task have emerged). Regulation refers to strategic actions performed by individuals in order to achieve their goals, obtain the expected results, and be more productive in their practice [46, 45, 112, 113]. Certainly, the key for *productive* individuals and teams are their strategic actions as these are expected to lead to desirable results with the minimum amount of time and effort while keeping high quality. For instance, effective time management, efficient use of resources and realistic task distribution.

I stress that the concepts from the theory of regulated learning in the learning science domain offers a new and promising perspective to consider collaboration in software engineering. It is important to note that the appropriation of knowledge is not exclusive to formal learning environments (e.g., school, college)— in fact, in order

to achieve personal and professional goals, learning must be present in every aspect of our practice. In collaborative contexts, besides the technical knowledge we may acquire from the execution of a particular task, we also learn to communicate among each other and discover ways to collectively produce something. What's more, learning is what facilitates the emergence of regulation as it is based on past experiences and our interpretation of the social context that we are able to foresee the results of our current actions and make strategic decisions to achieve our goals. Software engineering projects are particularly salient contexts for the emergence of regulation because they (a) often involve multiple distributed stakeholders, (b) are fluid and flexible in terms of their evolution, and (c) challenge conventional notions of a fixed externally defined goal product. For this reason, successful regulation in terms of managing multiple goals, integrating code development in cohesive ways, documenting intent and progress, and maintaining awareness of personal and collective activity are essential to success.

However, regulated learning, as far as I know, has not been studied in software engineering. Although one can recognize how learning and the regulation promoted by it is inherently present in software engineering communities facilitating change, evolution, and growth, the details of this learning process are unknown. Regulation is present in the interactions of the community participants, but what regulation looks like in this context needs to be investigated in order to create a model of collaboration for the domain.

I conducted three studies that allowed me to identify the particularities of collaboration in software engineering and integrate them into the emerging Model of Regulation:

1. Analysis of GitHub: demonstrates how the Model of Regulation can be used to evaluate and classify computer-based tools intended to support collaboration.

This study was published in 2015 [5] and a complete description is presented in Section 4.3 Chapter 4.

2. Using the Model of regulation to understand user-developer collaboration in a software project: uses the model as a lens to describe collaboration practices between the main stakeholders of a software project (users and developers). Details of this study are presented in Chapter 5.

3. The Model of Regulation in action: presents an instrument, drawn from the model, that can be used to profile collaboration practices and tool support. Details of this study are described in Chapter 6.

Based on the results from the study, in each chapter I highlight what is being refined in the model and its rationale. The Model of Regulation represents a prescription for productive collaboration. It provides vocabulary to refer to different processes and the scope of the strategic decisions required for the collaborative activity, and provides a sequential guide that suggests the order in which these should appear. Next, I present the constructs that create the Model of Regulation for software engineering and their interpretation within this domain.

4.2.1 Modes of regulation from the software engineering perspective

The model describes three modes of regulation: *Self-regulation*, *Co-regulation*, and *Shared regulation*. Together, these three forms of regulation influence the strategic activities that occur when people collaborate.

Self-regulation refers to an individual's processes with respect to a task. For instance, if a developer is trying to fix a bug, one strategic decision is to try to

understand the problem by reading about similar cases from different sources of information. Another example is when a person keeps track of their progress to ensure they finish on time. In a collaborative setting, self-regulation is always required but it is insufficient by itself. In collaboration, participants must make individual efforts towards a common goal—each define their own assessment of the problem at hand and the possible solutions. However, social processes are also required to achieve consensus and unified agreement, refining each other's perceptions and gradually leading to a shared understanding among members.

Co-regulation, when exercised by participants, refers to the recognition of each other's perspectives and the alignment of ideas with respect to the tasks to be completed. However, co-regulation is afforded and constrained by the complete social context including people, tools, technologies, and practices. When collaborating, this mode of regulation indicates how individual strategic responses are modeled, usually by interactions between participants or by interactions with their social context. It should be noted that *'participant'* is used here as a reference to a member of the group regardless of the role or their hierarchy. The purpose of co-regulation is for temporary meta-cognitive support for planning, monitoring, and evaluating, where individual team members or an entire group supports or influences regulation processes for one or more members [78, 47]. For instance, discussions among developers can help detect individual misunderstandings with respect to a task or can help improve individual work plans. Continual dialog between users and developers can prevent the creation of false expectations about a product and keep people updated about the latest use cases.

Finally, *shared regulation* involves joint control of the task through shared (negotiated), iterative fine-tuning of cognitive, behavioral, motivational, and emotional conditions/states as needed. Strategic decisions within this scope are intended to

affect the group as a unit, for example, when a participant suggests the creation of a shared calendar to facilitate meeting scheduling and other members reply with approbatory comments that contribute to further development of the initiative.

The mode of regulation active at a given time is controlled by the desired action. For instance, when the purpose is to affect one's processes, self-regulation is in play. If individual processes remain the target but the action is initiated by other participants or promoted by elements of the social context (e.g., tools, practices), then co-regulation is taking place. If the desire is to realign or adapt joint processes and the action is collectively promoted by members, shared regulation is involved. Furthermore, the three modes of regulation (self-, co- and shared) co-emerge in the collaborative activity, providing and taking feedback from one another, which makes the relationships between them bidirectional (as depicted in Figure 4.3). Good self-regulatory skills contribute to richer experiences in the collaborative context, and co- and shared regulation are social processes that also provide feedback about individual regulation [79].

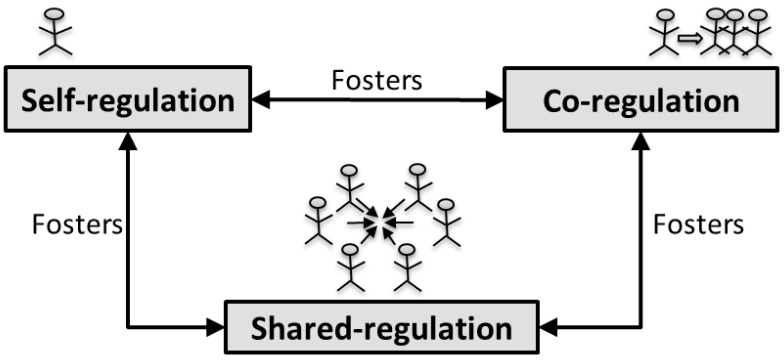

Figure 4.3: Modes of regulation in the Model of Regulation.

It must be noted that regulation is experienced during every aspect of collaboration, so it is likely that participants engage in multiple cycles of regulation during a collaborative activity.

Interestingly, these three strategic activities—or modes of regulation—have already been identified as *'interaction levels'* by some software engineering researchers [44]. Guzzi et al. named them as *individual work*, *coordination* and *collaboration* to represent the activities of an individual developer, interactions between developers, and simultaneous work on the same task, respectively. However, the focus of their study was tool support and the creation of Guzzi's model was a way to explain the levels of support provided by the tool. As far as I know, no further attempt was made to understand the role of these interactions in collaborative software development.

4.2.2 Processes of regulation from the software engineering perspective

The Model of Regulation also incorporates and defines processes related to the strategic decisions required to perform a task. In particular, regulation in software engineering consists of five cyclic processes (c.f. Fig. 4.4)

Task Understanding: As part of this process, individuals define general perceptions of the task at hand and foresee the way participants will contribute to the completion of the task [85].

In the software engineering context, task understanding defines the process by which participants of the collaboration invest time to think about the task requirements, purpose, scope, social context and responsibilities, and roles of participants with respect to the task or project. Together, these represent the comprehension or understanding of a project. More importantly, task understanding in software engineering is constantly influenced and shaped by different stakeholders and external communities. Monetary interests can certainly change the purpose of a project and user feature requests can modify initial requirements. Also, an external community or a company developing or working on a similar product for the same market represent

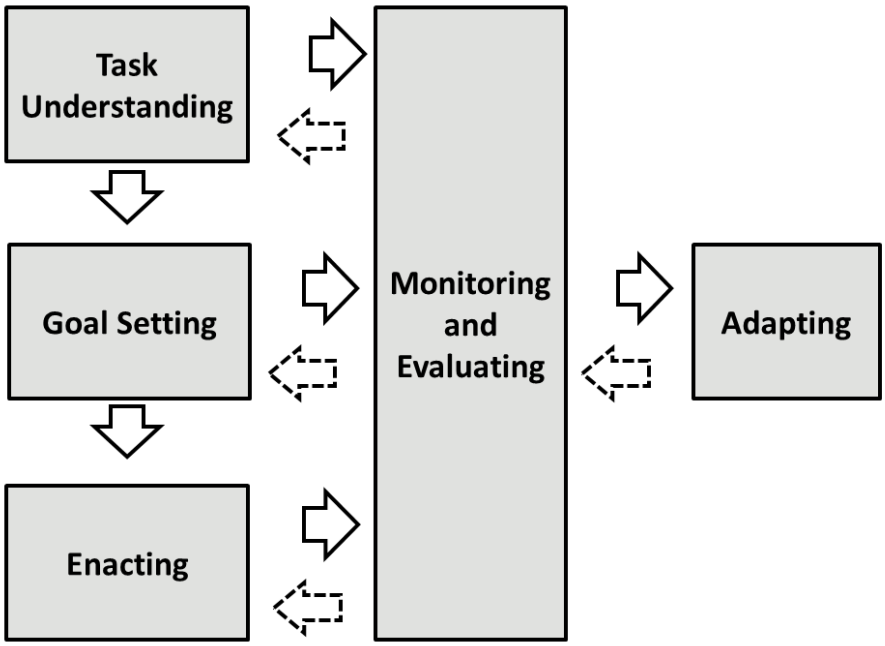

Figure 4.4: Processes of regulation in the Model of Regulation. Solid line arrows represent outcomes going out of the process, while dashed line arrows indicate feedback going into the process.

direct competition that alters the comprehension of one's project.

Goal Setting: This process establishes a work plan, including strategies to motivate and engage participants [105]. In software engineering, a work plan usually documents measurable goals, task standards (e.g., deadlines, acceptable levels of product quality), and resource allocations (technological and human resources). It also covers strategies and tools to support collaboration, including resources to (a) accomplish tasks (e.g., definition of programming languages), (b) acquire information (e.g., project wiki, discussions on virtual channels, books), and (c) facilitate collaboration (e.g. selecting communication technologies, hosting services).

Enacting: With this process, group members follow the guidelines defined in the plan. For example, developers follow code standards like documentation within the code and on the project wiki. Participants make proper use of the established

communication channels and use the selected tool for version control. They also implement strategies to stay motivated and engaged in the task (e.g., reward system). All these parameters must be included in the work plan.

Monitoring and Evaluating: As part of this process, group members monitor performance and preliminary outcomes and compare them against goals to detect misalignments.

The Monitoring and Evaluating progress is an important activity in software engineering and a key to success in many cases. And because of the dynamic nature of software development, developers not only monitor progress, they also constantly scan their environments to ensure project requirements and scope have not changed.

Adapting: As part of this process, group members make purposeful changes in any of the previous processes because their preliminary outcomes are not as expected; the source of the misalignment between the expected and actual outcomes is identified, revised, and adapted. For example, in a software project, a particular programming language may be selected because it offers strong support for creating graphs. However, when the project implementation begins, developers experience problems with the selected language. In this case, a strategic decision about the choice of language is required. They may change the choice of language or determine that they need more training with the selected language. In either case, they require an *adaptation* to their goals (Goal Setting process) or their tasks (Enacting process) accordingly.

It should be noted that the Adaptation process is often not sequential and can happen over any of the regulation processes as required. Furthermore, in software engineering projects, requirements are modified frequently and project scope expands with time. In this context, adaptations also occur when there is a modification to the project understanding in the form of new requirements, which may alter the scope of

the project. In software development, customers requesting new features or changing their minds about others is more often a rule than an exception. As a consequence, adaptations triggered by this type of event require strategic decisions—regulation—about *Task Understanding* in order to integrate the new requests into the previous comprehension of the project. Also, the *Goal Setting* process must be revised to align the work plan with the new goals, and by a cascading effect, the *Enacting* process is also modified.

4.2.3 Activating regulation: how processes and modes of regulation are experienced

So far I have presented the different processes that occur during regulation, but the description does not highlight how one process leads to another during collaborative work. Here I illustrate how each of the processes activates the other processes describe the role of the modes of regulation in this model.

For any given task, *Task Understanding* is executed first as individuals must first understand what they are required to do and what the work implies, regardless of whether the task is given or self-assigned. Note that Task Understanding is influenced by more than one entity including customers, stakeholders and external communities. Just before moving into Goal Setting, the *Monitoring and Evaluating* process is activated to take on the preliminary results of the Task Understanding process. Next, individuals move to define a work plan in *Goal Setting* and the active process of Monitoring and Evaluating ensures the plan is aligned with the project understanding. Then, individuals are ready to execute the plan in *Enacting*. Again, the active process of Monitoring and Evaluating takes the preliminary outcomes of this activity and compares them to the established work plan from Goal Setting and the project understanding result from Task Understanding. If outcomes are not aligned in any

comparison, for example, if the plan is not aligned with the project comprehension or the preliminary outcomes from enacting the task are not aligned with the plan, the Monitoring and Evaluating process will trigger a strategic decision in the *Adapting* process. Finally, the Adapting process provides input for the processes that require revision so an update can be performed.

Now, if the task is to be performed by a single individual, then regulation processes (e.g., Task Understanding, Goal setting) only occur at the individual level: self-regulation. Meanwhile, if the task represents a collaborative activity to be performed by more people, then the regulatory processes would start at the individual level (self-regulation), expand to include participant interactions (co-regulation), and finally are executed within the unit of collaboration (shared regulation). That is, each cycle of regulatory process occurs within the three modes of regulation. Figure 4.5 illustrates the way processes of regulation are activated.

Research in learning science has also suggested a classification system for tools based on their support for regulation —for complete definitions and references see Section 4.1.1. Next, I present short definitions for each category and examples of tools from software engineering for each category in the classification system.

4.2.4 Tool support for regulation: examples from software engineering

The contribution from these categories of tool support to self-, co-, and shared regulation depends on the scope of the information and the way it is presented to the user. For instance, GitHub issues can be used for self-regulation when used to create self-reminders, for co-regulation when they are created and assigned as a to-do for someone else or issues can promote shared regulation when created as an open request for everyone in the group.

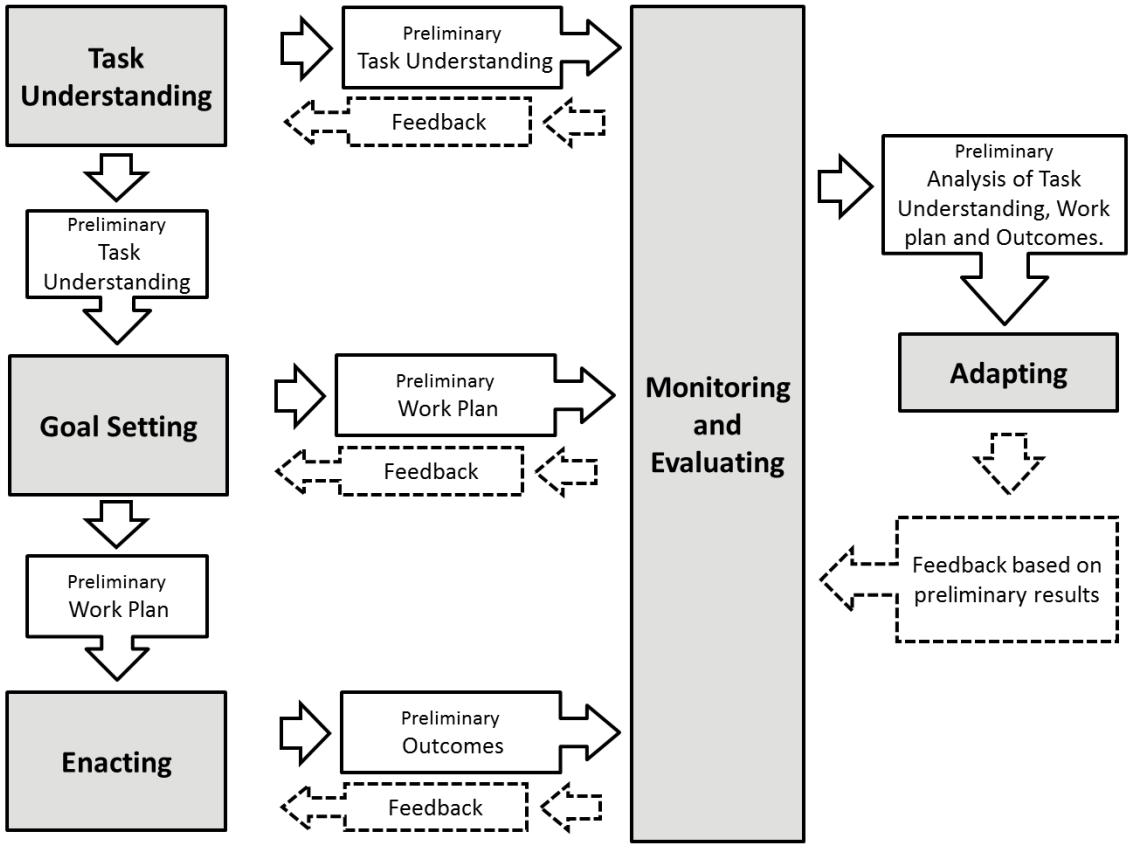

Figure 4.5: Activation of the processes of regulation in the Model of Regulation. Solid line arrows represent outcomes going out of the process, while dashed line arrows indicate feedback going into the process.

The first category of the classification system suggested in the learning science domain is *Sructuring support*, which includes strategies that intend to guide interactions or script a situation before it occurs. Types of structuring support include roles, scripts, and prompts. *Roles* are functions intentionally assigned to each member of the team (e.g., designer, developer, or tester) while *scripts* are lists of steps that suggest the correct order of activities required to complete a bigger task. For example, the life cycle of a software development project may require the following order of general activities: requirements gathering, design, implementation, and evaluation. A script could provide the list of activities with a detailed set of steps for each phase.

Finally, *prompts* are messages that can be delivered to the team to provide hints and suggestions about correct processes for a particular activity. For instance, when code changes are made, a prompt can remind the developer to register the changes in the team's Wiki or notify other team members. An application based on this principle that helps with group management is iDoneThis [1]. A few examples of structuring support are illustrated in Figure 4.6.

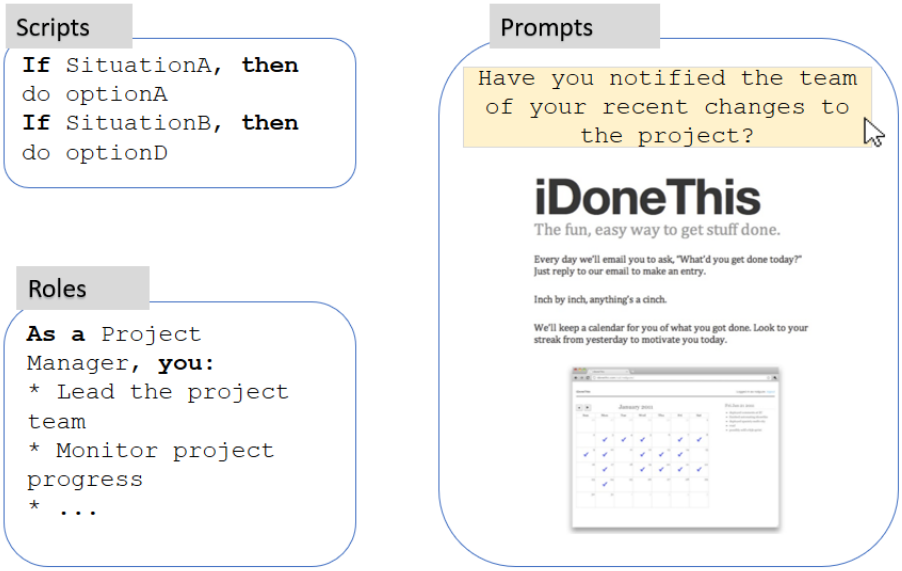

Figure 4.6: Examples of Structuring support

The second category of computer-based tools is *mirroring support*, which refers to mechanisms that reflect individual or collective actions by gathering and summarizing data [94].

An example of mirroring support used for self-regulation can consider individual actions as a summary of activities, with completed tasks distinguished by color or some other cue, like the tasks presented in Trello [2]. Mirroring support can also be achieved with a visualization such as a pie chart or a time line graph that shows

[1] https://home.idonethis.com/
[2] https://trello.com/

aggregated contributions.

Several software development environments provide views to highlight such information (e.g., Trello [3], WakaTime [4], GitHub [5], and Codealike [6] which are illustrated in Fig. 4.7).

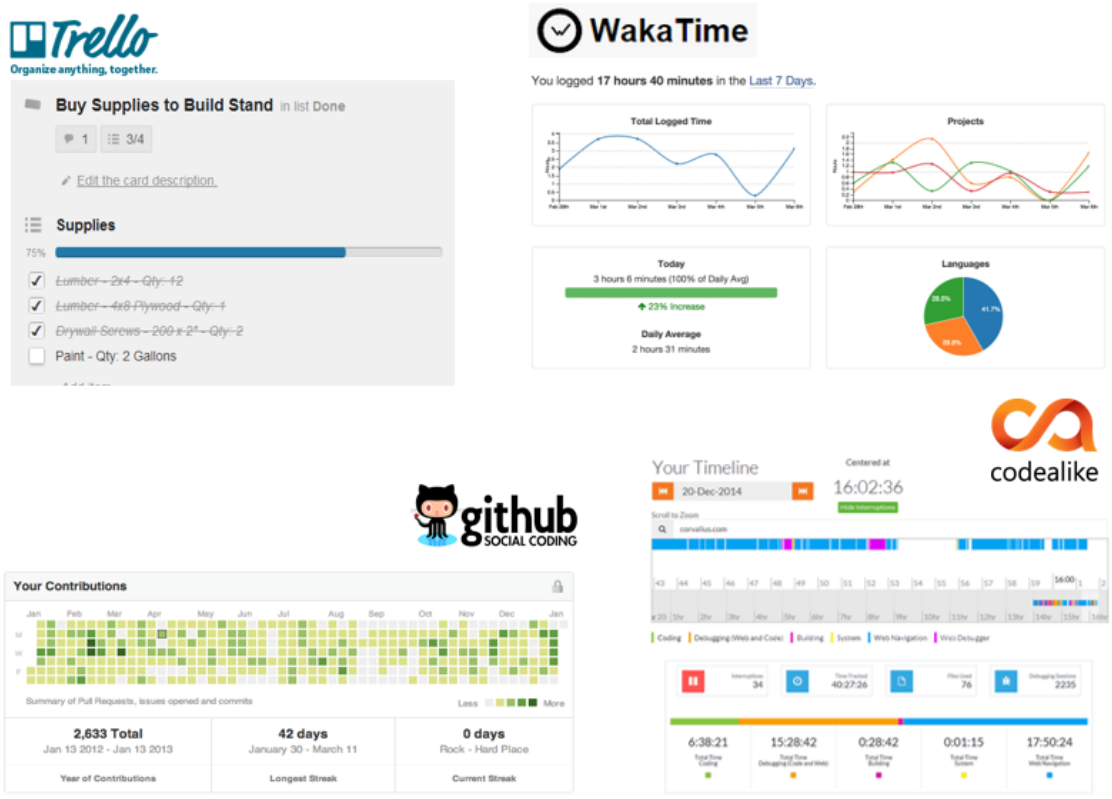

Figure 4.7: Examples of Mirroring support

Awareness tools represent the third category. These tools are similar to mirroring support, except that they allow for self-comparison or comparisons between group members. The scope of the information included within the tool determines whether it promotes self-, co- or shared regulation. For instance, self-regulation is supported

[3] https://trello.com/
[4] https://wakatime.com/
[5] https://github.com/
[6] https://codealike.com/

if the tools allow one to compare individual performance at different points in time. Also, an awareness tool that invites co-regulation will be restricted to comparisons between a small set of members, while a context for shared regulation requires simultaneous performance analysis of the whole group. Potential examples of awareness tools include a visualization presenting individual timeline graphs per contributing member (as shown in Fig. 4.8) or a visualization showing the lines of code introduced by each member.

Figure 4.8: An example of Awareness support provided by GitHub

Finally, *Guiding Systems* include programs that behave like a virtual presence. A guiding system can promote all three modes of regulation depending on the context. For instance, a guiding system that sends reminders of an individual's pending work is helping with self-regulatory processes. If the guiding system provides instructions on a subtask designated for a set of members, it supports co-regulation. Whereas a guiding system that shows visualizations of ideas not picked up during previous group discussion creates context for shared-regulation. Guiding system support in

software engineering is relatively new and is mostly provided in the form of 'bots' or 'conversational bots' integrated into tools used by developers. In fact, the increased adoption of bots by developer teams[7] demonstrates the great potential in guiding collaboration.

Next, I describe how the Model of Regulation can be used to describe the tool support provided by GitHub.

4.3 Using the Model of Regulation to describe tool support

GitHub is a web-based repository hosting service based on Git, an open-source version control system that allows all modifications to be saved with a specific ID in a central repository. This function allows users to save different versions as the work progresses. The platform uses a specific vocabulary for its many artifacts and functions: repository, clone, commit, push and pull are some of those [36]. In GitHub, a *repository* is a project, which can be private (only accessed by owners and contributors) or public (accessed by either registered or unregistered GitHub users). Designed for collaborative work, GitHub allows users to *clone* a remote repository in order to create a local copy for individual work; they can *commit* their changes to the remote repository as needed. Each commit has (a) an unique identifier (provided by the system), (b) a small description about the modifications in the commit (e.g., fixed bug in function createTable), and (c) the list of files affected by the changes.

To upload changes to the remote repository, contributors *push* their commits. Likewise, when participants need to download changes, they *pull* commits from others into their local version. Usually the remote repository merges all contributions,

[7] http://www.wired.com/2015/10/the-most-important-startups-hardest-worker-isnt-a-person/

providing asynchronous collaboration support [80].

GitHub integrates tools often used by software developers, like issue tracking, change logs, wikis, and task lists. In addition, the platform provides a social component that allows each contributor to maintain a profile site with visible statistics about their performance and a list of their recent contributions to public repositories. Participants can follow other members in the GitHub community or watch a project to get updates about its changes [37].

Since GitHub was launched, it has become the largest code-hosting site in the world [60]. Surprisingly, the platform is not used exclusively for developers to share code, which it was initially designed for [35]. Recent research has illustrated that GitHub is used mainly for personal projects and the purpose of the repositories is diverse: users report using it for experimentation, hosting free websites [38], for academic projects or education-related activities [117], and file storage [60]. Researchers have also expressed a particular interest in GitHub because of the open nature of the data. Certainly, the availability of meta-data, through its API or off line GHTorrent project [41], have made the tool very attractive to study developers and project activities, performance, and practices.

Some of the research conducted on GitHub suggests that its social features are the reason for the high number of contributors [72]. Others argue that the social component acts as a facilitator that creates transparency and visibility for project activity, which eventually leads to reduced communication between participants in a collaborative context [20, 102]. Also, research shows that the open environment of collaboration supported by different awareness mechanisms (e.g., pull requests) influences behavior [80], promoting participant engagement and reducing the time required to merge contributions [40]. However, to the best of my knowledge, no analysis has been conducted on GitHub using regulation theory.

4.3.1 Analysis of GitHub features using the Model of Regulation

Using the Model of Regulation, I examine the collaboration support from 5 GitHub features I considered to be the most used: *Issues*, *Graphs*, *News feed*, *Commit* and *Blame* [8]

Issues are defined as work items that *"can be labeled and assigned to a user"*. This definition suggests that the feature offers context primarily for the regulation experienced on participant interactions (co-regulation), but in practice, users use this feature for each of the three modes of regulation. When an Issue is intended for the same member that created it, the feature is used as a self-reminder and supports individual regulation (self-regulation). But if the Issue is assigned as a to-do item for someone else, then co-regulation is supported. Also, issues can be created from a group planning discussion as an open reminder of the group's next steps. This case supports shared regulation as any member has the opportunity to comprehend the collective state of the group. As mentioned previously, how a tool is used and what information it can manage affects the mode of regulation it supports. Regarding the classification of Issues within the categories of regulation tools, I identify it as mirroring support.

Graphs[9] allow team members to visualize individual and collective statistics (e.g., contributor activities, number and time of commits) for a particular repository.

For contributor activities, illustrated in Figure 4.9, the Graphs feature presents two views. The top view shows a graph of aggregated contributions to the project whereby individual work is not identified. In this case, the visualization supports the foundation for shared regulation. Since it is not possible to make comparisons between

[8] `https://help.github.com/articles/github-glossary/`
[9] `https://help.github.com/articles/about-repository-graphs/`

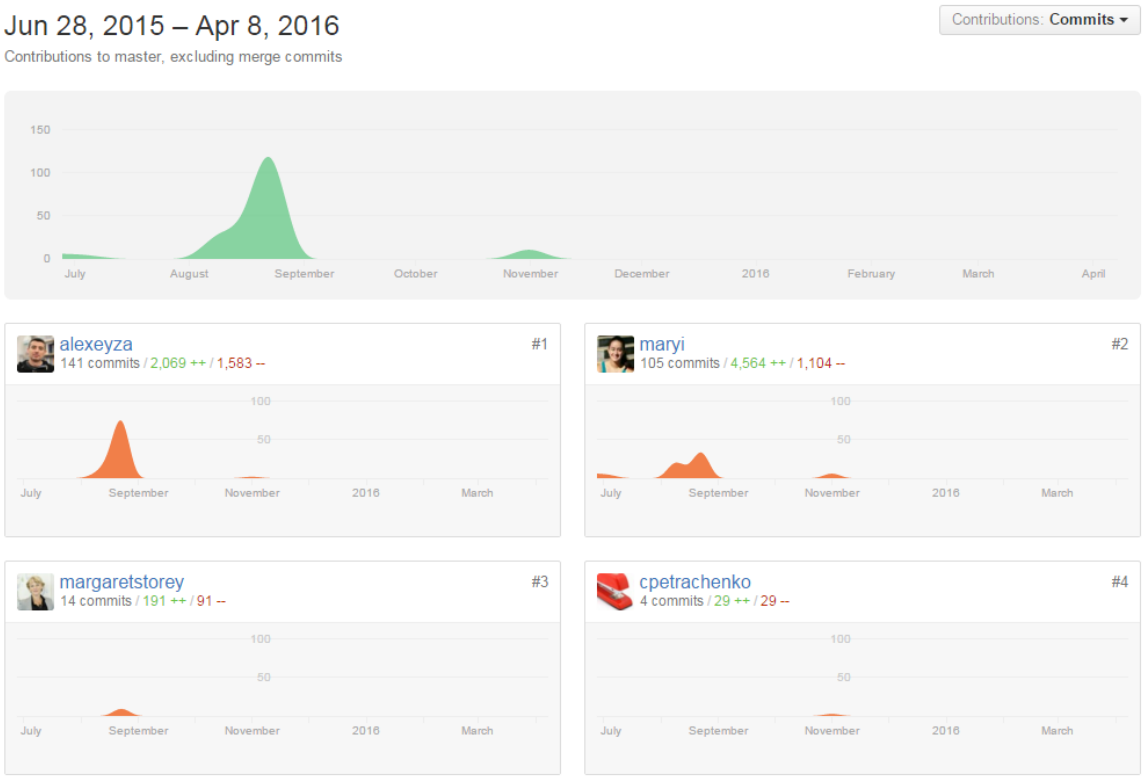

Figure 4.9: GitHub graphs - Example of a contributor's activity view.

individual contributions, this feature is classified as mirroring support. The second view (at the bottom) creates the context for peer-to-peer regulation (co-regulation) and provides awareness support as it shows individual graphs per contributing member and permits comparison across all members.

The *News feed* shows a list of recent activities related to the people or projects the user is working with or following. This feature provides mirroring support and can be used to promote shared regulation because it shows an overview of work being completed towards the shared collaborative vision. Nonetheless, it can also be used to provide co-regulation in the case of individual members watching the activity of close collaborators.

The *Commit "is an individual change to a file (or set of files)"*[10] and is one of

[10] `https://help.github.com/articles/github-glossary/`

the most used features of GitHub. From the web interface in a particular repository, is possible to see the list of commits in a timeline (the most recent at the top) as illustrated in Figure 4.10

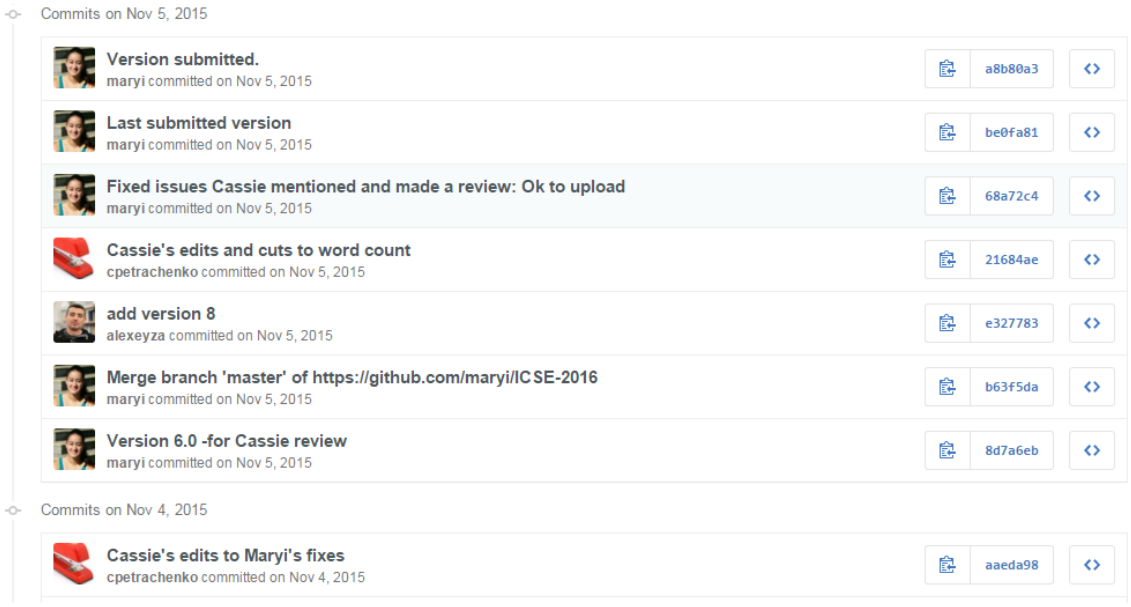

Figure 4.10: GitHub - Example of a timeline of commits

Also, the second button on the right-hand side of the timeline allow access to the modified files where each line has the option to create an inline comment (Fig. 4.11).

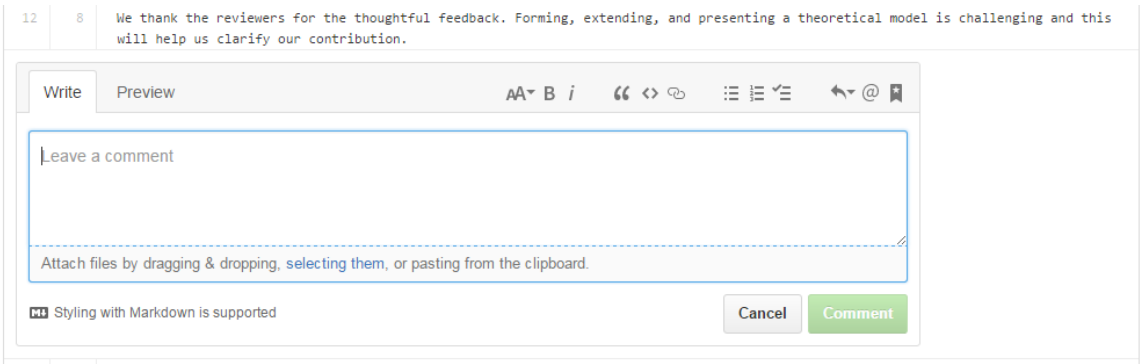

Figure 4.11: GitHub - Example of an inline comment

The view offered to explore the commits allows participants to get a sense of progress and direction over time while recognizing the contributions by others. The

ability to explore the files and leave comments allows users to experience closer collaboration despite the virtual environment. Further, the transparent view of contributions helps to keep contextual awareness and increase participant motivation. Although a commit could allow indirect comparisons of individual work, this possibility is not explicit in the tool, so I classify the commit feature as mirroring support. With respect to regulation support, the visual representation of individual contributions helps with self-reflecting activities thus promoting self-regulatory processes. Also, the timeline provides an overview of collective work which has the potential to support group regulation (shared regulation).

The *Blame* feature allows users to track author's changes on each file line in a commit. In any commit file, click on 'View' and then on 'Blame'. Both buttons can be found in the upper-right corner. The Blame button opens a view that shows each file line connected to a list of contributors (that at some point modified the content) along with a link to their respective commits. If a commit is selected, related files are opened with the option to register inline comments (Fig. 4.12).

Blame provides the history of collaboration at a very detailed level, thus I classify it as an awareness tool that can easily allow comparisons between individual contributions and help detect possible misalignments in people's perceptions or ideas about the task. In a collaborative context, Blame can support self-, co- and shared regulation. The history of file modifications can be used for self-control of contributions and progress. For instance, as collaborative work progresses, I can easily forget what part of the project or task was done by me. Likewise, the transparency provided by Blame allows users to identify opportunities to align ideas with others and prompts co-regulation regulation. The inline comments provide a channel to co-regulate others. Similarly, shared regulation can benefit from the public Blame history and the inline comments represent the current state of the group and can invite collective

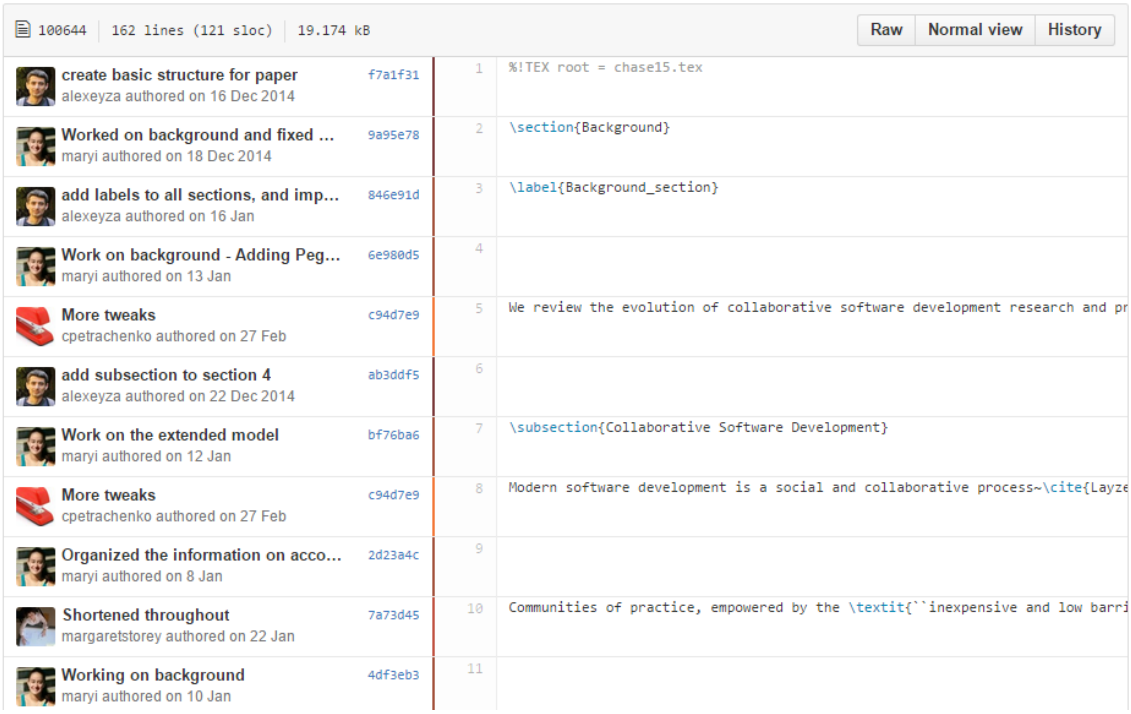

Figure 4.12: GitHub - Example of the Blame view

actions.

Finally, I recognize the name of the feature itself might not suggest a tool for constructive discussion and may imply a negative connotation that could affect its use among members. However, more research is required to discover the different uses of the feature.

4.4 Discussion and conclusions

The theory of regulated learning was proposed over 25 years ago and has been evolving ever since thanks to the continuous efforts of numerous researchers in the learning science domain. In particular, Hadwin's contributions to the theory focus on regulated learning in collaborative settings and they extend beyond to what is presented in this chapter (e.g., COPES system [113]).

The theory of regulated learning developed by Hadwin and colleagues presents a very detailed model that aims to describe how regulation unfolds in learning activities. However, in order to transfer and apply the model to other domains, one requires a deep understanding of the regulated learning theory and the domain it is intended for.

Certainly, I can see how regulation, and informal learning, is experienced in every knowledge domain, but the model that explains how regulation happens has to adapt its constructs and conceptualizations to integrate the particularities of the context it is intended for. As far as I know, there are no other studies that attempt to use Hadwin's theory of regulated learning in software engineering, and as this research being the first attempt, I decided to use only two constructs from her theory (modes and processes of regulation) to create my collaboration model: the Model of Regulation. My reasons for selecting only these two constructs was to keep the model simple enough to be understandable in a context unfamiliar to the concept of regulation, but still be able to bring extra value to the understanding of collaboration in the software engineering domain.

I have no doubt the complete model drawn by Hadwin and colleagues can bring even more insights about collaboration from the regulation perspective, however, research will be required to keep borrowing constructs from learning science and investigate how they fit in software engineering contexts.

The Model of Regulation I introduce for software engineering introduces the concept of regulation in the domain, describes the situations in which regulatory episodes occur (modes of regulation), and illustrates the components and the sequence of these episodes (processes of regulation). This descriptive model shows how productive collaborations take place and can be used as a prescriptive model to guide future collaborations. What's more, the Model of Regulation provides specific vocabulary that

allows analysis and discussions about collaborative experiences at a fine-grained level.

In addition, I also make part of the Model of Regulation a classification system (see section 4.1.1). Although software engineering has a long collection of taxonomies and classification systems for tools, as presented in the background on section 3.2, none of these use feature or tool support for collaboration processes as the base criteria for categorization. Furthermore, the integration of the tool support classification (e.g., Structuring support, Mirroring support) into the Model of Regulation adds an actionable component to the model. The processes and modes of regulation describe human actions that are not easily controlled, however, the technology and the mechanisms we use to facilitate collaboration are always within our control.

Indeed, an understanding of the processes of collaboration and tool support from the same perspective of regulation can provide the theoretical foundation that has been lacking in the development of tools and has the potential to boost individual and team productivity. However, when discussions over a broader context of collaboration in software engineering are included, such as virtual teams and global software development, constraints like time zones, culture, and language must be taken into account in order to continue the refinement of the framework. I expect the model to evolve as more studies are conducted and learn more about the nuances of regulation and the tools that support it in software engineering.

In this chapter, I also showed how the Model of Regulation can be used to for describing and classifying features within a software development tool, GitHub. Note that in any similar analysis, it is important to consider the design purpose of the tool and also how it is really employed by users.

Results from using the Model of Regulation to analyze a tool should not be interpreted as positive or negative based on their support for regulation (or lack of). Rather it should be taken as a characterization of the technology or mechanism to

achieve a deep understanding of the benefits a particular tool can bring to collaboration. A single tool is not expected to provide complete support for regulation nor provide all services required to execute a particular task, but a deep understanding of the tool properties will allow us to make a wiser and more accurate selection of complementary tools to fit our needs that together brings support for all aspects of regulation. I believe the collaborative nature of software development can only reach its full potential if we can clearly identify which components of group interaction are supported by the myriad of tools and resources we have available today.

Following that, it should be noticed that this preliminary analysis was based only on one tool. More research is required to complete the analysis with other socially enabled code hosting tools (e.g., BitBucket) and other types of technologies like management and collaboration task boards (e.g., Trello, ZenHub), software development-focused communication tools (e.g., Slack, HipChat), and personal and team metric tools (e.g., WakaTime). However, from this first exercise using the Model of Regulation, I noticed the model to be particularly helpful in the establishment of a guideline for the evaluation and classification of collaborative tools, which can later bring support for the development of technology intended to be used in a collaborative context. Maybe reducing the number of tools required for a development project while keeping complete support for all aspects of regulation can be the ultimate goal of future collaborative software development.

Chapter 5

Analysis of a software development community using the regulation lens

In this chapter, I illustrate how the Model of Regulation can be used as a lens to examine a software development community. In particular, I introduce the importance of the role of users in software development. Then, I use the Model of Regulation to describe the way regulation is experienced in the open source development community created by the Neo4J project and analyze how users contribute to the regulation of the community.

5.1 User participation in software development

User participation in software development is a widely researched topic [33, 103], particularly in three research areas [1]: *Human Aspects of Software Engineering*, *Requirements Engineering* [26], and *Information Systems* [19]. Some studies have examined the influence of user feedback on project success [104], while others have focused on studying the division of labor between users and developers [19] with the aim of measuring the contribution of their activities [56]. For modern software projects, user involvement is not only beneficial, but considered crucial for survival and success.

However, users need to be empowered to make effective contributions [33], whether through the use of tools, work processes, incentives, or other factors.

Gallivan and Keil [33] proposed a process model for user-developer communication, demonstrated by an analysis of a software project that failed despite high levels of user involvement. The model defines four necessary stages of communication between users and software developers: (1) users become conscious of the messages they need to communicate to developers (e.g., requirements, suggestions, complaints, and other feedback); (2) users transmit messages to developers through one or more communication channels; (3) developers receive and interpret messages from users; and (4) developers set priorities and take action based on their interpretation of the messages.

Users can be categorized into different roles and types depending on their involvement [17, 22]. Furthermore, Damodaran [22] proposes three forms of user involvement: *informative*, where users provide or receive information; *consultative*, where users comment on a predefined service or range of facilities; and *participative*, where users influence decisions related to the whole system. However, as far as I know no research has examined the role of users in a software project using the perspective of regulation. In the following, I present a case study that describes the way regulation is experienced in the Neo4J community and characterizes their collaboration and participants.

5.2 Methodology

The research methodology consisted of a two-phase qualitative case study [86, 116]. In phase I, the purpose of the work was *exploratory* as it was the first attempt to study regulation in a software development community. This phase provided a basic

understanding of regulation in this context. In phase II, the goal was to *describe* (a) the way regulation is experienced in the community, and (b) the role of users in the community and in the evolution of the software project.

Both phases of the study investigated the Neo4J open source community[1]. Neo4J is a graph database management system launched in 2007. Currently, Neo4J is an open source project with a GPL3-licensed open-source 'community edition' and extensions under a commercial license. The open source development community associated with the project includes hundreds of thousands of developers and the project has over 1M downloads to date. The Neo4J community was selected because it is a well-established project with a visible and active user community. However, it is important to note that despite using the same *case* for both phases, they had different purposes (*exploratory* vs. *descriptive*) and relied on different processes and data. This research study was conducted with the approval of the Human Research Ethics Board (HREB) of the University of Victoria. Next, details of the data collection and data analysis processes are presented for each phase.

5.2.1 Phase I: Exploratory study

The purpose of this phase was to explore using the Model of Regulation as a way to understand collaboration in a software development community.

Data collection

The Neo4J community provides open access to their communication channels: GitHub, Stack Overflow, and Google Groups. The official project Website lists these channels and indicates their general purpose: Stack Overflow should be used for asking technical questions about Neo4J; Google Groups is mainly for sharing of ideas; and GitHub

[1] `http://neo4j.com/open-source-project`

should be used for reporting issues and suggesting features[2].

For this exploratory phase, I manually selected the 10 most recent issues on GitHub, Q&A threads on Stack Overflow, and discussions on Google Groups, for a total of 30 conversations from the Neo4J community. GitHub issues were manually downloaded as PDF's files, Stack Overflow Q&A threads were collected using Stack-Exchange [3] (for specific queries, see Appendix A), and Google Groups discussions were manually copied and pasted into text files. These conversations included an average of 4 messages each (120 messages in total). Original data is freely accessible through the communication channel Website[4].

Data analysis

Due to the exploratory nature of the work, the analysis of the data started with a flexible conceptualization guided by the original constructs of regulation (modes and processes) as defined in the learning science domain integrated with my personal notion of regulation in software engineering. The dataset was examined for traces of regulation and then the conversations were coded based on the current definitions of the Model of Regulation.

Despite starting with a theoretical model to guide the coding process, I adopted a largely inductive and emergent approach loosely informed by the iterative coding cycles adopted in grounded theory analysis [13, 91]. Thus, each trace of regulation identified in the dataset allowed me to refine the conceptualizations in the Model of Regulation, improving the understanding of regulation in software development. For every refinement in the model, a new coding iteration over the dataset was conducted in order to ensure consistency in the coding criteria.

[2] `https://neo4j.com/community/`

[3] http://data.stackexchange.com/stackoverflow/query/new

[4] GitHub: `https://github.com/neo4j/neo4j`, Stack Overflow: `http://stackoverflow.com/questions/tagged/neo4j`, Google Groups: `https://groups.google.com/forum/!forum/neo4j`

Analysis of the data in this phase showed that the way regulation is experienced and promoted by participants in the community strongly depends on their role in the collaboration. The particular goals and expectations of a participant with respect to the collaborative activity lead and define their regulatory actions in the group. In particular, messages generated within the Neo4J community were identified as coming from two different actors or perspectives: *Contributors* and *Users*. Contributors are those directly helping with the creation of the product, which includes participation in the development process and documentation. Users are those using the product resulting from the contributors' efforts. Although it is possible for a single participant to adopt both perspectives —Contributor, User— this analysis assumes those are exclusive to simplify the exposition. A copy of the original data and the coded conversations are available online[5].

5.2.2 Phase II: Applying the model

The purpose of this phase was to apply the Model of Regulation to describe the way regulation is experienced in an open source development community and to describe the role of users in the regulation of the software project.

Since this study relies on qualitative data that is broader and richer but less precise than quantitative data, there was a need for triangulation in phase II. *Data triangulation* [96] was achieved by using different communication channels as data sources.

Data collection

For phase II, conversations from GitHub, Stack Overflow, and Google Groups were collected for a period of four months beginning on March 25th, 2015. This specific

[5] `https://github.com/maryi/Neo4J-CaseStudy`

time frame was chosen as it followed the release date of the most recent significant version of the Neo4J software product[6] (at the time of conducting this study). Only the conversations started during that time frame were considered—messages created on or after that time frame, but from older conversations, were not included in the sample.

GitHub issues were collected using the GHTorrent project[39]. Stack Overflow data was collected through the Stack Exchange API[7] by searching for all posts tagged with `Neo4J`. The data collected included questions, answers, and associated comments. Lastly, Google Groups conversations were manually collected, and for each conversation, a PDF file was created. Original data is freely accessible through the communication channel Website[8]. For technical details (e.g., codes, queries) of the data collection, see Appendix A Section A.2.

Data analysis

For the coding process [13] in phase II , two coding schemes were defined: regulation modes and regulation processes. For both schemes, a second level of coding was added to identify the actor or perspective from which the messages were created (as guided by the findings of phase I). The criteria defined to identify contributors and users is presented in table 5.1.

The resulting coding scheme allowed me to estimate each actor's impact and contribution to the collaboration dynamic. Table 5.2 shows the complete coding scheme.

The coding process was performed by three researchers, one per communication channel. The researchers were instructed on the Model of Regulation and then pro-

[6] `http://neo4j.com/release-notes`

[7] `https://api.stackexchange.com`

[8] GitHub: `https://github.com/neo4j/neo4j`, Stack Overflow: `http://stackoverflow.com/questions/tagged/neo4j`, Google Groups: `https://groups.google.com/forum/!forum/neo4j`

Actor type	Criteria
Contributor	- Listed as a contributor in the Neo4J GitHub repository. - Participants that had made a contribution (commit) to the project on or after March 25th 2015. - Self-identified as a Neo4J staff member or developer in their messages. - Their messages were written from the contributor's perspective.
User	- Self-identified as a user in their messages. - Their messages were written from the user's perspective.

Table 5.1: Criteria to identify contributors and users

ceeded to individually code the data, analyzing the conversations sequentially, one at a time. When regulation traces were found in the dialog, the researchers proceeded to code the data. First, they coded the regulation processes (e.g., Goal Setting) and the actor the message was created by. Then, they coded the modes of regulation based on the regulation purpose and the actors involved. Consequently, each regulation trace had a minimum of four codes. As an example, Figure 5.1 shows a coded GitHub conversation between a user and a contributor. The stripes on the right side indicate the scope of each code assigned to the conversation. The top message involves an evaluation of the software product from a user's perspective, thus it was coded with *Monitoring & Evaluating* and *Users* codes. When coding for regulation modes, since the message represents an individual reflection, it was coded as *Self-regulation* and *User*. Also the events in each communication channel were coded (e.g., when a conversation is tagged, an issue is assigned to someone) as they also represent participants' intentions and activities. In the previous example (Fig. 5.1), a *tagging* event occurred and was coded as *Enacting* and *Contributors*.

For coding, an iterative and collaborative process was followed. Researchers strategically exhausted the data in terms of confirmations, evidence of regulation, and

Regulation processes	**Actor**
Task Understanding	- Contributor - User - Contributor&User
Goal Setting	- Contributor - User - Contributor&User
Enacting	- Contributor - User - Contributor&User
Adapting	- Contributor - User - Contributor&User
Monitoring and Evaluating	- Contributor - User - Contributor&User
Regulation modes	**Actor**
Self-regulation	- Contributor - User
Co-regulation	- ContributorTOContributor - UserTOUser - ContributorTOUser - UserTOContributor
Shared regulation	- Contributor - User - Contributor&User

Table 5.2: Coding scheme

contradictions to the conceptualizations in the working Model of Regulation. Each contradiction was a call for group discussion, which was used to revisit, adapt, and change the working model. Every time the model was adapted or refined, researchers went back to the first conversation of their dataset and started a new iteration of the coding. This was done in order to ensure consistency in the coding criteria throughout the data. In addition, this process allowed the researchers to become deeply familiar with their particular data stream and allowed the study to make use of distributed expertise. Conversations were coded until a saturation point was reached. That is,

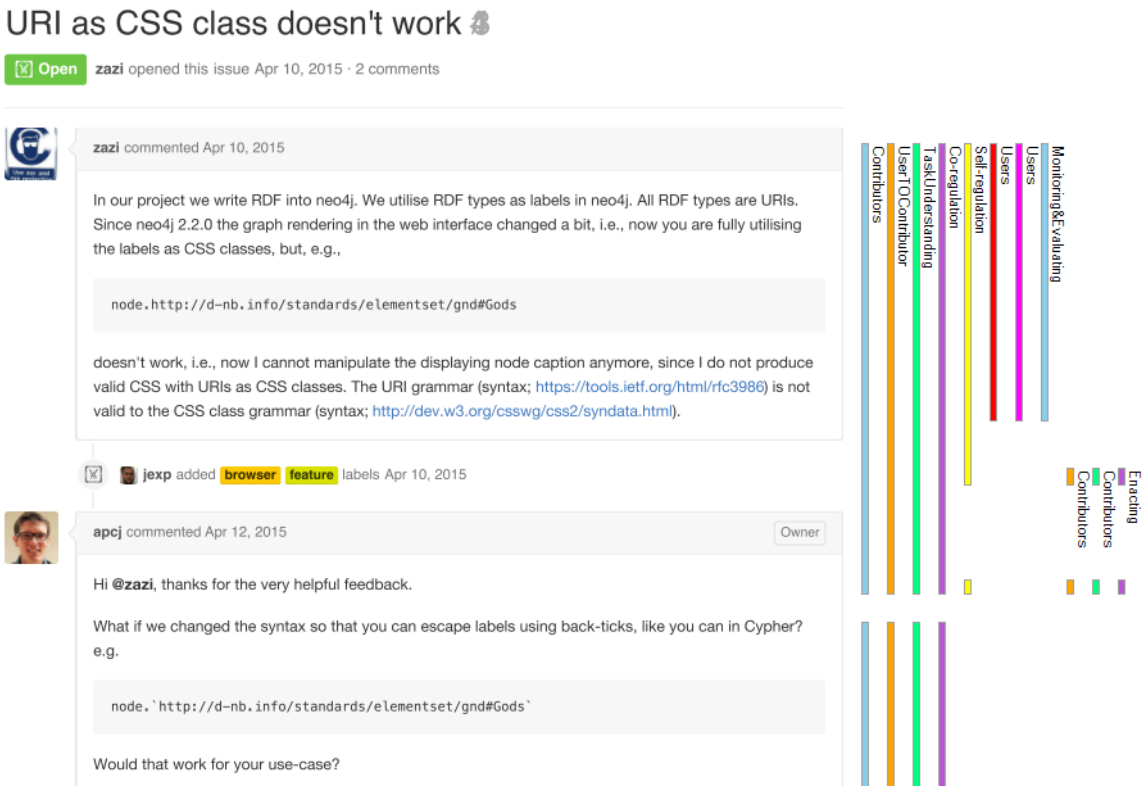

Figure 5.1: The coding process [GH Issue #4413]

when the themes of the regulatory experiences in the project were exhausted and there was nothing to add or refine in the Model of Regulation. The saturation point was reached after 242 GitHub issues, 216 Stack Overflow threads, and 103 discussions on Google Groups. PDF copies of the original data and the coded conversations are openly available online[9]. Tables 5.3 and 5.4 present a summary of the regulation traces per actor found in the analysis of the data from the Neo4J community.

5.3 Threats to validity

Here, I discuss threats to the validity of the approach [86] and present the measures taken to mitigate them.

[9] `https://github.com/maryi/Neo4J-CaseStudy`

Modes of regulation per actor	Number of traces
Self-regulation	*613*
Contributors	412
Users	193
Co-regulation	*348*
ContributorTOContributor	137
ContributorTOUser	111
UserTOContributor	40
UserTOUser	87
Shared regulation	*25*
Contributors	20
Users	1
Users&Contributors	4

Table 5.3: Modes of regulation per actor: summary of the regulation traces found for the Neo4J community.

5.3.1 Construct validity

Construct validity relates to the relationship between the theory and the data analysis. To reach the emerging themes in this case study, it was necessary to rely on subjective human judgment during the data coding phase; researchers have to decide if a message falls within a specific coding category. To alleviate this issue, the researchers were instructed on the Model of Regulation prior to the data coding phase and had access to an expert in ambiguous situations. Furthermore, the findings are traceable as the data sources used in the case study are publicly available. Also, the coded data is openly available online[10].

Another concern on construct validity is the use of qualitative data which is broader and richer but less precise than quantitative data. This issue is mitigated by using data triangulation, i.e., three different sources of data.

[10] `https://github.com/maryi/Neo4J-CaseStudy`

Processes of regulation per actor	**Number of traces**
TaskUnderstanding	*406*
Contributors	53
Users	225
Users&Contributors	156
GoalSetting	*85*
Contributors	21
Users	25
Users&Contributors	39
Enacting	*373*
Contributors	339
Users	30
Users&Contributors	4
Monitoring&Evaluating	*163*
Contributors	103
Users	49
Users&Contributors	17
Adapting	*2*
Contributors	0
Users	2
Users&Contributors	0

Table 5.4: Processes of regulation per actor: summary of the regulation traces found for the Neo4J community.

5.3.2 Internal validity

Internal validity refers to factors that could influence the results of the case study presented here. One possible factor can be the data size and whether it is representative of all possible cases. This issue is mitigated through the study design by planning to collect a large data set—the communication data for a period of four months. The study design was further reassured by reaching a saturation point prior to analyzing the full data set.

Another possible factor can be the saturation points during the data coding phase of the analysis. The saturation point for each channel was subjectively recognized by the researcher responsible for coding that channel. It is possible that analyzing

additional data, or using a different dataset, may reveal new cases. This concern is minimized by the case study design and by choosing to use an openly accessible data source.

5.3.3 External validity

External validity refers to the generalization of the case study findings. Choosing the Neo4J project and its community as the case study may have introduced a bias towards this population, thus the findings may only apply to open source projects similar to Neo4J. However, I believe that the same or similar communication channels (GitHub, Stack Overflow, and Google Groups) are used in other open source and commercial projects, indicating that the findings of the study may apply in these settings as well.

5.4 Findings from phase I: Identifying participants in the software development community

The main finding from phase I was the identification of the large influence participant's roles have in the regulatory episodes of the community. Regulation refers to strategic actions in the face of changes, and when examining a particular unit of collaboration, one may assume the way regulation is experienced is the same for all participants. However, this study proved that assumption to be wrong — despite collaborating, each participant may have different motivations for executing the strategic actions that define their regulation. Participants still regulate themselves (self-regulation), influence others (co-regulation), and regulate the group as a unit (shared regulation), but the particular goals when executing these actions are different, and their traces of regulation reflect only those components of the project they

are interested in. Nonetheless, the overarching goal of participants is the same and that is the reason a community like Neo4J holds together.

Through the analysis of the messages created by the Neo4J community, it was possible to identify traces of regulation from each of the processes described in the Model of Regulation. For example, messages associated with the Task Understanding process were found, however, their content was different in nature. Some messages would discuss technical implementation of a feature within the Neo4J project, while others would discuss how to integrate the project within a particular context. These differences were noticed for all processes of regulation found in the data and it was concluded they were due to participant's goals or motivations for participating in the collaboration. That is, each different actor has their own particular expectations from participating in the community, which makes them regulate only specific aspects of the project. While a developer's goal may be to gain experience in programming languages, users might collaborate because Neo4J is a product they would like to implement in their own projects. Because of these differences, the traces of regulation processes looked different for each actor. In particular, two main actors were identified in the regulation of the Neo4J community: *Users* and *Contributors*.

The analysis of the messages in the community revealed the following motivations for collaboration.

- *Contributors*: (1) Develop a software product; and (2) maintain high visibility on project resources so that any interested user can use the product.
- *Users*: (1) Use the software product; and (2) integrate the software product into their own particular project.

This differentiation of actors led to a revision of the understanding of regulation in the community, adding their perspectives for each process and mode of regulation.

Figure 5.2 represents a discussion between the different actors in a GitHub issue showing the messages directed between them and the regulation experienced in a vertical timeline per participant.

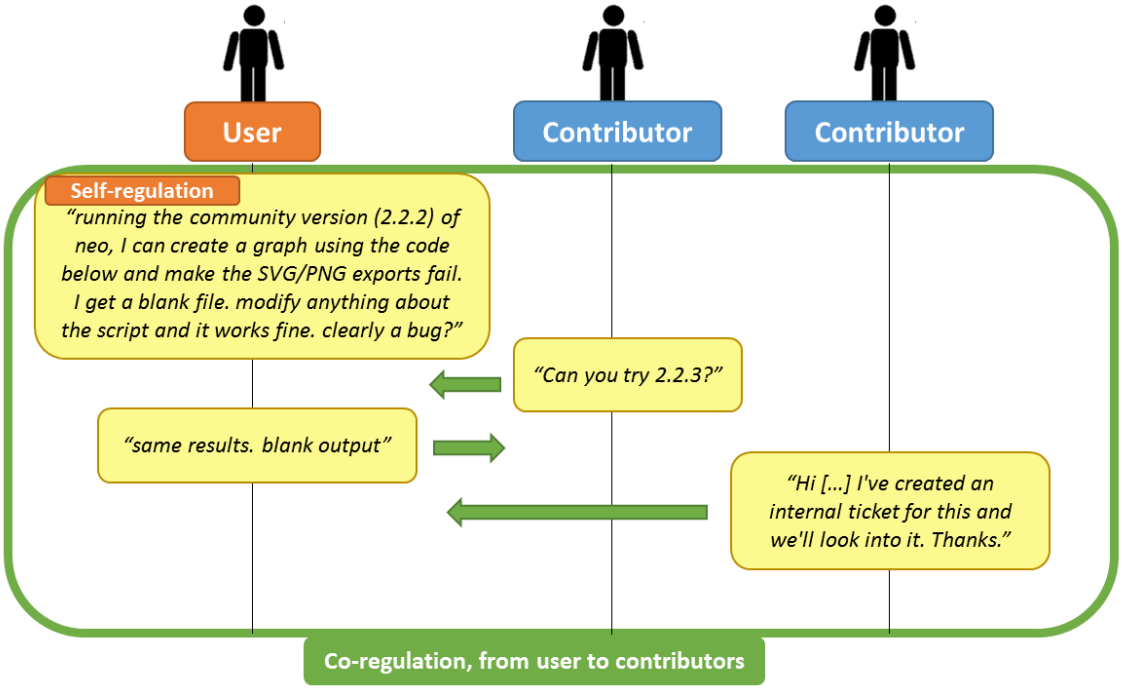

Figure 5.2: Example of regulation in the community of Neo4J — GitHub issue id 4893.

In addition, a detailed inventory of regulation processes and regulation modes from each actor perspective was created to describe the regulation possibilities. This inventory was drawn from the case study example and is illustrated in Table 5.5 for the regulation processes, and in Table 5.6 for the regulation modes. In these tables, the first column presents the definition of the regulation process or mode, and the next columns present comments from each actor as a guideline for coding the regulation of users and contributors.

Processes of Regulation	**Guidelines for Contributors**	**Guidelines for Users**
Task Understanding: Unifying perceptions about the project/task at hand. Project comprehension or understandings include: requirements, purpose, scope, social context, and roles and responsibilities of participants with respect to the task.	- What is the purpose of the Neo4J product? - What are the requirements to implement functionalities? - What is the scope of the project?	- What is the project about and how can it help me? - What is required to deploy Neo4J the project? - What programming languages are required to integrate Neo4J with my personal project?
Goal Setting: Establishment of a work plan, which includes goals, task standards (e.g., deadlines, product quality), resource allocation, strategies, methods and tools to support collaboration and task performance.	- What features should we include on the next release? - New features should be ready a week before the release due date to allow proper testing. - We will use GitHub to manage technical information.	- New documentation was generated to understand the integration of Neo4J with our personal project. - What steps are required to implement a particular feature of Neo4J? - What communication channel should we use to get help with this issue?
Enacting: Following the plan while providing support for motivational engagement.	- We are using Stack Overflow to answer technical questions for the users, as we defined. - Documentation, coding and testing of the product is done following the guidelines defined.	-We read the documentation and the FAQ before implementing the Neo4J project. -The documentation presents this configuration as a simple process, however, we have problems getting the expected results.

Processes of Regulation	**Guidelines for Contributors**	**Guidelines for Users**
Monitoring and Evaluating: Tracking and assessment of project comprehension and the work plan.	- Communication channels are not being used as intended, people are using Google groups to ask technical questions when they should be using Stack Overflow. - Are the new features properly tested and documented?	- The information provided by the developers is not consistent with the output we get from Neo4J. - This feature does not work as you describe it should for my case.
Adapting: Refining the work plan based on partial outcomes or changes in the project understanding.	- We need to re-think this feature, the implementation is getting more complex than expected. - If we want to provide support for all operating systems, we are going to need to change this feature.	- We discovered it is not possible to implement Neo4J they way we wanted, so we are exploring this other way of doing things. - We want to implement Neo4J in this other way, is it possible to have this feature in the next release?

Table 5.5: Inventory of possible traces of regulation processes for the Neo4J community

Modes of Regulation	**Guidelines for Contributors and Users**
Self-regulation: Individuals engage in strategic actions to optimize the results of their own work. Regulation can be experienced by a contributor or user.	- *Self-regulation of a contributor*: I found this bug, I will update the documentation accordingly. - *Self-regulation of a user*: I have been working with the tool and I've found there is an easier way to create this graph.
Co-regulation: Individuals and the community support or influence one or multiple member's regulation processes. Regulation can be experienced (a) between contributors, (b) between users, and (c) between users and contributors.	*Case (a) co-regulation between contributors:* You need to make sure the new features are supported by all versions of the Neo4J product. *Case (b) co-regulation between users*: In my company, we do weekly reviews of the error log in the Neo4J project to ensure stability of the implementation, maybe that could help you, too. *Case (c) co-regulation between Contributors and Users*: -*Contributor to user*: Remember to use the last version of the Neo4J project for better results. -*User to contributor*: There is a bug in this feature that requires your attention.
Shared regulation: The community collectively negotiates and defines or realigns group regulation processes. Regulation is experienced by (a) the group of contributors, (b) the group of users, and (c) the community formed by users and contributors.	*Case (a) shared regulation between contributors*: - *Contributor*: We should create a label for critical bugs on GitHub. - *Group of contributors reply*: Yes, we could use it to tag important issues and keep track of them. *Case (b) shared regulation between users*: - *User*: When someone finds the solution for their implementation of Neo4J, please post it on the channel so other users can benefit from the information, too. - *Group of users reply*: Yes, and when you post your solution, be as specific as possible and list all the tools you used to solve the problem. *Case (c) shared regulation between contributors and users*: - *Group of contributors*: We will post the list of possible new features to be added in the next release and you (users) please vote for those you think are critical. - *Group of users*: Yes, you could use the votes to set-up implementation priorities.

Table 5.6: Inventory of possible traces of regulation modes for the Neo4j community

The initial inventory guided the coding process in phase II and was gradually refined and completed with the findings from the data.

5.5 Findings from phase II: Collaboration of the community from the perspective of regulation

Discovering the influence the participant's role had on the regulation traces made it clear there was a need to include their perspectives when describing regulation in the community. Later, with the analysis of more data in this phase, I developed a more accurate description of the regulation experienced in the community.

Regulation in the Neo4J open source development community is mainly supported by the contributors who are actively involved with the software project, either developing code or creating documentation and resources, while regulation is supported by the users from the outside. This dynamic is illustrated in Figure 5.3 with each actor using its own regulation processes (users coded in orange and contributors in blue), which may or may not align with each other.

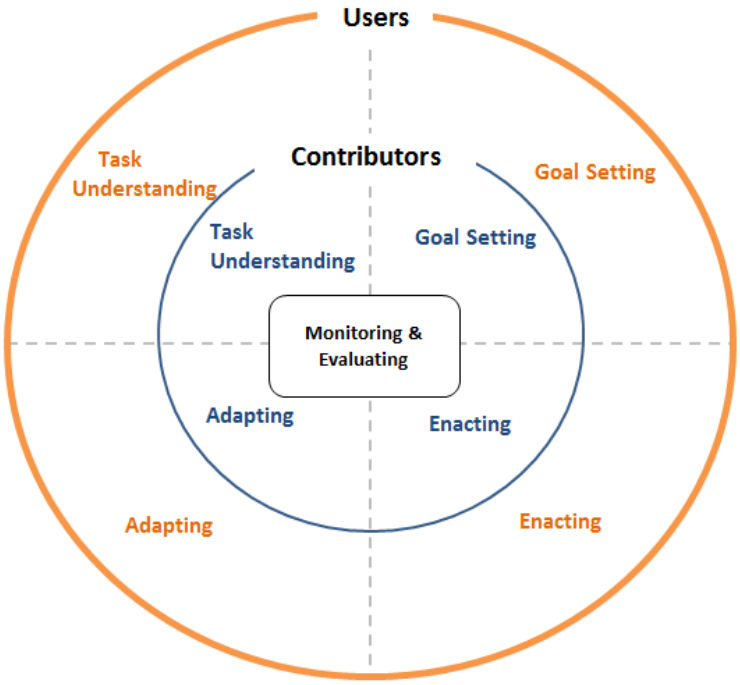

Figure 5.3: A representation of the regulation processes in the Neo4J community

Similarly, the representation of the regulation modes (self-, co-, and shared regulation) was also extended to include the actor's perspective. Interestingly, this illustrated how social modes of regulation (co- and shared regulation) can be experienced between different actors, creating an *inter-perspective* regulation. For example, co-regulation from users to contributors or shared regulation between users and contributors. A representation of the regulation modes for the Neo4J community is illustrated in Figure 5.4. As before, it should be noted that the regulation mode from one actor may or may not have linear correspondence with the regulation mode of the other actor. This illustration should be considered as a *snapshot* of the possible regulation modes in the Neo4J community at a particular moment in time.

In addition to users and contributors, Figure 5.4 depicts additional actors or perspectives that may participate in the regulation of the project from the outside. However, in the dataset examined, their participation was not substantial enough to be considered as main actors (e.g., they were an indirect influence from the user's environment or from other projects).

The inventory of regulation possibilities for Neo4J (Table 5.5 and 5.6) was created in phase I considering all possibilities of regulation participants from the community may have experienced, and had the purpose to guide the coding process in phase II. The expectation of this phase was to find traces of regulation for each of the cases considered in the inventory. However, this was not possible because the data mainly contained conversations between contributors and users. So, it was decided to focus the analysis in the inter-perspective co-regulation between users and contributors.

In accordance with the Model of Regulation's terminology, I report the emerging themes of the *inter-perspective co-regulation between users and contributors* in the Neo4J community. Here, only themes that were found multiple times in the three streams of data are presented. In the following, bolded text is used to indicate main

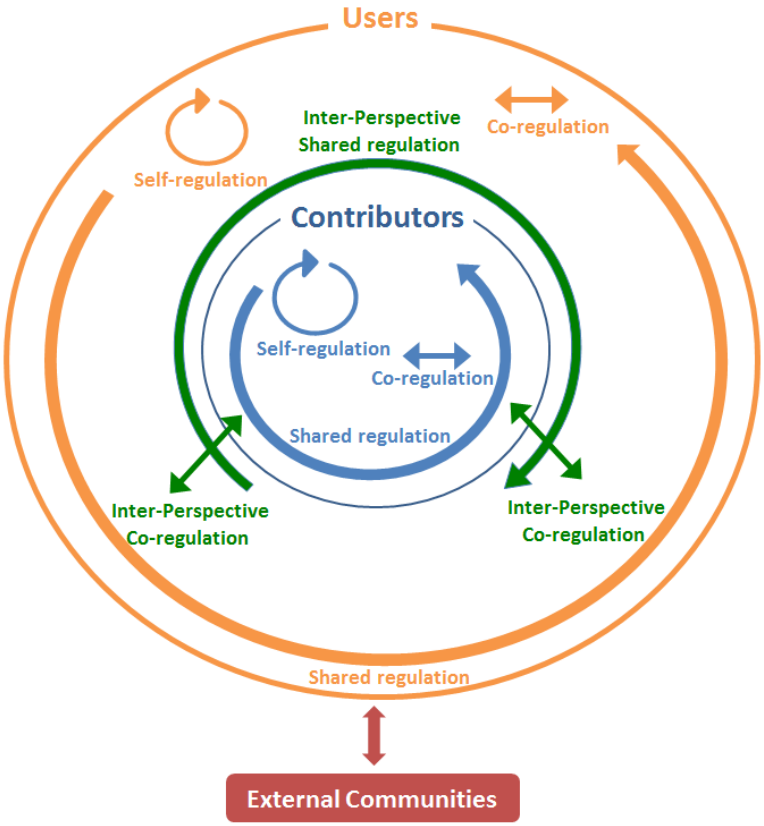

Figure 5.4: A representation of the regulation modes in the Neo4J community

themes that emerged from the analysis, and representative quotes are shown in italics and linked to each communication channel with an identifier. SO# is used to indicate the question ID as assigned by Stack Overflow. GH# is used to indicate the issue number inside the Neo4J GitHub project. And for Google Groups, GG is used with the creation date of the message. Note that some messages were coded with multiple codes.

User participation promotes task understanding processes among contributors. For instance, a contributor replied to a user's inquiry that required deep knowledge of the project's scope and requirements: *"This [task] can take a lot of time. The good news is that Neo4J 2.3 will fix this: [link]."* [SO 29371681] In a different case, a contributor's answer showed the user's demand for awareness of the

project's progress: *"I've confirmed that we have resolved this [bug] in Neo 2.2.0."* [GH 4352] This level of user participation demands an active and knowledgeable team of contributors.

Users also participate in project task understanding by reporting their experiences with new versions of the product: *"After updating to the latest version, the above query does not seem to care about label anymore."* [GH 4316] Such input helps contributors be aware of new issues or project limitations. In some cases, user-contributor discussions identify that the real issue is with the documentation or other written communication. In these cases, the dialog between them allows contributors to define and improve the common vocabulary in order to better understand each other: *"So perhaps the error message should be different, but it is an invalid query in the first place."* [GH 4342]. Achieving agreements like this facilitates and promotes future collaboration.

Finally, users contribute to the evolution of project task understanding in two ways. First, users describe interesting scenarios that call the contributor's attention: *"Interesting—what's your use-case for this?"* [GG, April 7] The following shared ideas, promoted by this kind of message, allow contributors to discover new ways in which their product is being used and allow them to recognize a broader scope for the project. Second, users contribute by suggesting appropriate features that end up being accepted for future inclusion in the project. It was noted that the Neo4J community has an implied understanding that accepted features do not require more explanation than merely tagging an issue with the 'feature' label (e.g., GH 4470). When user perception is aligned with the project vision, users are able to create requirements that are within the project scope and can easily be integrated as part of the project plan.

User participation contributes to the definition of project goals and

standards. For example, a contributor's response to a user reinforced a goal for the contributor's team: *"...the bug fix for this shall be part of the 2.2.2 milestone release, in about a month's time."* [GH 4522] In more interesting cases, contributors try to align their goals with those of the user: *"When do you plan your first release? We hope to get... within one to two months. If that's ok with you, I'd start with that."* [SO 29434348] These findings show different ways that user participation contributes to project planning.

In addition, user-contributor interactions create opportunities to co-define standards for both actors. For example, a contributor replied to a user: *"...please add a label to your nodes, not sure what the type is."* [GG, April 18] While from the user's side, the user articulated the kind of answer they expected: *"You should give more information... Please explain HOW to..."* [SO 29434079] Also, contributors direct users to appropriate resources to help them attain their goals with messages like: *"You can check this page for an example..."* [SO 29500388]

Close collaboration between users and contributors helps them reach agreement on the strategic selection of tools and resources to achieve their goals. For example, a user approached a project member using email and the contributor pointed him to a more appropriate channel: *"I'm forwarding your message on to our Google groups who should be able to address it better that I can here."* [GG, April 23]. Contributors teach users to select more appropriate resources in the future by suggesting which tools to use.

In other cases, users were confused about the results they were getting. After a discussion with a contributor, it turned out that the strategies the users were following were not appropriate for their goals. This contributor guidance promotes user engagement with the software product, showing users that their problems originated from their approach to the tool and not the tool itself. I also found cases of contribu-

tors being thankful for user contributions and encouraging their future participation in the project, as expressed here: *"I wanted to reach out to you [community] to ask for feedback. If you successfully integrated Neo4j... we'd love to read about it."* [GG, May 4] This message was a call for all community members to share details on their use cases, as this contributor described: *"Document and link some of those approaches... so others have an easier time getting started."*

Users play an important role in evaluating the project and supporting collaboration. Contributors consider and include user reports on bugs and issues when developing their project plans: *"Great catch, I'll pass that [bug report] to our team."* [GH 4333] In this way, users' technical skills serve to evaluate the product and prompt contributors to take action when results are not aligned with goals or standards. More importantly, discussions with users created opportunities for contributors to reflect and re-evaluate their own decisions: *"I think 2.1 did the wrong thing here."* [GG, April 10] This shows that user-contributor dialog opens the space for self-reflecting activities about the project process that can be beneficial for future contributor decisions.

Users also support collaboration, especially the more experienced users that helped to monitor the correct use of communication channels when messages were off-topic, providing relief to the contributor's team (e.g., SO 29347397).

Finally, users can contribute to the evaluation of the project plan by providing new directions and features for the project. For example, a user described the functionality he wants: *"Allow a user to send a parameter for the label... I would like some additional usefulness."* [GH 4334] In this evaluation and monitoring phase, users can suggest features as part of their evaluation of the project. These suggestions need to be assessed by the contributor's team, and once approved, can become part of the project's regulation process related to task understanding.

The previous quote was taken from a conversation where a feature was requested but not added to the project plan. The issue was left open but not tagged as a 'feature' for immediate implementation due to architectural limitations. This kind of user report provides the Neo4J community with a list of requirements, some of which may have not been considered. Users provide a different point of view and can offer more diverse and applicable use cases that Neo4J can then choose to include in their roadmap. This exchange of ideas allows contributors to recognize the users' perception about what they think the tool should do. More importantly, this input allows Neo4J to know what its users want from the project.

5.6 How this study changed the Model of Regulation?

The constructs that compose the Model of Regulation were borrowed from the learning science domain. Through this study, I confirmed those constructs —processes and modes of regulation— applied to the software engineering context and provided mechanisms to study collaboration from a different perspective not considered before in the domain. However, this study also illustrated the constructs definition needed to be adapted in order to represent the objects of regulation that were relevant for software engineering practice.

The refinements on the constructs definition were done based on the findings from the study of the Neo4J community, as well as from my personal knowledge and that of the research team that helped me to conduct the study. Although this is a small sample of the software engineering domain, I believe it is representative enough to say the Model of Regulation is useful for understanding collaboration in this context. In the following I describe how the processes and modes of regulation were adapted.

For the processes of regulation, the Task Understanding process as defined by the learning science domain describes how *"students scan their environment including tasks set by a teacher, exercises posed by a textbook's author, and knowledge they have about themselves. Using the information they perceive, students construct a personalized profile of the task at hand"* [113]. In contrast, the Task Understanding process for participants of the Neo4J community includes the comprehension of user requirements, purpose of the project, scope and a general understanding of roles and responsibilities of participants with respect too the task.

The Goal Setting process in the learnig science domain describes how students after understanding the task, *"may set one or several goals for which they will strive"*. Further, *"goals can refer to overt behaviors, forms of cognitive engagement, changes in motivation, or all three"* [113]. In software engineering, through the Neo4J study it was illustrated the Goal Setting process involves the creation of a plan which contains the goals —as described by the learning science literature—. However, the plan also includes acceptable standards for conducting the task (e.g., deadlines, product quality standards), description of resource allocation, and strategies and tools to be used in supporting collaboration. These additional objects of regulation relevant to software engineering were added to each process definition.

The Enacting process did not require major adaptations. This process in the learning science domain describes the phase where *"students put their plan into action by activating study tactics and taking other steps to reach goals"* [113] and in software engineering, this phase also includes following the plan set in the Goal Setting process. The only adaptation to the construct definition was the explicit addition of considerations for the motivational support participants require while executing their plans. As cognitive (knowledge) and behavioral components are usually the focus of software engineering plans, the support for motivational engagement according to the

researchers experience is typically not considered.

Also, in the learning science domain the Adapting process describes a phase where *"students deliberate about how to change their methods for completing tasks...to close gaps they perceive between the results of their work and goals"* [113]. Through the Neo4J study, I found participants of the community adapt their practice when the outcomes are not as expected —as described by the learning science literature—, but they also experience an adapting process due to the highly dynamic nature of software engineering projects, where requirements and scope are constantly modified by users and other stakeholders.

Finally, monitoring and evaluating is not recognized as a phase in the learning science domain. It is considered as a facilitator that allows all the other four processes (Task Understanding, Goal Setting, Enacting and Adapting) to unfold by monitoring and evaluating student's progress and providing feedback to the other processes. However, in the case of software engineering, through the Neo4J study it was possible to identify how important and critical the task of Monitoring and Evaluating progress and changes to the social context is for the success of a collaboration and the development of a software asset. Thus, Monitoring and Evaluating in the Model of Regulation is considered as a process.

For the modes of regulation, I discovered their definitions had to be broadened. In the learning science domain regulation is executed by students to students — as it can be seen from the quotes above—, however, in software engineering there is not only one type of participant. Software engineering projects include several stakeholders whose motivations and interests in the project makes them regulate very different components of it. While a junior developer is interested in gaining programming experience regardless of the particular task assigned, a senior developer is more interested in administrative tasks and high level design of the software and a

user is interested in the general idea of the project and how they can use it in their own context.

Through the Neo4J study it was possible to discover the importance of the role of participants when studying regulation and adjust the definitions of self-, co- and shared regulation. Self-regulation still make reference to individual regulation, but in software engineering it could be experienced differently depending on the role of the participant. Co-regulation refers to the regulation promoted by interactions of participants with their social context, and due to the diversity of stakeholders that participate in a collaboration in software engineering, co-regulation could happen between participants with the same role (e.g., co-regulation between users) or it could be experienced between participants with different roles in the project (e.g., co-regulation between users and developers). This consideration was also applied to shared regulation.

In sum, through the Neo4J study the processes of regulation were adapted to include the objects of regulation relevant to software engineering practice and the modes of regulation were adapted to include considerations for the multiple stakeholders that usually participate in software engineering projects.

5.7 Discussion and conclusions

The case study presented in this chapter is the first attempt to explore and describe collaboration in an open source development community using the processes of regulation. This chapter showed how the Model of Regulation provides new mechanisms and conceptualizations to analyze collaboration in software engineering and describe its participants.

It must be noted that the analysis presented here is a showcase of the Model

of Regulation potential for analyzing software development communities. A deeper analysis can be conducted using the model and integrating previous research. For example, in phase I of the case study, it was discovered that the participant's role has a big influence in the regulation of the community. It was also identified two main perspectives or actors: Users and Contributors. Then, the modes and the processes of regulation were extended and adapted to include the actors. However, there is existing research in software engineering that investigates the different activities developers participate in when working in a project [100, 115, 4] and on characterizing users [17, 22]. Results from these studies could be integrated with the Model of Regulation to achieve a more detailed characterization of participants and their regulation in a software development community.

Phase II of the case study described regulation in the Neo4J community and analyzed the role of users in the regulation of the project. This particular focus on users was due to the nature of the data collected, where few conversations between contributors were found. Further investigation of this issue revealed Neo4J uses GitHub, Stack Overflow and Google Groups as public channels to stay in touch with their user community, while other private channels are used for the communication between contributors. The reason behind this separation of channels is because of the project commercial license, which prevents Neo4J's corporate owner from disclosing certain data to the public. It was a shame to discover this limitation in the data. The few traces of regulation found in conversations between contributors confirmed the conceptualizations in the Model of Regulation, but it was not possible to conduct a deep analysis due to the lack of evidence in the dataset.

Finally, case study presented in this chapter was conducted on an open source development community where the the main criteria for selection was the easy access to data for research purposes. Similarly, the Neo4J project was selected for having a

big community that actively uses several different communication channels. However, the methodology followed in this case study can also be applied to the study of software projects of different nature (e.g., commercial software projects), and the Model of Regulation and the extended version of the regulation processes (Fig. 5.3) and modes (Fig. 5.4) can be used to understand and describe regulation in these other software engineering contexts.

Chapter 6

The Model of Regulation in action

In this chapter, I show how the constructs in the Model of Regulation can be turned into an instrument for software engineering practitioners and researchers. I introduce the instrument I developed to assist in investigating collaboration practices of individuals and larger units of collaboration. Then I present the results from applying the instrument to two software projects.

6.1 Research design

In the Model of Regulation, the processes of regulation (e.g., Enacting, Adapting) describe the set of strategic decisions that must be made while doing a task, and the modes of regulation (self-, co-, and shared regulation) indicate the scope within the regulatory cycle. To demonstrate how this theoretical framework can be used by practitioners and researchers, I developed an instrument, a questionnaire based on the Model of Regulation. The goal of this instrument is to guide a contextually sensitive diagnosis of regulatory strengths and potential areas for improvement in terms of working process, communication channels, and tool use. Once the instrument was ready, I tested it by conducting interviews with three developers currently engaged in collaborative software development.

6.1.1 Participants

For this study, I interviewed three software engineers (1 women and 2 men). Interviews were conducted face-to-face, audio recorded for analysis and interview transcripts removed any identifiable data from participants.

First, I interviewed a software engineer with approximately four years of experience in software development (referred to as P1). At the time of the interview, P1 had been working as a co-op developer and UX designer for a period of seven months. All development-related tasks were of a collaborative nature and the work setting was completely virtual as the team was distributed. P1 worked with a team of around twenty people on the development of low level components of a management system. Next, I interviewed the CEO and the CTO of a software-based startup company (referred to as P2 and P3, respectively)–both were interviewed together. Their company was composed of sixteen people who work co-located on the design and implementation of a referral program software.

Participants were contacted to participate on the study using snowball sampling and had informed consent about the study. This research was conducted with the approval of the Human Research Ethics Board (HREB) of the University of Victoria.

6.1.2 Instrument

The creation of the questionnaire was done through three iterations. A summary of these iterations is presented in Table 6.1 First, we used the Model of Regulation to define a subset of questions. On the second iteration, an expert from learning science and another one from software engineering helped us to ensure the questionnaire was complete in terms of the model and to validate the vocabulary was clear and appropriate. These iterations led to refinement of the questions so they were easily

understood by participants. On the third iteration, through two pilot interviews only one question was reworded arriving at the instrument shown in Table 6.2. The processes of regulation are highlighted in bold text and followed by a short definition. In subsequent rows, I identify the mode and scope of regulation: Self-regulation refers to *individual* processes; co-regulation describes how individual strategic responses are modeled by the participant's *interactions* with their social context; and shared regulation refers to the joint processes experienced by the *group*. The questions shown in normal font are designed to ask about the existence of the activity, while the questions in italic font inquire about documentation and tool support.

	Iteration I	Iteration II	Iteration III
What?	A subset of questions for every aspect of the Model of Regulation was created. Then, the subset of questions was shortened by condensing questions until the questionnaire was generally applicable.	(a) Checked the questionnaire completeness, in terms of the aspects covered on it, with respect to those included in the Model of Regulation. (b) Reviewed and refined the vocabulary used in the questionnaire.	Tested the questionnaire by conducting two pilot interviews to software developers.
Why?	To reduce the amount of questions in the subset. Initially, the subset was composed by 90 questions making the questionnaire too long to be applicable.	(a) To ensure Iteration I, where the initial subset of questions was shortened and condensed, had not removed important components of the Model of Regulation. (b) To ensure the vocabulary used in the questionnaire was clear and understandable for software engineering.	(a) To validate the vocabulary used in the questionnaire was clear and appropriate. (b) To confirm the usability of the questionnaire to investigate collaboration practices and tool support.
Who?	The researchers involved in the study	The researchers as leaders of the process. In addition, one expert from the learning science helped check the questionnaire completeness and vocabulary, and one software developer familiar with the Model of Regulation helped to review and refine vocabulary.	I conducted the interviews. Interviewee 1 had approx. 4 years of experience in software development and recently had worked for a software-based company as a CO-OP for eight months. Interviewee 2 has approx. 7 years of experience working for well established software-based companies as well as startups.
Outcome:	Questionnaire of 21 questions.	Refined questionnaire of 21 questions.	The instrument

Table 6.1: Sumary of the iterations followed in the creation of the instrument.

Task Understanding	**Unifying perceptions about the project/task at hand. Project comprehension or understandings include: requirements, purpose, scope, social context, and roles and responsibilities of participants with respect to the task.**
Self-regulation: Individual	Do you take the time to understand the task at hand and the project within its particular context? *Where is it documented?*
Co-regulation: Interactions	Do you discuss your project comprehension with other team members and search for alignment of ideas? *Where are these discussions documented?*
Shared regulation: Group	Does the group hold discussions and reach agreement about the project understanding? *Is the result of group discussions documented and available to all members? Where is it?*
Goal Setting	**Establishment of a work plan, which includes goals, task standards (e.g., deadlines, product quality), resource allocation, strategies, methods and tools to support collaboration and task performance.**
Self-regulation: Individual	Do you define a personal work plan? *Where is it documented?*
Co-regulation: Interactions	Do you assist other team members to define or improve their personal work plans? *Is the outcome of these conversations documented? Where?*
Shared regulation: Group	As a group, do you define and agree on a group work plan? *Does the group document the agreement about the work plan and make it available for all members? Where?*
Enacting	**Follow the plan while providing support for motivational engagement**
	Plan execution
Self-regulation: Individual	Do you follow your personal work plan?
Co-regulation: Interactions	Do you support other team members to execute their work plans?
Shared regulation: Group	Does the group follow the work plan as defined? i.e., the group effectively uses collaboration strategies, methods and computer-based tools as selected in the work plan.
	Motivational engagement
Self-regulation: Individual	Do you implement strategies to stay motivated and engaged in the face of challenges?
Co-regulation: Interactions	Do you help other team members to stay motivated and engaged during plan execution?
Shared regulation: Group	As a group, do you discuss and agree on strategies for staying motivated and engaged in challenging situations?

Monitoring and Evaluating	**Tracking and assessment of project comprehension and the work plan**
	Against expected results and work plan
Self-regulation: Individual	Do you monitor and evaluate whether the outcome of your work is aligned with your project comprehension and the work plan? *Where are these evaluations documented?*
Co-regulation: Interactions	Do you help other team members to monitor and evaluate whether the outcome of their work is aligned with the project comprehension and with the work plan? *Where are these conversations documented?*
Shared regulation: Group	As a group, do you hold discussions to monitor and evaluate whether the outcome of the work is aligned with the project comprehension and the work plan? *Where are these discussions documented and available for all members?*
	Of changes in project comprehension or in the work plan. i.e., Verify that the project requirements are still the same, check that your responsibilities with respect to the project have not changed.
Self-regulation: Individual	Do you monitor and evaluate changes in your project understanding and in your work plan? *Where did you document the changes detected (if any)?*
Co-regulation: Interactions	Do you assist other team members to monitor and evaluate changes in the project understanding and in the work plan? *Where are these evaluations documented?*
Shared regulation: Group	Does the group hold discussions to monitor and evaluate changes in the project understanding and in the work plan? *Where are these discussions documented and available for all members?*
Adapting	**Refining the work plan based on partial outcomes or changes in the project understanding**
Self-regulation: Individual	Do you adapt your work plan when the outcome was not as expected or when your project comprehension changed? *Where did you document the changes to the original plan and the reasons behind those?*
Co-regulation: Interactions	Do you assist other team members to adapt their work plan when the outcome was not as expected or when their project understanding changed? *Where did you document the discussions about these adaptations?*
Shared regulation: Group	As a group, do you adapt your work plan when the outcome was not as expected or when the group project comprehension changed? *Where did you document the modifications to the original plan and the reasons behind those?*

Table 6.2: Instrument to profile collaboration based on the Model of Regulation

6.2 Findings: Using the instrument to reveal work pratices and tools

Using the instrument as a guide, the interviews allowed me to construct a profile of collaborative practices for each case. I present my observations below with the resulting profiles documented in tables, one for each regulation process (e.g., Task Understanding, Goal Setting, etc.). I summarize the findings for both cases side by side with P1's interview in the left-hand column of each table, and P2 and P3's interview in the right-hand column of each table.

6.2.1 Reflection on the Task Understanding process

The way Task Understanding is experienced by P1, P2 and P3 is different mainly because of their geographic distribution. P1's team does not have many opportunities for informal communication: *"There is no way you can just pop over the cubicle and say 'hey, what are you working on?' That sketch isn't right, you need to be working on this, this and this."* [P1] Their virtual environment does not naturally invite the casual chat typically present in face-to-face interactions, and as a result, group members have to make extra effort to stay up to date on the project and show themselves as being available and active on their team's communication channels. For example, the team holds a video conference meeting shortly after starting a task to ensure alignment of ideas, however, this usually fails because teammates do not review the tasks ahead of time.

For self-regulation, P1 takes the time to analyze the assigned task before each meeting, sketching flowcharts and making personal notes of what the task implies and the possible solutions. The team uses Slack for most communication (if the

discussion is not overly complex), otherwise they host video calls through Google Hangouts. Formal documentation is created *"if the task involves management and we are going to need a reference in the future or it is the kind of task that I can see us forgetting."* [P1] Documentation exists on the company internal/external wiki or on Google Docs, depending on the target audience.

Meanwhile, P2 and P3 take the time to think about a task and create personal documentation only if the task is perceived as complex. Also, P2 and P3 described that co-regulation related to Task Understanding occurs naturally in daily communication through their tools and face-to-face conversations, while collective understanding (shared regulation) is achieved in group meetings. The complexity of the project determines the length or number of meetings—the more complex the project, the longer it takes to reach consensus. Also, meetings can be internal or open to stakeholders who *"come in [to share] problems, what's going on, and how customers are using the system."* [P2] For P2 and P3, Slack is their main communication tool and the source of raw documentation. Agreements are formally documented on Google Docs where they usually capture the final ideas and what was agreed on. *"Quite often there's sort of a requirement sheet, but it's not like a very strict format of requirements."* [P3] A summary of these practices is illustrated in Table 6.3.

6.2.2 Reflection on the Goal Setting process

Continuous project changes prevent P1's team from creating long-term plans. Therefore, their Goal Setting process focuses on immediate tasks. *"We don't have clear requirements... but individual task level is a lot better because we break things really small and I could tell you what it's going to look like in a couple weeks."* [P1] On P1's team, they experience shared regulation in their daily stand-up meeting, where each participant reports the work they have finished and what they plan to do that day.

	Distributed Team Practices and Tool Support	Co-Located Team Practices and Tool Support
Task Understanding		
Observation	**This team is geographically distributed and they find it harder to engage with each other. Their virtual environment presents extra challenges when trying to understand and reflect on tasks.**	**This team is co-located and routinely takes the time needed to understand tasks because it is easy to discuss things in person. However, they tend to dedicate less time to reflect as task understanding is more routine and easy to engage in.**
Self-regulation	P1 makes sure to take the time needed to understand the task at hand and create sketches of possible solutions. Tools: Flowcharts, personal notes.	P2 and P3 think about tasks before formulating a plan. They create formal documentation if the task is perceived as complex.
Co-regulation	The team uses specific communication tools due to their geographic distribution. Tools: Slack, Google Hangouts.	The team discusses matters at a product startup meeting, which is the first meeting held with the team after a project has been accepted. Tools: Face-to-face discussions, Slack.
Shared regulation	The team holds video conference kick-off meetings shortly after starting a task. This usually fails because teammates do not review the tasks ahead of time. Tools: Slack (if the discussion is not overly complex), otherwise video calls through Google Hangouts. Formal documentation is created under certain criteria.	Collective understanding is achieved in group meetings, which can be internal or open to stakeholders. Tools: Slack (raw documentation). Agreements are documented on Google Docs.

Table 6.3: Task Understanding—The Model of Regulation in action: work practices and tool support of practitioners in two different organizations. P1 works on a distributed team and P2 & P3 work on a co-located team.

In preparation for the meeting, P1 creates a report with the help of personal notes, to-do lists on Wunderlist, and the configuration of task reminders through a Slack bot.

For co-regulation, P1 uses Slack to check on everyone's planning activities and maintain a record of the conversations that have occurred. This checking is more likely to happen when there are dependencies between P1's tasks and someone else's work. Also, P1 uses Google Hangouts with the same usage criteria as for the Task

Understanding process; other team members use a tool similar to Trello.

In the case of P2 and P3, their Goal Setting process is defined at different levels: *"At the higher levels, responsibilities are weighted out and we know what are we going to do, and at the lower level, what comes out is that each task is assigned to a person and it's documented by the leader they work with."* [P2] For co-regulation, the team holds weekly meetings and the team leaders checkin through Slack daily. If team members need help, they create a direct message through Slack or request a face-to-face meeting. For shared regulation, each team leader creates a work plan, and in the weekly meeting, each team member describes what they will get done during the week. Plans are divided into individual goals and each is configured as a Git issue, which is later visualized using Waffle.io.

For P2 and P3, their working culture defines how to approach planning. For example, P3 describes their flexible policies about deadlines: *"If it takes an extra day, but you will feel happier with the quality and this is acceptable for the customers, go for it."* Interestingly, P2's current working culture slowly emerged over time: *"Coming into that standing order was a 3-hour debate back in the day."* Today those shared agreements (e.g., quality standards, regular procedures) are followed by all members and make the work easier. A summary of the collaboration experiences for the Goal Setting process of regulation is presented in Table 6.4.

6.2.3 Reflection on the Enacting process

In both projects, interviewees highlighted that their team purpose is to always follow the plan, but unexpected things usually come up: *"The important thing that I bring out immediately is that we have this plan, something that we are meant to do, and then without fail, something will come up."* [P2]

In particular, this situation is more often experienced by P1's team. *"One day I*

	Distributed Team Practices and Tool Support	Co-Located Team Practices and Tool Support
Goal Setting		
Observation	**The team does not have access to higher-level strategic plans and their requirements change rapidly. Because of this, they must create small, specific goals.**	**The team places a high focus on product quality rather than making a deadline.**
Self-regulation	In preparation for each daily stand-up meeting, P1 creates a report. Tools: Personal notes, to-do lists on Wunderlist, and configuration of task reminders through a Slack bot.	P2 and P3 described that goal setting happens at a high level where tasks are weighted out and decided on, and at a low level where the tasks are assigned.
Co-regulation	P1 checks on everyone's planning activities (usually when there are dependencies between P1's tasks and someone else's work). Tools: Slack, Google Hangouts (usage criteria is the same as the Task Understanding process), some people use a tool similar to Trello.	The team holds weekly meetings and the team leaders check in through Slack daily. If team members need help, they create a direct message through Slack or request a face-to-face meeting. Tools: Face-to-face discussions, Slack.
Shared-regulation	The team holds daily stand-up meetings. Tools: Slack, Google Hangouts. Formal documentation follows the same criteria as in the Task Understanding process.	Each team leader creates a work plan and developers plan their work based on that. Plans are divided into individual goals. A Git issue is created for each goal and goals are later visualized using Waffle.io. Tools: GitHub issues, Waffle.io.

Table 6.4: Goal Setting—The Model of Regulation in action: work practices and tool support of practitioners in two different organizations. P1 works on a distributed team and P2 & P3 work on a co-located team.

think I know what's going on with the project and the project comprehension seems pretty good, and the next day someone says something and then I have no idea what I'm building." [P1] To help with task enacting, the team has a Slack channel named 'help request' to *"report issues if we are stuck and we're having troubles with the task we said we were going to complete that day."* [P1]

In the case of P2 and P3, their task enacting is basically making sure everyone on the team is on track, which is expected from their administrative positions. For co-regulation, P2 and P3 also reported the active use of Slack to support their indi-

vidual work and their teammates. The team helps each other execute their individual tasks using a 'developer channel' Slack channel, which has the purpose of *"trying to encourage people to broadcast what they are doing and encourage feedback and cross-communication."* [P3] Table 6.5 presents the summary of the work practices of both projects on Enacting - Plan execution.

	Distributed Team Practices and Tool Support	Co-Located Team Practices and Tool Support
Enacting		
Plan execution		
Observation	**The team's goal is to follow the plan agreed upon in the daily stand-up meeting, however, it is not unusual to encounter unexpected problems.**	**Even though the plan is accurate, sometimes during plan execution things get modified due to the dynamic nature of software projects.**
Self-regulation	P1 tries to follow their work plan but requirements tend to change often (within hours), which changes the expected output of their work.	P2 and P3 make sure everyone on the team is on track.
Co-regulation	Team members use the Slack channel 'help request'.	The team uses the Slack channel 'developer channel'.
Shared regulation	Same as the co-regulation process for Enacting - Plan execution.	Same as the co-regulation process for Enacting - Plan execution.

Table 6.5: Enacting, plan execution—The Model of Regulation in action: work practices and tool support of practitioners in two different organizations. P1 works on a distributed team and P2 & P3 work on a co-located team.

When discussing strategies for motivational engagement, P1 reported having a supportive culture in the team although this has never been explicitly formalized among members. P1 highlights that the important thing is *"feeling everybody wants to help you... knowing that you're not given a task and sent alone to a room."* [P1] In this sense, all teammates help each other, promptly answering questions on Slack, providing guidance when needed, celebrating when the work is done, and trying to keep everybody engaged in long video calls.

In the case of P2 and P3, they highlighted the role of all team members for keeping motivation and engagement: *"People are very crucial, to help each other and even*

feel sympathetic when it is really shitty." [P2] P2 and P3 believe their team's shared regulation and co-regulation is influenced by their culture: *"It's uncompetitive and so people are very much willing to help and they care about each other's happiness."* [P3]

In particular, P2 and P3 described the way they contribute to the team's motivation: *"In regular communication we try to decide if [the employees] are happy with the things they are working on. If not, what would make them happy and try to find opportunities..."* [P3] Also, team members encourage mutual support: *"Trying to use a bit of thanks and cheering upon the [Slack] channel when things are going well."* [P2].

For self-regulation, P3 described how doing things in a more creative way helps their motivation —they also use the motto 'action precedes motivation': *"It's basically I really don't feel like doing something but just do it for five-minutes and usually after that five minute period all of the sudden you're like in the groove."* While P2 uses the 30-minute Pomodoro Technique: *"You tell yourself I have to work for 30 minutes and after 30 minutes I get a treat... although to be honest, at the end of that, I'm in the zone so much I don't stop."* The Enacting - Motivational engagement experiences for both projects are summarized in Table 6.6.

6.2.4 Reflection on the Monitoring and Evaluating process

In the case of P1, requirements change often, sometimes within a couple of hours. Plans are constantly modified and no plan is ever complete. *"It's weird because no one knows anything, not even what the project is going to look like in a month."* [P1]

As part of self-regulation, P1 monitors critical channels on Slack where developers, product owners, and people that make feature requests talk to *"summarize all that into some sort of idea of what's going on with the application"*. To make this

	Distributed Team Practices and Tool Support	Co-Located Team Practices and Tool Support
Enacting		
Motivational engagement		
Observation	**There is no explicit agreement on strategies for staying engaged and motivated, however, the team appears to believe in mutually supporting each other.**	**All team members contribute with strategies and support for motivational engagement.**
Self-regulation	P1 tries to keep a positive attitude.	P3 tries to do things in a creative way and regularly thinks of the motto 'action precedes motivation'. P2 uses the 30-minute Pomodoro Technique.
Co-regulation	All teammates help each other, promptly answering questions on Slack, providing guidance when needed, celebrating when the work is done, and trying to keep everybody engaged in long video calls.	The team has a culture of mutual support promoted by the uncompetitive environment.
Shared regulation	There is no collective discussion about motivational and engagement strategies, but all teammates help to maintain the supportive culture.	P2 and P3 make sure everyone is happy with the things they are working on. Also, team members encourage a culture of mutual support on the communication channels.

Table 6.6: Enacting, motivational engagement—The Model of Regulation in action: work practices and tool support of practitioners in two different organizations. P1 works on a distributed team and P2 & P3 work on a co-located team.

monitoring task more efficient, P1 configured Slack with a list of keywords so that they are notified when any of those words are mentioned in a conversation.

Also, P1 detects opportunities for co-regulation with the keywords configured in Slack, creating *"a newsfeed for every time any of those things get mentioned."* This allows them to easily get context around each notification and jump in to provide direction if something wrong or incomplete is detected in team conversations.

For shared regulation, the size of a project can make monitoring it a challenge, so P1 and the other product designers on the team strategically off-load certain things between each other. *"When talking to the three of us, you can probably get a good idea of what the entire picture of the project is."* [P1] In case changes are required,

the approach is top down: *"Usually the big bosses decide what they want changed, then the top managers talk about it. Then they call the design team in and maybe two developers to ask 'what do you guys think?' Once we have an agreement, a new plan is created."* [P1]

P1's team has to put a lot of effort into keeping up with changes. As a result, P1 and their team experience monitoring and evaluating of both *expected results* and *changes to task understanding* at the same time. This can be seen in the left column of the summary Table 6.7.

In the case of P2 and P3, their self-regulation is achieved through monitoring the work of others. In this respect, they recognize that monitoring progress is *"an implicit and unspoken process"* [P2], of which the tools play a great role. *"Anyone in the company can go and see in the [Waffle.io] board where the project is, and that actually helps to reduce a lot of communication overhead."* [P2]. For co- and shared regulation, P2 and P3 highlight that the size and makeup of the team facilitates the monitoring of tasks as *"people are just interested in each other... we are still a small team, most people know roughly the plan for anyone in the project."* [P2]. In particular, P2 and P3 have a 3-dot system to monitor their projects. Once a month *"we report if [the project] is red, yellow, or green—meaning 'way off-track', 'on-track', or 'it's done'."* [P2]. At a lower level, the team also uses Waffle.io to visualize the status of their GitHub issues.

P2 and P3's team also experiences changes: modifications of their plans, customers changing requirements, and approaches that did not work out as expected. However, these changes are usually spread out over time, which make it easier for the team to clearly distinguish between monitoring and evaluating their progress compared to the planned results (summary reported on the right column of Table 6.7), and monitoring and evaluating changes to their task understanding (summary reported on the right

column of Table 6.8).

When discussing P2 and P3's practices for monitoring changes to their project understanding, they described how this is also a natural process that is part of their everyday work: *"Something comes up, and you react, and you keep reacting."* [P2] For co-regulation, the team monitors and evaluates in an organic way as *"the conversations between two counterparts kind of naturally leads to checking in with the understanding and plans."* [P3] Also, *"we may review a project and then through that process discover 'oh you just said something that sounds different to me', and we would follow up on that."* [P2] Finally, any new product of the team's interactions are documented and shared with everyone.

6.2.5 Reflection on the Adapting process

Software projects are characterized by their changing nature and loosely defined scope, which sometimes requires adjustments to the work even after a plan has been set. The interviewees showed they are aware of this fact and reported established practices they follow to make adaptations to their work.

P1 adapts their personal plan every time a change is detected and makes active use of the communication channels to stay informed. P1 mentioned how new plans or adaptations are defined by the project managers and the team has an established practice for handling these situations. If an adaptation is required, the practice is to record notes at the bottom of the applicable documents or create visible comments on the specific part. However, some managers do not follow this policy and impede the team's shared regulation. *"Sometimes they would have not done the proper homework ahead of time"*, meaning the original plan is incomplete and it looks bad for them to leave specifications as visible changes. So, *"they would directly edit the original bit, which won't show the change."* [P1]

	Distributed Team Practices and Tool Support	Co-Located Team Practices and Tool Support
Monitoring & Evaluating		
Against expected results and work plan		
Observation	**The team finds it difficult to monitor and evaluate progress as project scope is typically unknown and requirements are constantly changing.**	**Team awareness is facilitated by computer-based tools which play a major role in collaboration.**
Self-regulation	P1 monitors critical channels on Slack where developers, product owners, and people that make feature requests talk about and summarize the information to stay updated. Tools: Notification of keywords configured on Slack.	For P2 and P3, individual regulation is accomplished by monitoring the progress of others, which is perceived as a natural task that takes place on every open conversation and meeting.
Co-regulation	P1 has a newsfeed for every time a configured keyword is mentioned in Slack, which allows them to easily jump in to provide direction if something wrong or incomplete is detected in team conversation. Tools: Notification of keywords configured on Slack	The small size and makeup of the team facilitates the monitoring of tasks.
Shared regulation	P1 and the other product designers on the team strategically off-load certain things between each other. In case changes are required, the approach is top down: from the bosses to the managers, and then to the developers.	P2 and P3 have a 3-dot system to monitor their projects, and at a lower level, the team also uses Waffle.io to visualize the status of their GitHub issues.

Table 6.7: Monitoring & Evaluating, against expected results and work plan—The Model of Regulation in action: work practices and tool support of practitioners in two different organizations. P1 works on a distributed team and P2 & P3 work on a co-located team.

In the case of P2 and P3, adaptations are a regular part of the work and documentation is created only if *"we think it's important and useful for others to know."* [P3] Adaptations are documented on Google Drive most of the time and *"we would document changes with comments and stuff like that, we would annotate or just change the document... we would document what we wanted to do, but not necessarily why, unless it's really important to the future task understanding."* [P3] Interestingly, increasing team complexity (team size) is affecting how much documentation is produced: *"As we are growing, we are doing more documentation."* [P3]

	Distributed Team Practices and Tool Support	Co-Located Team Practices and Tool Support
Monitoring & Evaluating		
Of changes in project understanding or work plan		
Observation	**Due to the frequency of changes to requirements, the team's monitoring and evaluation of planned progress and changes in task understanding occur at the same time.**	**Open communication between teammates and stakeholders allows participants to stay in the loop as the understanding of the project evolves.**
Self-regulation	Same as the self-regulation process in 'Monitoring and Evaluating - Against expected results and work plan'.	P2 and P3 continuously monitor changes in their task understanding.
Co-regulation	Same as the co-regulation process in 'Monitoring and Evaluating - Against expected results and work plan'.	The team monitors and evaluates their everyday interactions in an organic way.
Shared regulation	Same as the shared regulation process in 'Monitoring and Evaluating - Against expected results and work plan'.	Any new agreements are documented and shared with the team.

Table 6.8: Monitoring & Evaluating, of changes in project understanding or work plan—The Model of Regulation in action: work practices and tool support of practitioners in two different organizations. P1 works on a distributed team and P2 & P3 work on a co-located team.

A summary of the practices of Adaptation for the two cases is presented in Table 6.9.

6.3 Reflection on the Model of Regulation

The application of the instrument allowed me to have a rich discussion about the affordances and constraints of team collaboration practices and tool support.

Through the instrument, the Model of Regulation was presented to the interviewees, who found its common language to be useful. *"Often we find that a common language is what's needed to have a constructive conversation. Once you build a language you can start talking about how to move from one step to the other and how improving what we're doing in one step... we have a way to verbalize it formally."*

	Distributed Team Practices and Tool Support	Co-Located Team Practices and Tool Support
Adapting		
Observation	**The team's task understanding and plans are re-adapted every couple of hours due to constant changes in project requirements.**	**Adaptations are made everyday, however, not every change is documented. This depends on the possible number of people the change affects and whether the information might be important in the future.**
Self-regulation	Constant changes to the project require P1 to also adapt their individual plans. Tools: Slack and Google Hangouts to communicate changes and updates.	Individual adaptation was not explicitly discussed during the interviews.
Co-regulation	Same as the self-regulation process listed above.	If a change is required, it is implemented and documentation is not always created.
Shared regulation	New plans or adaptations are defined by the project managers, who should follow established practices to document the changes. However, some of them do not follow the guidelines.	Adaptations are documented on Google Drive most of the time. Interestingly, the increasing team complexity (team size) is also increasing the amount of documentation produced.

Table 6.9: Adapting—The Model of Regulation in action: work practices and tool support of practitioners in two different organizations. P1 works on a distributed team and P2 & P3 work on a co-located team.

[P2] P3 also recognized that the model does not add new steps to their daily work, but instead provides mechanisms to formalize and give structure to their existing practices as the concepts of the model are *"something that you kinda understand at some level, like in an abstract way... when you see [the model], you start to see how it fits with things you've already experienced."* [P3] Furthermore, P2 appreciated the ability to formalize the modes of regulation (self-, co- and shared) because *"it removes the ability to kinda escape ownership of the work... some people are used to say[ing] 'I don't really know my work plan, but my group's got it figured it out, I'm sure it's fine'."*

More interestingly, a couple days after the interview, P1 commented that they had started to think about the issues they were having with their teammates in

terms of the model. P1 was using the vocabulary of the model they had learned from the interview in order to describe the reasons for friction and breakdowns in team collaboration. For example, P1 mentioned a coworker that *"was very bad at spending the least amount of time possible to understand the task; they would automatically assume they understood exactly what needed to be done, and would go off on their own and work away...usually they had completely misunderstood the task."* P1 admits they *"obviously had some sort of disconnect with task understanding and it wound up costing a lot of time / resources."*

6.4 How this study changed the Model of Regulation?

Through this study the basic concepts in the Model of Regulation were not modified, their definition was only refined to be more concise and clear.

The goal of this study was to show how the Model of Regulation could be transformed into an action-oriented instrument for practitioners and researchers. To achieve this goal, the instrument a) had to be complete in terms of the model and b) had to be short enough to be easily applicable (i.e., in the first iteration the instrument had ninety questions and was approx. 6 pages long with the processes of regulation definition included). During the development of the instrument, I and a team of supporting researchers iterated over the questionnaire to ensure both criteria, which resulted in the refinement of the presentation of the processes of regulation —no construct meaning was altered or modified in this study—.

In sum, this study helped to create a more concise, precise and clear definition for the processes of regulation.

6.5 Discussion and conclusions

In this chapter, I presented the usefulness of the Model of Regulation by creating an instrument and applying it to collect information about collaboration practices and tool support in two software projects. Using the instrument as a guide for my interviews, I discussed how the modes and processes of regulation were experienced by the interviewees. This also allowed me to see how team and task complexity are key variables for both software projects in the decision making process, a fact already described by [21].

The main differences found in the two projects I investigated are due to their geographic distribution. While P1 worked in a virtual environment, P2 and P3 were in a co-located setting. However, it is important to notice that their perspectives were also different. P1 was a developer and UX designer actively involved in the software development process who usually had their tasks assigned by the leaders. While P2 and P3, CEO and CTO of the company, respectively, were at the top of the administrative hierarchy and had more control over their tasks. Naturally, each interviewee reported what they perceived as a collaboration practice from their particular role.

Thus, it is not possible to draw a complete profile of practices from a single application of the instrument, but it can start to reveal some of their work practices. Each role could have something extra to add to the group's collaboration profile. In that sense, the instrument is a facilitator for investigating collaboration and the richness of the data collected depends on the methodology selected by the researcher or the practitioner applying it.

Furthermore, the analysis of work practices and tool support reported in this chapter serve as a showcase of the instrument's potential to elicit existing collabora-

tion practices and providing a framework to classify tool support. This instrument can also be applied to a high scale, for example, to analyze open-source software communities, and guide evaluations of collaborative practices within or across units of collaboration.

Finally, the introduction of the Model of Regulation to software engineering is an important output from my research, but I'm more excited about the ways we can make this theory action-oriented (like with the instrument) to get insights into the practices and tools required to support regulation in a particular development context. Using the instrument, we can now have discussions about the affordances and constraints of collaborative practices and tool support. And the model can be used to provide explanations as to why a particular technological ecosystem is formed around a task and to understand the role of each tool (something we struggled with before having this model, for example: why do developers use Slack?).

Chapter 7

Discussion and future work

In this chapter, I discuss the findings of this research, their limitations and also outline interesting areas for future work.

7.1 Discussion

First, I show how the concepts combined into the Model of Regulation are interpreted and applied differently in a software engineering context compared to a learning context. Also, I describe how the model could be applied to other knowledge worker domains.

Next, I explore contexts beyond the team to see how the model can be applied at the organization or community level through the instrument, areas where I feel there is great potential for both, the model and its instrument.

As presented earlier, current collaboration models that have been applied to software engineering emphasize only a particular subset of collaboration concerns. The goal in composing the Model of Regulation was to arrive at a model that considered more of the important dimensions of collaboration—behavior, cognition, and motivation. Through the discussion, I explore how this more comprehensive model provides the insights and vocabulary needed to capture the theoretical underpinnings of why

certain tools and practices are used.

7.1.1 From the learning science domain to software engineering context and beyond

I borrowed constructs of the theory of regulated learning from the learning science domain and adapted them to software engineering. While learning is crucial for facilitating the emergence of regulation in both domains, through the studies conducted and described in this thesis I identified that there are important differences in the way regulation is experienced (cf. Winne and Hadwin [113]). In software engineering, tasks tend to be more open ended and evolve constantly. The complex nature of the tasks and knowledge manipulated as well as the diversity of stakeholders involved and their motivations to participate in projects vary widely. For instance, in the learning science domain regulation has been studied only between students, while in software engineering we recognize a large diversity of stakeholders that define the way regulation is experienced —as I noticed during the case study on the Neo4J community (see Chapter 5). Also, to study regulation processes (e.g., Task Understanding, Enacting) in software engineering contexts I required to identify the typical characteristics of the domain artifacts manipulated in each process. So, while in formal learning environments Task Understanding process refers to the comprehension of the tasks assigned, in software engineering this process refers to the understanding of project requirements, project purpose, scope, social context, etc. These differences appear to be rather subtle—or may appear to be variations along a common continuum—-but they forced me to change the definitions of the modes and processes of regulation. For details on how the Model of Regulation and its concepts were refined refer to Sections 5.6 and 6.4 from Chapters 5 and 6 respectively.

This thesis borrows and combines constructs of regulation —modes, processes,

and tool support— into a single model and adapts them to fit the software engineering domain. I explicitly discuss how the different processes activate other processes. This thesis also describes how through the use of the model it was possible to derive an "action-oriented" instrument for eliciting how regulation occurs in software development teams. Interestingly, colleagues in the learning sciences have indicated to me that a similar instrument could prove to be very useful in the learning science domain to interview and question students about their learning strategies and techniques. Thus, composing the model and adapting it to software engineering has brought potential benefits to the learning domain as well.

Although my research did not investigate the applicability of the model in other knowledge domains I believe the model has potential for it. Indeed, software developers are often described as the "prototype of the future knowledge worker" [1] as they tend to innovate and adopt tools before other knowledge workers do (e.g., email, Wikipedia, and GitHub). I suggest applying a further adapted model and instrument to other domains would bring important insights.

7.1.2 Beyond the team: Understanding collaboration in organizations and communities

My work proposes that the Model of Regulation can be used to describe collaboration practices across any unit of collaboration, from small teams, to project-wide teams, to communities of thousands of participants (e.g., the *Ruby on Rails* project has more than 2700 contributors[2]). In this thesis, I have discussed how the model can be applied to small industrial teams through the use of the instrument (see Chapter 6). And I also showed how it can be applied to the Neo4J open source development

[1] `http://allankelly.blogspot.ca/2014/04/theprototype-of-future-knowledge.html`
[2] `https://github.com/rails/rails`

community (see Chapter 5). In the Neo4J case study, I used the model to describe how stakeholders from a wider community can "regulate" development activity. For example, users of the Neo4J graphing technology (through GitHub and their Google Groups channels) give input to and request features from the core Neo4J developers. I believe the instrument could also be useful in future analysis of this wide community by helping to structure interviews to study their collaboration practices.

Through the case study where the instrument was developed and applied, the vocabulary provided by the model allowed me and the practitioners I interviewed to have rich discussions about collaboration as we could make reference to very specific processes related to an activity (e.g., Task Understanding, Enacting) at different levels (self-, co-, shared). These discussions allowed me to reflect on collaboration beyond the individual and team levels and to understand where practices and tools could be improved. One of the interviewees mentioned that they found many benefits from using this vocabulary and they continued to reflect on their practices and tool use some time after the interview.

7.1.3 From a theory to actionable principles, work practices, and tools

The model presented in this thesis is based on the concept of regulation, which considers collaboration at a metacognitive level and explores behavior, cognition, and motivation. This Model of Regulation can be used as a descriptive model as it allows one to describe how regulation occurs within a collaboration unit and how tools support regulation. However, I believe that this model also shows promise as a prescriptive model that can be used to guide processes and suitable tool support for practitioners. The modes of regulation illustrate how collaboration requires individual efforts, interactions with other participants, and an awareness of the group's state

(an important factor as discussed by Gutwin *et al.* [43]).

The processes of regulation provide a high-level guideline for tasks and suggest how tools can help support regulation. And the model's tool support component helps to connect the theory with tool design principles, which is something I find interesting to further explore in future work. An example of tool support analysis is presented in Section 4.3 of Chapter 4. Furthermore, I believe the instrument drawn from the model can be used to guide evaluations and comparisons of collaborative practices within or across a single or multiple units of collaboration. For instance, the instrument can be applied to elicit the collaboration practices and build a collaboration profile of several open source development communities (like Neo4J) and later their profiles can be compared for analysis.

While the theory is an important output of this research, I am particularly excited about the action-oriented principles and insights it provides regarding the practices and tools required to support regulation in software development. For example, regulation tool support—an important part of the model—can be used to provide explanations as to why a particular technological ecosystem is formed around a task, allowing practitioners and researchers to understand the role of specific tools, something we have struggled with (such as why developers use Slack in addition to many other communication channels [69]).

Using the Model of Regulation, tools and channels can also be analyzed and classified based on the processes of regulation they support. For instance, Trello is effective for Goal Setting processes through mirroring and awareness mechanisms, while Slack supports conversations that lead to the alignment of ideas, thus providing support for Task Understanding. Furthermore, applying the instrument allows us to describe collaborative practices and tool support for any unit of collaboration, even at the community level.

7.2 Limitations

I recognize that the Model of Regulation composed and adapted for software engineering may need further refinements when applied to future projects and development contexts. In particular, the model does not include considerations for the different stakeholders that may be involved (such as testers) and it may not have concepts to encompass the diverse nature of interactions that take place in a project nor consider other factors (such as time zones, cultural issues and language in global software development). However, the iterative approach used in refining the model was effective at helping me understand the benefits of the many different tools developers use [100], as well as helped me understand the regulation activities of users in an open source community.

The usefulness of the model was in part validated by application of the derived instrument through the case studies performed on two projects with three developers. The instrument was able to bring insights on how regulation activities occurred across these two contrasting projects (one co-located setting and one distributed setting). However, it should be noticed the studies described in this work are not enough to generalize the application of the Model but I strived to offset this limitation by checking the completeness of the instrument with two domain experts as I developed it and piloted the instrument with two other developers before conducting the two case studies where I applied the instrument. As more interviews are conducted I do expect the instrument to go through at least some minor refinements in the questions it asks. The refinements to the instrument may in turn lead to changes in the Model itself.

Through the research so far, I recognize the model does not have mechanisms to conduct a quantitative analysis of productivity but perhaps integrating methods from

the PSP and TSP approaches [50, 51] can help to overcome this limitation. Using the Model of regulation may help in revealing variables of interest (e.g., how frequently tools are used to regulate activities) that could be measured quantitatively.

7.3 Future work

Next, I outline areas for future work based on the findings from each study described in this thesis.

7.3.1 The Model of Regulation as a mechanism to evaluate tool support

Section 4.3 of Chapter 4 presents the analysis of five GitHub features using the Model of Regulation. While only the GitHub social coding platform was analyzed similar analysis can be conducted with other types of tools. Future work can focus on the analysis of other social coding platforms (e.g., BitBucket) or on other types of tools, for example, management and collaboration task boards (e.g., Trello, ZenHub), software development-focused communication tools (e.g., Slack, HipChat), and personal and team metric tools (e.g., WakaTime).

Evaluation of the collaboration support provided by the myriad of tools available today will allow a deeper understanding of their features and provide explanations as to why some technologies are easily adopted. In this sense, future research would be mainly descriptive as the Model of Regulation would be used as a mechanism to asses tool support. Moreover, the Model of Regulation could also be adopted early in the tool development process and be used to provide theoretical grounding for design and implementation depending on the tool's purpose. Certainly, an interesting direction for future research could be to investigate using the model as a prescriptive framework

for tool development.

7.3.2 The Model of Regulation as a lens to characterize software development communities

Chapter 5 presents a case study that uses the Model of Regulation to describe collaboration in an open source software development community. Future research in this area could use the methodology followed in this study to investigate regulation in other types of software projects (e.g., commercial projects). The application of the model to other software engineering contexts will allow us to discover the way collaboration is experienced from the perspective of regulation identifying common practices and unique characteristics for each different context.

Another area for future work could look at integrating the Model of Regulation with previous research about the activities developers [100, 115, 4] and users [17, 22] perform in software projects. The Neo4J community case study illustrated the need to identify the participant's role in order to describe regulation in the community. Using previous research will allow us to characterize participants of a collaboration unit beyond only users and contributors, which could also help to conduct a deeper analysis of their regulation and support for collaboration practices.

In the long term, the analysis of regulation in collaboration units will create profiles of collaboration practices for each software engineering context — future research could also focus on the aggregated analysis of these.

In addition, the Neo4J project only made use of three communication channels at the time of conducting our case study (GitHub, Stack Overflow and Google Groups). However, shortly after the end of the case study, they started to also use Slack[3]. An interesting research direction could be to investigate the role of tools in the regulation

[3] `https://neo4j.com/blog/public-neo4j-users-slack-group/`

of a software development community, and specifically, the benefit Neo4J gains from the use of Slack.

7.3.3 The Model of Regulation instrument as a tool to profile collaboration

Chapter 6 presents the instrument developed to investigate collaboration practices and tool support in software engineering projects. It presents the experience of applying the instrument to two software projects and illustrates the importance of the vocabulary introduced with the Model of Regulation. This vocabulary allowed us to discuss affordances and the constraints of their particular collaboration practices.

Modern collaborative software development has diverse units of collaboration and software projects are only one of them. Although the vocabulary used in the instrument, may suggest that it is meant for groups or teams, the instrument can be applied to any unit of collaboration (e.g., group, project, community). The potential of the instrument relates to the information it can collect and future research can make use of the instrument in other software engineering contexts to investigate their collaboration practices and tool support characteristics. Furthermore, this characterization of collaboration profiles in software engineering can eventually lead to best practices and recommended technology support for each case.

7.3.4 The Model of Regulation as an emerging model

The model presented in this thesis is based on constructs developed for the theory of regulated learning in the learning science domain. In order to represent a model of collaboration for software engineering, the Model of Regulation required adaptations —every different aspect of collaboration explored in the studies in this thesis refined the concepts of the model for software engineering. The Model of Regulation is an

emerging model, and as such, is a work in progress that will continue to evolve and be refined as more research about regulation is conducted in software engineering.

It should be kept in mind that the original theory on which the model is based on has been around for approximately 25 years in the learning science domain. In software engineering, the concept of regulation has just been introduced. The theory of regulated learning has much more to offer to the understanding of collaboration in software engineering. So, future work can focus on continue studying the theory of regulated-learning from the learning science domain and exploring ways their constructs can be extended and applied to software engineering contexts.

Chapter 8

Concluding remarks

In this chapter, I list the main contributions of this thesis.

- First, I extend existing research by introducing a new descriptive model of collaboration: The Model of Regulation, which adds new vocabulary to describe collaboration processes tool support.

  Chapter 4 introduces the theory of regulated learning from the learning science domain, illustrates the constructs that were borrowed from it, and presents the adapted version of the constructs that compose the Model of Regulation for software engineering. The process of adapting theory constructs from the learning science domain to software engineering context was through a series of studies (see Section 4.3 of Chapter 4, Chapters 5, 6) and the following remarks describe the additional contributions from each of them.

- Second, I showcase how the Model of Regulation can be used to describe collaborative tools in terms of their benefits to collaboration.

  In Section 4.3 of Chapter 4, I use the vocabulary of the model to describe the support for collaboration provided by five commonly used features (Issues, Graphs, News feed, Commit, Blame) of GitHub. This example shows the way

the model can be used to describe tool support and the same analysis can be replicated to understand other technologies.

- Third, I demonstrate the Model of Regulation can be used to explain collaboration in open source software development communities.

  In Chapter 5, I use the Model of Regulation to describe collaboration and the user-contributor interaction in the open source community created by the Neo4J project. The descriptive power of the model extends beyond open source development and the model can also be used to describe and analyze collaboration in projects of commercial nature.

- Fourth, I showed the Model of Regulation can be used to promote discussions in software engineering about constraints and affordances of collaboration practices and tool support.

  In Chapter 6, I use the model to develop a questionnaire that ask about every aspect of regulation and tool support. Then, I used the resulting instrument to conduct interviews and profile collaboration practices and tool support for individuals and teams. I highlight the instrument can be applied to any unit of collaboration. Moreover, the model can used to guide the creation of other action-oriented instruments.

- Fifth, I outline interesting areas for future research based on the findings and work described in this thesis.

  In Chapter 7, I describe different ways to carry this research forward and continue (a) evaluating tools and gaining a better understanding of their support for collaboration, (b) using the regulation lens to characterize software development communities, (c) applying the instrument to profile collaboration practices and tool support, and (d) refining the Model of Regulation.

Bibliography

[1] U. Abelein and B. Paech. Understanding the influence of user participation and involvement on system success — a systematic mapping study. *Empirical Software Engineering*, 20(1):28–81, 2015.

[2] Agilemanifesto.org.

[3] N. Ahmadi, M. Jazayeri, F. Lelli, and S. Nesic. A survey of social software engineering. In *Automated Software Engineering-Workshops, 2008. ASE Workshops 2008. 23rd IEEE/ACM International Conference on*, pages 1–12. IEEE, 2008.

[4] J. Aranda and G. Venolia. The secret life of bugs: Going past the errors and omissions in software repositories. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 298–308, Washington, DC, USA, 2009. IEEE Computer Society.

[5] M. Arciniegas-Mendez, A. Zagalsky, M.-A. Storey, and A. F. Hadwin. Regulation as an enabler for collaborative software development. In *Proceedings of the Eighth International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 97–100. IEEE Press, 2015.

[6] L. Augustin, D. Bressler, and G. Smith. Accelerating software development through collaboration. In *Proceedings of the 24th International Conference on Software Engineering*, pages 559–563. ACM, 2002.

[7] L. J. Bannon and K. Schmidt. Cscw-four characters in search of a context. *DAIMI Report Series*, 18(289), 1989.

[8] M. Boekaerts and M. Niemivirta. *Handbook of self-regulation*, chapter Self-regulated learning: Finding a balance a learning goals and ego-protective goals, page 417Ű450. Academic Press, 2005.

[9] J. Buder and D. Bodemer. Supporting controversial cscl discussions with augmented group awareness tools. *International Journal of Computer-Supported Collaborative Learning*, 3(2):123–139, 2008.

[10] J. M. Carroll, D. C. Neale, P. L. Isenhour, M. B. Rosson, and D. S. McCrickard. Notification and awareness: synchronizing task-oriented collaborative activity. *International Journal of Human-Computer Studies*, 58(5):605–632, 2003.

[11] M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley. Identification of coordination requirements: Implications for the design of collaboration and awareness tools. In *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*, CSCW '06, pages 353–362, 2006.

[12] K. S. L. Chan, N. Muthukrishna, and J. G. Borkowski. *Issues in the measurement of metacognition*, chapter A process-oriented model of metacognition: Links between motivation and executive functioning, pages pp.1–41. Lincoln, NE: Buros Institute of Mental Measurements, University of Nebraska., 2000.

[13] K. Charmaz. *Constructing grounded theory*. Sage, 2014.

[14] S. Cherry and P. N. Robillard. Communication problems in global software development: Spotlight on a new field of investigation. In *Proceedings of the International Workshop on Global Software Development, International Conference on Software Engineering. Edinburgh, Scotland*, pages 48–52, 2004.

[15] S. Cherry and P. N. Robillard. The social side of software engineeringŮa real ad hoc collaboration network. *International Journal of Human-Computer Studies*, 66(7):495–505, 2008.

[16] C. Cook. Collaborative software engineering: An annotated bibliography. Technical report, Department of Computer Science and Software Engineering, University of Canterbury, 2004.

[17] W. W. Cotterman and K. Kumar. User cube: A taxonomy of end users. *Commun. ACM*, 32(11):1313–1320, Nov. 1989.

[18] K. Crowston, H. Annabi, J. Howison, and C. Masango. Effective work practices for software engineering: Free/libre open source software development. In *Proceedings of the 2004 ACM Workshop on Interdisciplinary Software Engineering Research*, WISER '04, pages 18–26, 2004.

[19] K. Crowston, Q. Li, K. Wei, U. Y. Eseryel, and J. Howison. Self-organization of teams for free/libre open source software development. *Information and Software Technology*, 49(6):564 – 575, 2007. Qualitative Software Engineering Research.

[20] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1277–1286. ACM, 2012.

[21] G. A. Dafoulas, K. Swigger, R. Brazile, F. N. Alpaslan, V. L. Cabrera, and F. C. Serce. Global teams: Futuristic models of collaborative work for today's software development industry. In *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*, pages 1–10. IEEE, 2009.

[22] L. Damodaran. User involvement in the systems design process-a practical guide for users. *Behaviour & Information Technology*, 15:363–377, 1996.

[23] C. R. De Souza and D. F. Redmiles. The awareness network, to whom should i display my actions? and, whose actions should i monitor? *Software Engineering, IEEE Transactions on*, 37(3):325–340, 2011.

[24] J. DeFranco-Tommarello and F. P. Deek. Collaborative software development: a discussion of problem solving models and groupware technologies. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 568–577. IEEE, 2002.

[25] A. Dix, J. Finlay, G. Abowd, and R. Beale. Human-computer interaction: Pearson prentice hall. *Inc, England*, 2004.

[26] C. Dörner, J. Heß, and V. Pipek. Fostering user-developer collaboration with infrastructure probes. In *Proceedings of the 2008 International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '08, pages 48–44, New York, NY, USA, 2008. ACM.

[27] P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, pages 107–114. ACM, 1992.

[28] K. Dullemond, B. van Gameren, and R. van Solingen. Collaboration should become a first-class citizen in support environments for software engineers. In

Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on, pages 398–405, Oct 2012.

[29] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*, pages 285–311. Springer, 2008.

[30] M. S. Elliott and W. Scacchi. Free software developers as an occupational community: Resolving conflicts and fostering collaboration. In *Proceedings of the 2003 International ACM SIGGROUP Conference on Supporting Group Work*, GROUP '03, pages 21–30, New York, NY, USA, 2003. ACM.

[31] C. A. Ellis, S. J. Gibbs, and G. Rein. Groupware: some issues and experiences. *Communications of the ACM*, 34(1):39–58, 1991.

[32] M. Fowler and J. Highsmith. The agile manifesto. *Software Development*, 9(8):28–35, 2001.

[33] M. J. Gallivan and M. Keil. The user-developer communication process: a critical case study. *Information Systems Journal*, 13(1):37–68, 2003.

[34] M. A. Gerosa, H. Fuks, and C. Lucena. Analysis and design of awareness elements in collaborative digital environments: A case study in the aulanet learning environment. *Journal of Interactive Learning Research*, 14(3):315–332, 2003.

[35] GitHub Inc. About github. https://github.com/about. 2015.

[36] GitHub Inc. Github glossary. https://help.github.com/articles/github-glossary/. 2016.

[37] GitHub Inc. Github help: Be social. https://help.github.com/articles/be-social/. 2015.

[38] GitHub Inc. Github pages. https://pages.github.com/. 2015.

[39] G. Gousios. The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA, 2013. IEEE Press.

[40] G. Gousios, M. Pinzger, and A. van Deursen. An exploration of the pull-based software development model, 2013.

[41] G. Gousios and D. Spinellis. Ghtorrent: Github's data from a firehose. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 12–21. IEEE, 2012.

[42] J. Grudin. Computer-supported cooperative work: History and focus. *Computer*, (5):19–26, 1994.

[43] C. Gutwin, R. Penner, and K. Schneider. Group awareness in distributed software development. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 72–81. ACM, 2004.

[44] A. Guzzi, A. Bacchelli, Y. Riche, and A. van Deursen. Supporting developers' coordination in the ide. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pages 518–532. ACM, 2015.

[45] A. Hadwin, M. Miller, and E. Webster. Promoting and researching adaptive regulation in cscl: Scripting, visualization, and awareness tools. *Conf. of the European Association for Research on Learning and Instruction*, September 2013.

[46] A. F. Hadwin, S. Järvelä, and M. Miller. Self-regulated, co-regulated, and socially shared regulation of learning. *Handbook of self-regulation of learning and performance*, 30:65–84, 2011.

[47] A. F. Hadwin, M. Oshige, C. L. Gress, and P. H. Winne. Innovative ways for using gstudy to orchestrate and research social aspects of self-regulated learning. *Computers in Human Behavior*, 26(5):794–805, 2010.

[48] R. Hertz-Lazarowitz and I. Bar-Natan. Writing development of arab and jewish students using cooperative learning (cl) and computer-mediated communication (cmc). *Computers & Education*, 39(1):19–36, 2002.

[49] W. S. Humphrey. *A discipline for software engineering*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[50] W. S. Humphrey. *Introduction to the personal software process (sm)*. Addison-Wesley Professional, 1996.

[51] W. S. Humphrey. *Team Software Process (TSP)*. Wiley Online Library, 2000.

[52] S. Järvelä and A. F. Hadwin. New frontiers: Regulating learning in cscl. *Educational Psychologist*, 48(1):25–39, 2013.

[53] S. Järvelä, H. Järvenoja, and M. Veermans. Understanding the dynamics of motivation in socially shared learning. *International Journal of Educational Research*, 47(2):122–135, 2008.

[54] S. Järvelä, P. A. Kirschner, E. Panadero, J. Malmberg, C. Phielix, J. Jaspers, M. Koivuniemi, and H. Järvenoja. Enhancing socially shared regulation in collaborative learning groups: designing for cscl regulation tools. *Educational Technology Research and Development*, 63(1):125–142, 2014.

[55] C. Jensen and W. Scacchi. Collaboration, leadership, control, and conflict negotiation and the netbeans. org open source software development community. In *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*, pages 196b–196b. IEEE, 2005.

[56] H. C. Jiau and C. H. Kao. Assessing the efficacy of user and developer activities in facilitating the development of oss projects. *Journal of Software Maintenance and Evolution: Research and Practice*, 21(5):287–314, 2009.

[57] R. Johansen. *Groupware: Computer support for business teams*. The Free Press, 1988.

[58] D. W. Johnson and R. T. Johnson. *Learning together and alone: Cooperative, competitive, and individualistic learning* . Prentice-Hall, Inc, 1987.

[59] D. W. Johnson and R. T. Johnson. *Cooperation and competition: Theory and research.* Interaction Book Company, 1989.

[60] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 92–101. ACM, 2014.

[61] P. Kirschner, F. Kirschner, and J. Janssen. The collaboration principle in multimedia learning. *The Cambridge handbook of multimedia learning*, pages 547–575, 2014.

[62] L. W. Knowlton and C. C. Phillips. *The logic model guidebook: Better strategies for great results*. Sage, 2012.

[63] K. Kreijns, P. A. Kirschner, and W. Jochems. The sociability of computer-supported collaborative learning environments. *Educational Technology & Society*, 5(1):8–22, 2002.

[64] F. Lanubile, C. Ebert, R. Prikladnicki, and A. Vizcaino. Collaboration tools for global software engineering. *Software, IEEE*, 27(2):52–55, March 2010.

[65] P. Layzell, O. Brereton, and A. French. Supporting collaboration in distributed software engineering teams. In *Software Engineering Conference, 2000. APSEC 2000. Proceedings. Seventh Asia-Pacific*, pages 38–45, 2000.

[66] C. P. Lee and D. Paine. From the matrix to a model of coordinated action (moca): A conceptual framework of and for cscw. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pages 179–194. ACM, 2015.

[67] P. Leinonen, S. Järvelä, and P. Häkkinen. Enhancing awareness of networked collaboration: A study of a global virtual team. *Computer Supported Collaborative Work Journal*, 14:301–322, 2005.

[68] K. Lewis. Knowledge and performance in knowledge-worker teams: A longitudinal study of transactive memory systems. *Management science*, 50(11):1519–1533, 2004.

[69] B. Lin, A. Zagalsky, M.-A. Storey, and A. Serebrenik. Why developers are slacking off: Understanding how software teams use slack. In *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion*, pages 333–336. ACM, 2016.

[70] S. A. Lynham. The general method of theory-building research in applied disciplines. *Advances in developing human resources*, 4(3):221–241, 2002.

[71] R. Martignoni. Global sourcing of software development-a review of tools and services. In *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, pages 303–308. IEEE, 2009.

[72] N. McDonald and S. Goggins. Performance and participation in open source software on github. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '13, pages 139–144, New York, NY, USA, 2013. ACM.

[73] M. McLure Wasko and S. Faraj. "it is what one does": Why people participate and help others in electronic communities of practice. *The J. of Strategic Information Systems*, 9(2):155–173, 2000.

[74] M. B. Miles, A. M. Huberman, and J. Saldana. *Qualitative data analysis: A methods sourcebook*. SAGE Publications, Incorporated, 2013.

[75] A. M. O'Donnell, C. E. Hmelo-Silver, and G. Erkens. *Collaborative learning, reasoning, and technology*. Routledge, 2013.

[76] F. F. Oliveira, J. C. Antunes, and R. S. Guizzardi. Towards a collaboration ontology. In *Proc. of the Snd Brazilian Workshop on Ontologies and Metamodels for Software and Data Engineering*, 2007.

[77] J. S. Olson and G. M. Olson. How to make distance work work. *interactions*, 21(2):28–35, 2014.

[78] N. E. Perry. Understanding classroom processes that support childreni£¡s self-regulation of learning. *In D. Whitebread, N. Mercer, C. Howe, & A. Tolmie (Series Eds.), British Journal of Educational Psychology Monograph Series II: Part 10. Self-regulation and dialogue in primary classrooms (pp. 45-68). Leicester, UK: British Psychological Society.*, 2013.

[79] N. E. Perry and A. Rahim. Studying self-regulated learning in classrooms. *Handbook of self-regulation of learning and performance*, pages 122–136, 2011.

[80] R. Pham, L. Singer, O. Liskin, F. Figueira Filho, and K. Schneider. Creating a shared understanding of testing culture on a social coding site. In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 112–121. IEEE, 2013.

[81] P. R. Pintrich. *The role of goal orientation in self-regulated learning.* Academic Press, 2000.

[82] PremierŠs Technology Council. A vision for 21st century education (special report), 2010.

[83] D. Redmiles, A. Van Der Hoek, B. Al-Ani, T. Hildenbrand, S. Quirk, A. Sarma, R. Filho, C. de Souza, and E. Trainer. Continuous coordination-a new paradigm to support globally distributed software development projects. *Wirtschafts Informatik*, 49(1):28, 2007.

[84] P. N. Robillard and M. P. Robillard. Types of collaborative work in software engineering. *Journal of Systems and Software*, 53(3):219–224, 2000.

[85] T. K. Rogat and L. Linnenbrink-Garcia. Socially shared regulation in collaborative groups: An analysis of the interplay between quality of social regulation and group processes. *Cognition and Instruction*, 29(4):375–415, 2011.

[86] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131–164, 2009.

[87] A. Sarma, A. Van Der Hoek, and L.-T. Cheng. A need-based collaboration classification framework. In *Proc. on Eclipse as Vehicle for CSCW Research, Workshop at CSCW 2004*, 2004.

[88] W. Scacchi. Understanding the requirements for developing open source software systems. In *Software, IEE Proceedings-*, volume 149, pages 24–39. IET, 2002.

[89] W. Scacchi, J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani. Understanding free/open source software development processes. *Software Process: Improvement and Practice*, 11(2):95–105, 2006.

[90] B. Schmitz and B. S. Wiese. New perspectives for the evaluation of training sessions in self-regulated learning: Time-series analyses of diary data. *Contemporary educational psychology*, 31(1):64–96, 2006.

[91] C. Seaman. Qualitative methods. In F. Shull, J. Singer, and D. SjÃÿberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 35–62. Springer London, 2008.

[92] D. I. Sjøberg, T. Dybå, B. C. Anda, and J. E. Hannay. Building theories in software engineering. In *Guide to advanced empirical software engineering*, pages 312–336. Springer, 2008.

[93] R. E. Slavin. *Cooperative learning: Theory, research, and practice.* Englewood Cliffs, NJ: Prentice-Hall,, 1990.

[94] A. Soller, A. Martínez, P. Jermann, and M. Muehlenbrock. From mirroring to guiding: A review of state of the art technology for supporting collaborative learning. *Int. J. Artif. Intell. Ed.*, 15(4):261–290, Dec. 2005.

[95] G. Stahl. Groupware goes to school: adapting bscw to the classroom. *International Journal of Computer Applications in Technology*, 19(3-4):162–174, 2004.

[96] R. E. Stake. *The art of case study research*. Sage, 1995.

[97] I. Steinmacher, A. P. Chaves, and M. A. Gerosa. Awareness support in distributed software development: A systematic review and mapping of the literature. *Computer Supported Cooperative Work (CSCW)*, 22(2-3):113–158, 2013.

[98] M.-A. Storey, L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky. The (r) evolution of social media in software engineering. In *Proceedings of the on Future of Software Engineering*, pages 100–116. ACM, 2014.

[99] M.-A. Storey, A. Zagalsky, F. Figueira Filho, L. Singer, and D. M. German. How social and communication channels shape and challenge a participatory culture in software development. *To appear in the Transactions on Software Engineering*, 2016.

[100] M. A. Storey, A. Zagalsky, F. Filho, L. Singer, and D. German. How social and communication channels shape and challenge a participatory culture in software development. *IEEE Transactions on Software Engineering*, PP(99):1–1, 2016.

[101] J.-W. Strijbos, R. L. Martens, W. M. Jochems, and N. J. Broers. The effect of functional roles on group efficiency using multilevel modeling and content analysis to investigate computer-supported collaboration in small groups. *Small Group Research*, 35(2):195–229, 2004.

[102] H. C. Stuart, L. Dabbish, S. Kiesler, P. Kinnaird, and R. Kang. Social transparency in networked information exchange: a theoretical framework. In *Pro-*

ceedings of the ACM 2012 conference on Computer Supported Cooperative Work, pages 451–460. ACM, 2012.

[103] R. Subramanyam, F. L. Weisstein, and M. S. Krishnan. User participation in software development projects. *Commun. ACM*, 53(3):137–141, Mar. 2010.

[104] D. Tesch, M. G. Sobol, G. Klein, and J. J. Jiang. User and developer common knowledge: Effect on the success of information system development projects. *International Journal of Project Management*, 27(7):657 – 664, 2009.

[105] G. Trevors, M. Duffy, and R. Azevedo. Note-taking within metatutor: interactions between an intelligent tutoring system and prior knowledge on note-taking and learning. *Educational Technology Research and Development*, 62(5):507–528, 2014.

[106] A. Van Der Hoek, D. Redmiles, P. Dourish, A. Sarma, R. Silva Filho, and C. De Souza. Continuous coordination: A new paradigm for collaborative software engineering tools. In *Workshop on Directions in Software Engineering Environments*, pages 29–36. Citeseer, 2004.

[107] Y. Verginadis, D. Apostolou, N. Papageorgiou, and G. Mentzas. An architecture for collaboration patterns in agile event-driven environments. In *2009 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, pages 227–230. IEEE, 2009.

[108] D. M. Wegner, T. Giuliano, and P. T. Hertel. Cognitive interdependence in close relationships. In *Compatible and incompatible relationships*, pages 253–276. Springer, 1985.

[109] A. Weinberger, B. Ertl, F. Fischer, and H. Mandl. Epistemic and social scripts in computer–supported collaborative learning. *Instructional Science*, 33(1):1–30, 2005.

[110] J. Whitehead. Collaboration in software engineering: A roadmap. In *2007 Future of Software Engineering*, FOSE '07, pages 214–225, Washington, DC, USA, 2007. IEEE Computer Society.

[111] P. Winne, A. Hadwin, and N. Perry. Metacognition and computer-supported collaborative learning. *International handbook of collaborative learning*, pages 462–479, 2013.

[112] P. H. Winne and A. F. Hadwin. Studying as self-regulated learning. *Metacognition in educational theory and practice*, 93:27–30, 1998.

[113] P. H. Winne and A. F. Hadwin. The weave of motivation and self-regulated learning. *Motivation and self-regulated learning: Theory, research, and applications*, pages 297–314, 2008.

[114] G. O. Wiredu. A framework for the analysis of coordination in global software development. In *Proceedings of the 2006 international workshop on Global software development for the practitioner*, pages 38–44. ACM, 2006.

[115] Y. Ye and K. Kishida. Toward an understanding of the motivation of open source software developers. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 419–429, May 2003.

[116] R. K. Yin. *Case study research: Design and methods*. Sage publications, 2013.

[117] A. Zagalsky, J. Feliciano, M.-A. Storey, Y. Zhao, and W. Wang. The emergence of github as a collaborative platform for education. In *Proceedings of*

the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing, pages 1906–1917. ACM, 2015.

[118] Z.-X. Zhang, P. S. Hempel, Y.-L. Han, and D. Tjosvold. Transactive memory system links work team characteristics and performance. *Journal of Applied Psychology*, 92(6):1722, 2007.

[119] B. Zimmerman. (2000). attaining self-regulation: A social cognitive perspective. *M. Bockacrts, PR Pintrich, 8c M. Zeidner (Eds), Handbook of self-regulation*, pages 13–39, 2000.

[120] B. Zimmerman and D. Schunk. *Self-regulated learning and academic achievement: theory, research, and practice*. Springer series in cognitive development. Springer-Verlag, 1989.

[121] B. J. Zimmerman. Investigating self-regulation and motivation: Historical background, methodological developments, and future prospects. *American Educational Research Journal*, 45(1):166–183, 2008.

[122] B. J. Zimmerman and D. Schunk. Motivational sources and outcomes of self-regulated learning and performance. *Handbook of self-regulation of learning and performance*, pages 49–64, 2011.

[123] B. J. Zimmerman and D. H. Schunk. *Self-regulated learning and academic achievement: Theoretical perspectives*. Routledge, 2001.

Appendix A

Neo4J case study - technical details of data collection

This appendix includes all code and queries used to collect the data from the Neo4J community. Note that all files used in the case study are available online[1].

A.1 Data Collection - Phase I

In this phase of the case study, Stack Overflow Q&A threads were collected using StackExchange [2]. The query to obtain the data is:

```
Select p.CreationDate, p.Id, p.ownerUserId, p.title, p.body,
c.CreationDate, c.text, p2.body, c2.text
from posts p, postTags pT, comments c, posts p2, comments c2
Where p.id=pT.PostId and
pT.TagId = 34480 and
p.id=c.PostId and
```

[1] https://github.com/maryi/Neo4J-CaseStudy
[2] http://data.stackexchange.com/stackoverflow/query/new

```
p.Id=p2.ParentId and
p2.id=c2.PostId
```

A.2 Data Collection - Phase II

In phase II, Stack Overflow Q&A threads were collected using StackExchange [3]. First, a query was executed to retrieve the post id from all posts created after March 25th 2015 with the tag `Neo4J`. The query used is:

```
Select p.Id
From posts p, postTags pT
Where p.id=pT.PostId and
pT.TagId = 34480 and
p.CreationDate >= '2015-07-01'
```

Results from the query were stored as a CSV file. Then, a Python script read the post id, use it to compose the url of the corresponding post, and downloaded the Stack Overflow Webpage as a PDF file with the format '<post_id>.pdf'. The Python script used is:

```
import os
# open list of issue ids
issue_ids = open("StackOverflowIds.csv")
# for each issue id in the file
for issue in issue_ids.read().splitlines():
```

[3] http://data.stackexchange.com/stackoverflow/query/new

```
url = 'http://stackoverflow.com/questions/' + issue;
os.system("wkhtmltopdf " + url + " " + issue + ".pdf")
```

GitHub issues were downloaded using the GHTorrent project[39]. In particular, the MySQL database on GhTorrent[4] was queried for the `issues_id` field in the table Issues with the same criteria as for downloading the posts from Stack Overflow. The query executed on the MySQL database on GHTorrent is:

```
SELECT issues.id
FROM issues
WHERE issues.repo_id='1366147' and
issues.created_at >='2015-03-25';
```

The results from the query were stored in a TXT file. Then a Python script was used to read each issue id from the file, use it to compose the url of the corresponding issue, and to download each GitHub issue as a PDF with the format '<issue_id>.pdf'. The Python script used to download the GitHub issues is:

```
import os
# open list of issue ids
issue_ids = open("issue_id.txt")
# for each issue id in the file
for issue in issue_ids.read().splitlines():
url = 'https://github.com/neo4j/neo4j/issues/' + issue;
os.system("wkhtmltopdf "+ url + " " + issue + ".pdf")
```

[4] `http://ghtorrent.org/dblite/`

To collect Google Groups discussions, a researcher manually filtered the data following the same criteria as for Github issues and downloaded each discussion as a PDF file.