Using the Model of Regulation to Understand Software Development Collaboration Practices and Tool Support

Maryi Arciniegas-Mendez, Alexey Zagalsky, Margaret-Anne Storey, Allyson F. Hadwin
University of Victoria
Victoria, BC, Canada
{maryia, alexeyza, mstorey, hadwin}@uvic.ca

ABSTRACT
We developed the Model of Regulation to provide a vocabulary for comparing and analyzing collaboration practices and tools in software engineering. This paper discusses the model's ability to capture how individuals self-regulate their own tasks and activities, how they regulate one another, and how they achieve a shared understanding of project goals and tasks. Using the model, we created an "action-oriented" instrument that individuals, teams, and organizations can use to reflect on how they regulate their work and on the various tools they use as part of regulation. We applied this instrument to two industrial software projects, interviewing one or two stakeholders from each project. The model allowed us to identify where certain processes and communication channels worked well, while recognizing friction points, communication breakdowns, and regulation gaps. We believe this model also shows potential for application in other domains.

Author Keywords
Collaboration; Theory; Regulation

ACM Classification Keywords
H.5.3 Group and Organization Interfaces: Computer-Supported Collaborative Work

INTRODUCTION
Software engineering involves the shared expertise and effort of multiple stakeholders. It is a highly collaborative activity that brings different people together to produce something better than any participant could conceive of or produce alone [24]. While software development emphasizes the creation of software assets, we 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 tools used in software development can significantly impact the success of a project and influence how developers learn and work together. For these reasons, practitioners and researchers have proposed the creation of or improvements to novel tools such as task trackers, configuration management systems, and lightweight messaging tools [26, 14]. Recent studies [42] 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 [8].

Numerous empirical studies have tried to understand developer work practices and uncover models, theories, and frameworks that can be used to describe how developers collaborate and to help improve their tools. However, these models do not adequately capture how development team members "regulate" themselves, one another, and their projects. We 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. Software engineering also 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.

In this work, we compose a model—the Model of Regulation—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 tasks. We use this model to uncover regulation work practices and understand why and how people use tools to support regulation. The Model of Regulation borrows constructs from the learning sciences and computer supported collaborative learning literature (CSCL) [17, 15, 30] and adapts them for the software engineering domain.

Our goal in this paper is to describe a *theoretical framework* of regulation for gaining *actionable insights* on the constraints and affordances of collaboration practices and tools. First, we discuss existing models of collaboration that have been applied to software engineering and motivate the need for the Model of Regulation. We describe the *modes of regulation* in collaborative development (self-, co-, and shared regulation), the *processes of regulation* (Task Understanding,

CSCW '17, February 25-March 01, 2017, Portland, OR, USA
 ISBN 978-1-4503-4335-0/17/03...$15.00
DOI: http://dx.doi.org/10.1145/2998181.2998360

Goal Setting, Enacting, Monitoring & Evaluating, Adapting), and present a *classification system for tools based on their support for regulation* (e.g., Awareness Tools and Guiding Systems). To describe the Model of Regulation, we use illustrative examples from modern collaborative software engineering projects and tools.

We also introduce a *regulation instrument* based on the Model of Regulation—a set of questions that an individual on a team-based software project can answer to help reflect on how they and their team members regulate their work and the various tools they use to support their work practices. Through this questionnaire, the Model of Regulation can be used by both researchers and practitioners as a lens for understanding collaboration practices in a project and identifying how well certain tools, processes, and communication channels work, as well as for recognizing friction points, communication breakdowns, and regulation gaps.

This paper showcases the regulation instrument by applying it to two industrial software projects: using the instrument questions as an interview guide, we interviewed stakeholders to learn more about their regulation practices and tools. We show how the instrument (and indirectly, the model) can be used to tell a "story" of the regulation processes in a project. We also discuss how the instrument helped our interviewees reflect on their regulation practices and tools and how they could improve them. Our findings lend support for the theoretical framework as a means for diagnosing, documenting, and guiding collaborative work practices and tool design for researchers and practitioners.

BACKGROUND

Understanding collaboration and how it can be improved with more effective processes and tools has been a longstanding challenge in software engineering practice and research. Indeed, there are many different frameworks and models of collaboration, each of which considers different contexts and viewpoints and plays an important role in describing collaboration processes. For example, MoCA [27] is a conceptual framework for characterizing the underlying context where collaboration takes place, however, this focus on understanding the context (such as size of the group) does not bring insights on how collaboration occurs nor how it can be improved.

In the following, we 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. We review models that are technology-centric, followed by a review of models that focus on characterizing the processes and participants involved. (Note that an extended review of these models is provided in a thesis [1].) We then propose how our Model of Regulation can complement the existing models and why it is an important contribution.

Technology-centric collaboration models

In the early days of CSCW, the emphasis was on designing tools needed to support collaboration rather than on understanding practices or developing theories [2]. Thus it is not surprising that the first models were focused on classifying systems and tool features. During this time, the term *groupware* was coined and frequently used to describe collaborative technologies [10].

One of the most widely used taxonomies to classify groupware—which was proposed by the CSCW research community—is Johansen's time-space matrix [23]. He suggested that all collaboration technologies can 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 to be incomplete and attempts were made to extend it by adding *predictability* [12] and additional space dimension states [6].

In the software engineering domain, Cook's 2004 work [4] reviewed existing collaboration tools and classified them based on their support for *design*, *development*, *management*, and *inspection*. In 2007, Whitehead [46] introduced a tool classification with four categories: *model-based collaboration tools* (for creating 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*.

Striving to understand collaboration through tools was a good initial step, however, collaboration is a complex activity that cannot be improved by just upgrading technologies. It is also difficult to productively collaborate when what collaboration consists of is still undefined. Moreover, as software development has become more distributed and dynamic, the community has experienced a proliferation of technologies—we now have such a vast array of computer-based tools that it is hard to list and even more challenging to classify them 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 approach. We describe these models next.

Towards characterizing collaboration processes and participants

As far back as 1991, Ellis *et al.* [10] 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 [10]. Later, Gerosa *et al.* [11] suggested that the 3C Model be amended to include *Awareness*: *"an understanding of the activities of others, which provides a context for your own*

activity." [7] Although the 3C Model has been used primarily to evaluate collaboration tools in software development (e.g., [46, 31, 39]), it represents one of the first approaches allowing for an analysis of collaboration processes.

Van der Hoek *et al.* [44] explored the notion of *Continuous Coordination* 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.

Collaboration has also been studied extensively in free and open source projects. Crowston *et al.* [5] investigated how to improve work team effectiveness through the application of Continuous Coordination principles and the **alignment** of task dependencies, team structures, resources, tool support, and actors. Additionally, Crowston *et al.* emphasized that **additional work** is sometimes required to deal with coordination issues. Likewise, Scacchi's [36] study of four OSS communities highlighted eight informal practices that are key for developing in OSS and showed how they are different from traditional software development. Other studies of FOSS projects have described development processes [37] as well as investigated conflict resolution and strategies for promoting collaboration in the GNU enterprise project [9] and Netbeans.org community [22]. These findings go beyond OSS as commercial projects and organizations have started adopting OSS-like practices and workflows [25]. Moreover, Kalliamvakou *et al.* found that industrial developers also use GitHub pull requests as critical *coordination points* between **individual** work and their **team** [25].

Olson and Olson [32] suggested that distributed teams should be explored from three different levels: the **individual**, the **manager**, and the **organization**. This distinction between individual and team and the importance of the individual's role in a collaborative effort was further emphasized by Watts Humphrey, a software engineer who developed the Personal Software Process (PSP) to provide structure to and increase quality in the work of an individual developer [18, 19]. PSP eventually became the foundation for the Team Software Process (TSP), an adaptation of the framework aimed to provide guidance for development teams while maintaining high productivity and quality [20]. One of the main strengths of this methodology is its applicability: it provides structured support for development activities by recommending methods and metrics that allow performance to be analyzed and quantified. However, this methodology focuses on the improvement of productivity and quality through quantitative methods, not on the analysis or understanding of collaboration.

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 [45]. TMS effectively helps one make sense of how teams manipulate and create knowledge [49], [28], 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). Continuous learning is facilitated by the way we regulate our past experiences and interpretation of the context, which allows us to see the results of our current actions and make strategic decisions to achieve our goals. Software engineering projects are informal learning environments and 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.

We 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. We 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. Next, we describe the Model of Regulation and its application to the software engineering domain.

THE MODEL OF REGULATION FOR SOFTWARE ENGINEERING

The Model of Regulation builds on the "theory of regulated learning" which contains concepts for understanding how learners strategically appropriate knowledge in their learning activities [47]. According to this theory, *regulation* is a key component of the learning process as it concerns how people make strategic decisions in the face of change. Changes can be either *external* (an activity has not provided the expected results) or *internal* (new ideas on how to approach a task have emerged), and they trigger an action that an individual must respond to strategically. Regulatory episodes occur in phases when people attempt to understand a task, set goals, perform strategic actions or engage in a task, and adapt their current and future work. These phases are described by Winnie *et al.* [47, 48] as **processes** that support regulated learning (such as Task Understanding and Goal Setting).

As people tend to learn together as well as contribute to common projects, the study of regulated learning was extended to collaborative settings by Hadwin *et al.*'s [15] investigation of the **modes of regulation** beyond self-regulation. While looking into these social forms of regulated learning (co- and shared), Hadwin and colleagues found that individuals lack the skills required to regulate themselves, which prevents a group's collaborative potential from being fulfilled [16, 21]. This work has led to an increased focus on harnessing technology and tools to guide and support individual and group regulated learning processes in order to help people be more productive and meet their goals. In particular, Hadwin and colleagues introduced a tool classification system that labels a **tool's support for regulation** [16, 21].

We introduce the Model of Regulation and describe how it combines the modes of regulation, regulation processes and regulation tool support, and how these components work together to *activate* regulation. Moreover, we propose that the Model of Regulation helps *document* the collaboration life cycle and can be used to compose *a prescription* for productive work. In particular, the model *provides a vocabulary* for the different processes and strategic decisions required in a collaborative activity, and includes *a sequential guide* that suggests the order in which these items should occur.

The Model of Regulation presented in this paper went through several iterations as we applied it to an earlier study that attempted to describe the collaboration experienced by the Neo4J open source development community[1]. During this two-phase case study, we analyzed Neo4J's three main communication channels—GitHub, Stack Overflow, and Google Groups—for traces of regulation over a four-month period. We do not discuss the Neo4J case study in this paper because its goal was to gain an understanding of how users regulate an open source development project, not to evaluate the model. Yet the case study helped expose the model's applicability to software engineering (and gaps) so we could adapt its processes. This paper presents the adapted model, while details on the case study are presented in a thesis [1].

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

In the following, we discuss the learning sciences constructs that led to the Model of Regulation and the adaptations that were made in order to cover all possible units of collaboration that one sees in modern development (e.g., software engineering teams, projects, organizations, and communities). Later, we discuss how these concepts have evolved from the learning sciences to the software engineering domain, showcasing examples to illustrate how the Model of Regulation can be applied in software engineering.

Before describing the components of the model, it is important to note the following points about regulation:

1. Regulation is multifaceted. It is not just about behaviors/actions, 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).

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

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

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

Modes of regulation

The Model of Regulation includes three modes of regulation: self-regulation, co-regulation, and shared regulation.

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 they face is to decide if they are going to understand the nature of the bug by consulting the logs or looking directly at the source code. In a collaborative setting, self-regulation is always required but is insufficient by itself. When collaborating, participants must make individual efforts towards a common goal—each person defines their own assessment of the problem at hand and the possible solutions. However, social processes are also required to achieve consensus and unified agreement, gradually refining each other's perceptions and leading to a shared understanding among members.

When exercised by participants, **co-regulation** refers to the recognition of each other's perspectives and the alignment of ideas regarding 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 by the participants' *interactions within the social context*. It should be noted that the term *"participant"* is used here as a reference to a member of the group regardless of their role or their location in the personnel hierarchy. The purpose of co-regulation is to provide temporary meta-cognitive support for planning, monitoring, and evaluating, where individual team members or entire groups support or influence the regulation processes for one or more members [33, 17]. For instance, discussions among developers can help detect individual misunderstandings regarding 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 a 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 defined by the executed 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, 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. The relationships between the three modes of regulation (self-, co-, and shared) are bidirectional, as depicted in Fig. 1. 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 [34].

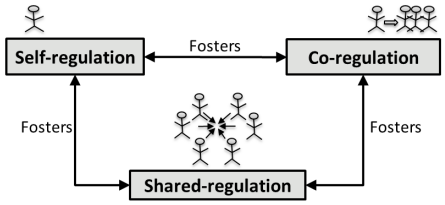

Figure 1. Modes of regulation in the Model of Regulation.

We note 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. Regulation can be applied to behavior (e.g., strategic actions, goals, plans), motivation (e.g., user motivation), cognition (e.g., task knowledge, self-knowledge, strategy knowledge), and emotion.

Interestingly, these three strategic activities—or modes of regulation—have already been identified to some degree as *"interaction levels"* by software engineering researchers [14]. 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 limited to 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 we know, no further attempt was made to understand the role of these interactions in collaborative software development.

Processes of regulation

As mentioned above, the Model of Regulation also includes processes related to the strategic decisions required to perform a task. In particular, regulation in software engineering consists of five interconnected processes (c.f. Fig. 2):

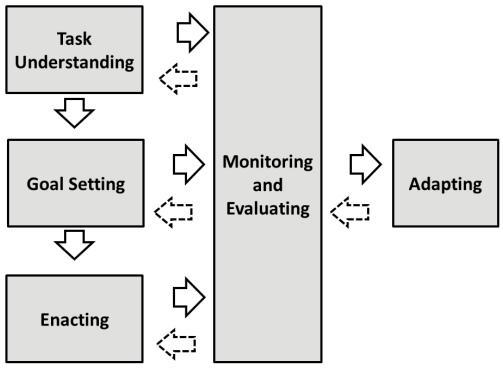

Figure 2. Processes of regulation in the Model of Regulation. Solid arrows represent outcomes going out of the process, while dashed arrows indicate feedback going into the process.

Task Understanding: As part of this process, individuals define their general perceptions of the task at hand and foresee the way participants will contribute to the completion of the task [35]. In software engineering, the Task Understanding process defines how participants invest time to think about the requirements, purpose, scope, social context and responsibilities, and roles 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 change the purpose of a project and user feature requests can modify initial requirements. Also, an external community or a company working on a similar product for the same market represent a direct competition that can alter the comprehension of one's project.

Goal Setting: As part of this process, individuals establish a work plan, including strategies to motivate and engage participants [43]. 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 accomplish tasks (e.g., definition of programming languages), acquire information (e.g., project wiki, discussions on virtual channels, books), and facilitate collaboration (e.g., selecting communication technologies, hosting services).

Enacting: With this process, group members follow the guidelines defined in the work plan. For example, developers follow code standards like adding 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 systems). For some projects, we see that some or even all of these project parameters are prescribed in their work plans.

Monitoring and Evaluating: As part of this process, group members monitor performance and preliminary outcomes and compare them against goals to detect misalignments. Monitoring and Evaluating is an important activity in software engineering and a key to success in many cases. Because of software development's dynamic nature, 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—sources of misalignment between the expected and actual outcomes are identified, revised, and adapted. For example, a particular programming language may be selected for a project because it offers strong support for creating the kinds of artifacts that are relevant to the project (e.g., in the case of Neo4J, programming support for graphs was important). However, when the 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 they need more training with the selected language. In either case, they *adapt* their goals or tasks accordingly. It should be noted that the Adapting process is often not sequential and can occur over any of the other regulation processes as required.

Activating the regulation processes

We have described the different processes that occur in regulation, but we have not covered how one process leads to another during collaborative work. Next, we discuss how the regulation processes activate each other.

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 initiated with the preliminary results from 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 results from Task Understanding. If outcomes are not aligned in any comparison, for example, if the plan is not aligned with the understanding of a project or the preliminary outcomes from enacting a task are not aligned with the plan, the Monitoring and Evaluating process triggers a strategic decision through the *Adapting* process as this process provides input for any processes that require revision so an update can be performed. For example, a software project's customers requesting new features or changing their minds about existing features is more often a rule than an exception. These frequent changes in project requirements lead to changes in the project's Task Understanding, Goal Setting, and Enacting processes.

Now if a task is to be performed by an individual, then regulation processes (e.g., Task Understanding, Goal Setting) only occur at the individual level: self-regulation. However, if a task represents a collaborative activity to be performed by multiple people, then the regulatory processes start at the individual level (self-regulation), expand to include participant interactions (co-regulation), and finally are executed within the entire unit of collaboration (shared regulation). For example, developers approach a collaborative software development project by individually trying to understand the project's requirements and how they will contribute to the final product (self-regulation of Task Understanding). Eventually, formal and informal communication that occurs between developers will help them refine their individual understanding of the project (co-regulation of Task Understanding), which will allow the group members to achieve a shared understanding of the project (shared-regulation of Task Understanding). That is, each cycle of regulatory process occurs within the three modes of regulation. Figure 3 illustrates how the regulation processes are activated.

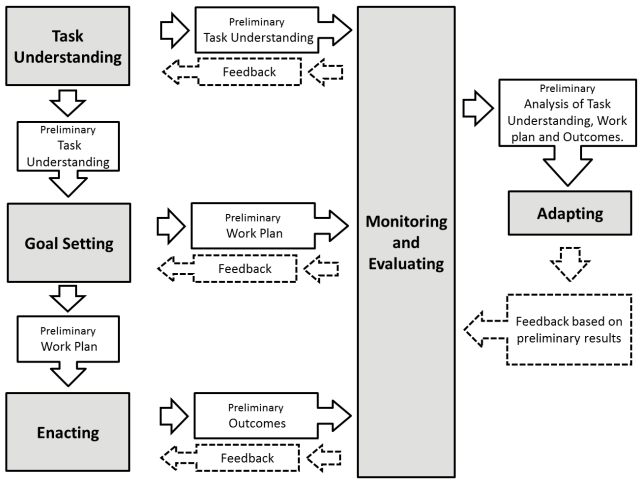

Figure 3. Activation of the regulation processes in the Model of Regulation. Solid arrows represent outcomes of the process, while dashed arrows indicate feedback going into the process.

Regulation tool support

Our Model of Regulation further articulates how computer-based tools can be used to support self-, co-, and shared regulation. Next, we describe the different categories of regulation tool support. Their contribution 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 they are created as self-reminders, they can be used 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.

Structuring Support refers to approaches that aim to guide interactions by designing or scripting a situation before it begins [38]. Structuring support consists of roles, scripts, and prompts. **Roles** are functions intentionally assigned to each member of the team (e.g., designer, developer, or tester), while **scripts** are a list 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 a list of activities that need to be performed 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. We showcase a few examples of how structuring support could be provided in a development team's toolset in Fig. 4.

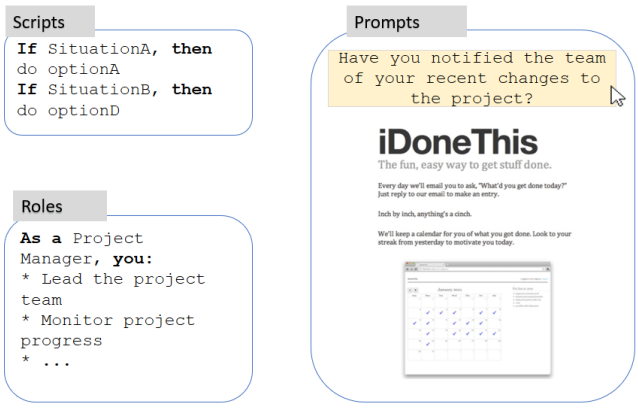

Figure 4. Examples of structuring support. Scripts (top left) refer to a list of steps shown in the format of If-Then rules. Roles (bottom left) define functions and responsibilities of a member with respect to a task. iDoneThis emails (shown on the right) give an example of prompts (messages) that provide hints about correct processes for a particular task.

Mirroring Support refers to mechanisms that reflect individual or collective actions by gathering and summarizing data [38]. An example of mirroring support used for self-regulation shows individual actions as a summary of activities, with completed tasks distinguished by color or some other cue. When used to support social modes of regulation, mirroring support can be achieved with a visualization such as a pie chart or a timeline graph that shows aggregated contributions. In this case, the visualization supports a shared understanding of the group perspective and creates context for shared regulation. Several software development environments provide views to highlight such information. For example, GitHub's profile page shows the frequency of individual contributions using color cues in a timeline matrix, Trello shows advances on a plan using a progress bar, WakaTime shows statistics of activities using graphs, and Codealike uses visualizations to analyze time invested in development-related activities (e.g., coding, debugging). The mirroring features of these tools are illustrated in Fig. 5.

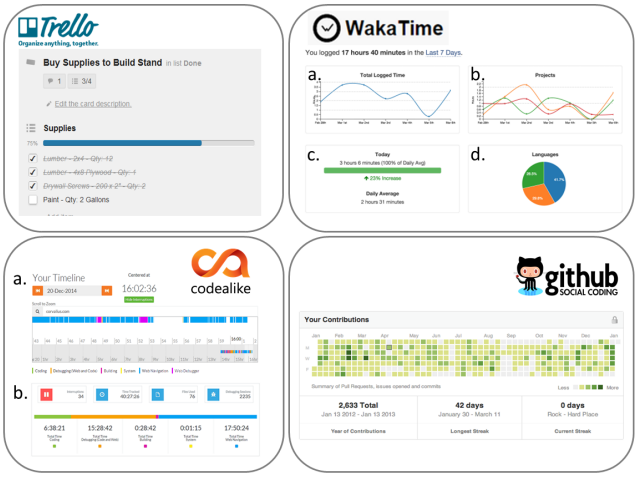

Figure 5. Examples of mirroring support tools that reflect individual or collective actions by summarizing data: Trello's progress bar (top left); WakaTime graphs (top right) show a) total logged time, b) time dedicated to different projects, c) today's logged time, and d) distribution of programming languages; Codealike visualizations (bottom left) show a) distribution of activities and b) statistics and total time per activity; and GitHub's frequency of contributions timeline matrix (bottom right).

Awareness Tools is a similar category to mirroring support except that it allows for self-comparison or comparisons between group members. For instance, an awareness tool that invites co-regulation will be restricted to comparisons between a small set of members, while 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. 6), or a visualization showing the lines of code introduced by each member.

Guiding Systems are programs that behave like a virtual presence, interpreting data and providing instructions when issues in the collaboration process are detected. For instance, a guiding system that sends reminders of an individual's pending work helps with self-regulatory processes. A guiding system that shows visualizations of ideas not picked up during previous group discussions supports 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 [41] demonstrates the potential for bots in guiding collaboration.

Next, we show how this model can be used to gain further understanding of today's collaborative practices and tools.

THE MODEL OF REGULATION IN ACTION

Processes of regulation (e.g., Enacting, Adapting) describe the set of strategic decisions that must be made while per-

Figure 6. An example of awareness support provided by GitHub. Awareness support refers to mirroring mechanisms that also allow for self-comparisons or comparisons between group members. Here, individual contributions to a GitHub repository are plotted on a timeline per member and presented in a two-column format.

forming a task, while the modes of regulation (self-, co-, and shared regulation) indicate the scope of each regulatory cycle. In this part of the paper, we show one way of applying the Model of Regulation by transforming it into an action-oriented instrument for practitioners and researchers.

Creating the instrument

To demonstrate how the theoretical framework can be used, we developed an instrument—a questionnaire based on the Model of Regulation—that aims to assist in investigating the collaboration practices of individuals and groups. The goal of this instrument is to provide a contextually sensitive diagnosis of the regulatory strengths in a collaboration activity and reveal potential areas for improvement in terms of working processes, communication channels, and tool use.

The questionnaire was created and refined over a series of iterations. First, we used the Model of Regulation to define a set of questions that referred to all regulation modes and processes. Then, two domain experts (one from the learning sciences and one from software engineering) ensured the questionnaire fit the model and validated that the vocabulary was clear and appropriate for software engineers. Finally, two pilot interviews led to the rewording of one of the questions (this version of the instrument is shown in Table 1). In bold, we highlight the processes of regulation followed by a short definition. In subsequent rows, we identify the mode and scope of regulation: self-regulation refers to *individual* processes; co-regulation describes how individual strategic responses are modeled by the participants' *interactions* within their social context; and shared regulation refers to the joint processes experienced by the *group*. Questions shown in normal font are designed to ask about the existence of the activity, while questions in italic font inquire about the documentation and tool support they create or use, respectively.

Using the instrument to reveal work practices and tools

After creating the instrument, we conducted interviews with three developers currently engaged in collaborative software development—these are not the same as the pilot interviews used to create the instrument. First, we 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 large technology provider 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, we interviewed the CEO and the CTO of a start-up software company (referred to as P2 and P3, respectively)—both were interviewed together. Their company consisted of sixteen people who work co-located on the design and implementation of referral program software.

Using the instrument as a guide, the interviews allowed us to construct a rich profile of collaboration practices for each case. We present the resulting profiles in the form of a table, one for each regulation process. We summarize the findings and observations for both cases side by side (P1's interview in the left column of each table, and P2 and P3's interview in the right column of each table). For the sake of brevity, we describe our observations for only two of the processes: Task Understanding and Monitoring & Evaluating (see Tables 2 and 3), while the others are included as an appendix to this paper. Further discussion is presented in a thesis [1].

Reflection on the Task Understanding process

The Task Understanding process refers to the way participants and teams approach a task and how they achieve consensus regarding what needs to be done. Through the application of the instrument, we revealed that geographic distances [3] define how Task Understanding is experienced by individuals and teams, as illustrated in Table 2.

Interviewees of both projects recognize the critical value of being aware and having a shared understanding of the tasks to be executed. However, their methods and strategies to achieve and ensure team consensus differ due to their physical settings. P1's team does not have many opportunities for informal communication so they have to rely on tool support, which is reflected in the large number of tools and variety of features they use. They expend extensive effort selecting technologies, coordinating the sharing of knowledge, and gaining and maintaining awareness of projects. For this team, the questionnaire helped elicit a friction point that concerns the lack of commitment from some members to make that extra effort to communicate and share knowledge.

P2 and P3's team is co-located and their alignment of ideas with respect to tasks naturally occurs in their daily communication (formal or informal). In this case, tools are used to persist information and to keep a record of the agreements made, rather than just for supporting team interactions (as in the case of P1).

Reflection on the Monitoring and Evaluating process

In software engineering projects, participants constantly monitor and evaluate their progress against work plans and expected results, as well as against changes in their project

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 the alignment of ideas? *Where are those discussions documented?*
Shared regulation: Group	Does the group hold discussions and reach agreement about their project understanding? *Is the result of group discussions documented and available to all members? Where is it?*
Goal Setting	**Establishing 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 help other team members 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	**Following the plan while providing support for motivational engagement.**
	Plan execution: Following the plan
Self-regulation: Individual	Do you follow your personal work plan?
Co-regulation: Interactions	Do you support other team members in executing 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: Implementing strategies for staying motivated and engaged
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 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.**
	Checking progress 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 understanding and work plan? *Where are the results of your evaluation documented?*
Co-regulation: Interactions	Do you help other team members monitor and evaluate whether the outcome of their work is aligned with their project understanding and work plan? *Is the result of these conversations documented? Where?*
Shared regulation: Group	As a group, do you hold discussions to monitor and evaluate whether the outcome of the work is aligned with the group's project understanding and work plan? *Where is the outcome of discussions documented and available for all members?*
	Checking the project understanding and work plan are up to date. (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 work plan? *Where did you document the changes detected (if any)?*
Co-regulation: Interactions	Do you help other team members monitor and evaluate changes in their project understanding and work plan? *Where is the result of this evaluation documented?*
Shared regulation: Group	Does the group hold discussions to monitor and evaluate changes in the group's project understanding and 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 understanding changed? *Where did you document the changes to the original plan and the reasons behind these changes?*
Co-regulation: Interactions	Do you help other team members adapt their work plans 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's project understanding changed? *Where did you document the modifications to the original plan and the reasons behind these changes?*

Table 1. Instrument to profile collaboration: a questionnaire based on the Model of Regulation that helps elicit collaboration practices and tool support from individuals and teams.

understanding. Checking progress is different from, for example, checking if the project requirements have changed. However, in the case of P1, their requirements change frequently (often within a couple of hours) and plans are constantly modified, meaning no plan is ever complete or stable. As a result, P1 and their team continually monitor and evaluate both the *expected results* and the *changes to task understanding* at the same time. This can be seen in the left column of Table 3. In this situation, the instrument helped reveal a friction point that arose from the abrupt and quick changes to plans that cause team members to put too much effort into monitoring progress and in keeping up with changes.

In the case of P2 and P3, the team also experiences modifications to their work plans, customers change requirements, and approaches they use do not always work out as expected. However, these changes are often distributed over time which makes it easier for the team to clearly distinguish between monitoring and evaluating their progress against the planned results with monitoring and evaluating changes to their task understanding.

Interestingly, all reflections on the processes of regulation for both projects share the following pattern. For P1, the model (through the instrument) was a mechanism to frame their collaboration friction points. While for P2 and P3, the model allowed them to describe their practices and provided a theoretical foundation for their existing activities.

Reflection on the Model of Regulation

The application of the model allowed us to have a rich discussion about affordances and constraints of team collaboration

	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 difficult to engage with each other. Their virtual environment presents extra challenges when trying to understand and reflect on tasks.** *"There is no way you can just pop over the cubicle and say 'hey, what are you working on?' "* [P1]	**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 a 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 start-up 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 *"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.	Collective understanding is achieved in group meetings. The complexity of the project determines the length and 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] Tools: Slack is the main communication tool and the source of raw documentation. Agreements are documented on Google Docs where they usually have a document that captures 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]

Table 2. 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.

practices and tool support, which we share through the examples presented in this section (see the Appendix and Thesis [1] for more details). 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."* [P2]

P3 recognized that using the instrument to reflect on their work practices and tool choices does not add new steps to their daily work. It instead provides a mechanism to formalize and structure how they discuss their existing practices as the concepts of the model were *"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."* 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, P1 commented in a follow-up discussion shortly after their interview that they had started thinking about their collaboration issues in terms of the model. P1 used the vocabulary of the model learned from the interview to describe the reasons for the friction and breakdowns they experienced with their team. For example, P1 mentioned that a coworker *"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 the coworker *"obviously had some sort of disconnect with task understanding and it wound up costing a lot of time / resources."*

DISCUSSION

In this section, we first discuss how the concepts we combined into the Model of Regulation are interpreted and applied differently in a software engineering context compared to a learning context. We also speculate how the model could be applied to other knowledge worker domains. Next, we look beyond the team to see how the model can be applied at the organization or community level, areas where we feel there is great potential.

As presented earlier, current collaboration models that have been applied to software engineering emphasize only a particular subset of collaboration concerns. Our 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. We finally discuss how this more comprehensive model provides the insights and vocabulary needed to capture the theoretical underpinnings of why certain tools and practices are used.

From learning to software engineering and beyond

We borrowed and adapted regulation constructs from the learning science domain and applied them to the software engineering domain. While learning is crucial for facilitating the emergence of regulation in both domains, there are important differences in the way regulation is experienced (cf. Winne and Hadwin [48]). In software engineering, tasks tend to be more open-ended and evolve constantly. The complex

	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.** *"It's weird because no one knows anything, not even what the project is going to look like in a month."* [P1]	**Team awareness is facilitated by computer-based tools, which play a major role in collaboration.** *"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]
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 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.	For P2 and P3, individual regulation is accomplished by monitoring the progress of others, which is perceived as a natural task that takes place in every open conversation and meeting: *"It's an implicit and unspoken process."* [P2]
Co-regulation	With keywords configured in Slack, P1 has created *"a newsfeed for every time any of those things get mentioned"*, which 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.	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]
Shared regulation	The size of a project can make it challenging to monitor, so P1 and the other product designers on the team strategically offload 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]	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.
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 mode in 'Monitoring and Evaluating - Against expected results and work plan'.	P2 and P3 continuously monitor changes in their task understanding. *"Something comes up, and you react, and you keep reacting."* [P2]
Co-regulation	Same as the co-regulation mode in 'Monitoring and Evaluating - Against expected results and work plan'.	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] In addition, *"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]
Shared regulation	Same as the shared-regulation mode in 'Monitoring and Evaluating - Against expected results and work plan'.	Any new agreements are shared with the team and documented.

Table 3. Monitoring & Evaluating, 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. See Appendix for a summary of all processes.

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. Also, to study regulation processes (e.g., Task Understanding, Enacting) in software engineering contexts, the typical characteristics of the domain artifacts manipulated in each process needed to be identified. While the Task Understanding process in formal learning environments 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 us to change the definitions of the modes and processes of regulation and to consider many different types of tools that support regulation in the software engineering domain (such as configuration management systems).

While the processes of regulation describe the steps one goes through when facing a challenge, the modes (self-, co-, and shared) allow us to reveal the different ways regulation is experienced and expose how these relate to and shape group interactions. In the co-located group studied, the modes of regulation allowed us to describe how, building on individual regulatory activities (self-regulation), team members refined their approaches through formal and informal interactions (co-regulation) to finally arrive at a form of group regulation (shared regulation). While with the distributed team, the modes of regulation allowed us to discover how the lack of individual regulatory actions on behalf of some members affected the group's overall performance.

These constructs—modes, processes, tool support—have been discussed in the learning science literature, but this pa-

per combines them into a single model, adapts them to fit the software engineering domain, and explicitly discusses how the different processes activate other processes. We'd like to stress that the regulation processes and their connections to each other are complex. It is important to understand not only the pieces involved but also how they work together—this is achieved by the modes of regulation. We also used the model to derive an "action-oriented" instrument for eliciting how regulation occurs in software development teams. Our colleagues in the learning sciences have indicated to us that a similar instrument could prove to be very useful in the learning domain. Thus, composing the model and adapting it to software engineering has brought potential benefits to the learning domain as well.

Moving forward, we believe the model could be adapted for other knowledge work domains. Indeed, software developers are often described as the "prototype of the future knowledge worker"[2] as they tend to innovate and adopt tools before other knowledge workers do (e.g., email, Wikipedia, and GitHub). We suggest that adapting and applying the model and instrument to other domains would bring other important insights.

Beyond the team: Understanding collaboration in organizations and communities

We propose 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 2,700 contributors[3]). In this paper, we have discussed how the model can be applied to small industrial teams. However, as part of our larger research project, we have applied the Model of Regulation to the Neo4J open source development community and explored how stakeholders from a wider community can "regulate" development activities. For example, users of the Neo4J graphing technology (through GitHub and their Google Groups channels) give input to and request features from core Neo4J developers [1].

Through our industrial case studies, the vocabulary provided by the model allowed us and the practitioners we 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 us to reflect on collaboration beyond the individual and team levels and understand where practices and tools could be improved. One of our interviewees mentioned they found many benefits from using this vocabulary and that they continued to reflect on their practices and tool use some time after their interview with us.

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

The model we present in this paper is based on the concept of regulation, which considers collaboration at a metacognitive level and explores behaviour, cognition, and motivation. Our Model of Regulation can be used as a descriptive model because it allows one to describe how regulation occurs within a collaboration unit and how tools support regulation. However, we believe that the 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.* [13]).

The processes of regulation provide a high-level guideline for tasks and suggest how tools can help support regulation. We propose that the model's tool support component helps connect the theory with tool design principles, which is something we wish to explore in future work. Furthermore, we believe the instrument drawn from the model can be used to guide tools evaluations and comparisons of collaborative practices within or across a single or multiple units of collaboration.

While the theory is an important product of our research, we are 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 us 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 [29]).

Using our model, 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.

LIMITATIONS

We recognize that our Model of Regulation may need further refinements when applied to future software engineering 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 include concepts that 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 [3]). However, the iterative refinement of the model presented in this paper was effective at helping us understand the benefits of the many different tools developers use (as discussed earlier in the paper) and the regulation activities of users in an open source community. Although we lack space to discuss the open source community case study we performed, we refer the interested reader to a related thesis for more details and insights about how the model was adapted and applied [1].

The usefulness of the model was validated in part by applying the derived instrument through the case studies we discussed in this paper. The instrument was able to bring insights on how regulation activities occurred across these two contrast-

[2] http://allankelly.blogspot.ca/2014/04/theprototype-of-future-knowledge.html

[3] https://github.com/rails/rails

ing projects (one co-located setting and one distributed setting). However, we note that two case studies means we cannot generalize the application of the model. However, we strived to offset this limitation by checking the completeness of the instrument with two domain experts as we developed it, as well as piloting the instrument with two other developers before conducting our two case studies. As we conduct future interviews, we expect the instrument to go through at least some minor refinements in the questions it asks, which may in turn lead to changes in the model itself.

Through our research to date, we recognize the model does not have mechanisms to conduct a quantitative analysis of productivity, but perhaps integrating methods from the PSP and TSP approaches [19, 20] can help overcome this limitation. Using the Model of Regulation may help reveal variables of interest (e.g., how frequently tools are used to regulate activities) that could be measured quantitatively.

Finally, this paper has shown how the instrument (and in turn, the model) can be used as a descriptive theory. Our future work will aim to investigate whether the model can be used to provide prescriptive recommendations and to then validate those in other studies.

CONCLUSIONS

Collaboration has become an integral aspect of software engineering. The widespread availability and adoption of social channels has led to a participatory culture where developers frequently collaborate with and learn from one another [42]. While collaboration in software engineering has been studied extensively, gaining an understanding of collaboration in today's participatory development culture is challenging as current models and frameworks were developed before the community embraced socially enabled channels [40]. Moreover, many of today's development contexts occur beyond the confines of a team or organization, and how the individual, team, and community regulate development and learning activities is critical to a project's success.

This paper described how we adopted and adapted concepts from the theory of regulated learning from the learning sciences to form a Model of Regulation. We demonstrated how the model, through a questionnaire instrument we developed, can be used to gain insights into the *work practices*, *tool needs*, and *friction points* of two contrasting development teams. In comparison to other collaboration models applied in a software development context, our model is potentially highly prescriptive as well as descriptive. We discussed how the model could lead to actionable outcomes for a software development team or project.

Developers now use a complex constellation of tools to support their work [42]—not just development tools, but also a wide array of productivity, communication, and social tools—and understanding and articulating why these tools are used is challenging. We propose that the Model of Regulation provides a theoretical basis for evaluating and comparing the affordances and friction points of the tools used and can bring insights into collaborative work practices. Understanding how individuals and teams of developers can be more productive is something companies and researchers care deeply about. We anticipate that the Model of Regulation can be used as a canvas for understanding productivity gains and challenges.

Finally, we believe this model can be applied to knowledge work domains outside of the learning sciences and software engineering. We believe that the Model of Regulation plays a complementary role to other models and frameworks offered by the CSCW community and will bring insights into how knowledge workers regulate their work and learning activities and how they collaborate with one another.

ACKNOWLEDGMENTS

The authors would like to thank Cassandra Petrachenko for her editing support and insightful comments that contributed to this work, as well as the developers that provided feedback on our instrument and participated in our interviews. We're especially thankful to Christoph Treude for his valuable advice and insightful feedback. Also, we thank the CSCW reviewers whose constructive comments greatly helped to improve this paper.

REFERENCES

1. Maryi Arciniegas-Mendez. 2016. Regulation in Software Engineering. *M.Sc. thesis* (2016).
2. Liam J Bannon and Kjeld Schmidt. 1989. CSCW-four characters in search of a context. *DAIMI Report Series* 18, 289 (1989).
3. Elizabeth Bjarnason, Kari Smolander, Emelie Engström, and Per Runeson. 2016. A Theory of Distances in Software Engineering. *Inf. Softw. Technol.* 70, C (Feb. 2016), 204–219. DOI: **http://dx.doi.org/10.1016/j.infsof.2015.05.004**
4. Carl Cook. 2004. *Collaborative software engineering: An annotated bibliography*. Technical Report. Department of Computer Science and Software Engineering, University of Canterbury.
5. Kevin Crowston, Hala Annabi, James Howison, and Chengetai Masango. 2004. 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)*. 18–26.
6. Alan Dix, Janet E. Finlay, Gregory D. Abowd, and Russell Beale. 2003. *Human-Computer Interaction (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
7. Paul Dourish and Victoria Bellotti. 1992. Awareness and coordination in shared workspaces. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*. ACM, 107–114.
8. Kevin Dullemond, Ben van Gameren, and Rini van Solingen. 2012. 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. 398–405.

9. Margaret S. Elliott and Walt Scacchi. 2003. 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)*. ACM, New York, NY, USA, 21–30. DOI: **http://dx.doi.org/10.1145/958160.958164**
10. Clarence A Ellis, Simon J Gibbs, and Gail Rein. 1991. Groupware: some issues and experiences. *Commun. ACM* 34, 1 (1991), 39–58.
11. Marco Aurélio Gerosa. 2003. 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 (2003), 315–332.
12. Jonathan Grudin. 1994. Computer-supported cooperative work: History and focus. *Computer* 5 (1994), 19–26.
13. Carl Gutwin, Reagan Penner, and Kevin Schneider. 2004. Group awareness in distributed software development. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*. ACM, 72–81.
14. Anja Guzzi, Alberto Bacchelli, Yann Riche, and Arie van Deursen. 2015. Supporting Developers' Coordination in the IDE. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*. ACM, 518–532.
15. Allyson Fiona Hadwin, Sanna Järvelä, and Mariel Miller. 2011. Self-regulated, co-regulated, and socially shared regulation of learning. *Handbook of self-regulation of learning and performance* 30 (2011), 65–84.
16. Allyson Fiona Hadwin, Mariel Miller, and Elizabeth Webster. 2013. 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).
17. Allyson Fiona Hadwin, Mika Oshige, Carmen LZ Gress, and Philip H Winne. 2010. Innovative ways for using gStudy to orchestrate and research social aspects of self-regulated learning. *Computers in Human Behavior* 26, 5 (2010), 794–805.
18. Watts S Humphrey. 1995. *A discipline for software engineering*. Addison-Wesley Longman Publishing Co., Inc.
19. Watts S Humphrey. 1996. *Introduction to the personal software process (sm)*. Addison-Wesley Professional.
20. Watts S Humphrey. 2000. *Team Software Process (TSP)*. Wiley Online Library.
21. Sanna Järvelä and Allyson Fiona Hadwin. 2013. New Frontiers: Regulating Learning in CSCL. *Educational Psychologist* 48, 1 (2013), 25–39.
22. Chris Jensen and Walt Scacchi. 2005. 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*. IEEE, 196b–196b.
23. Robert Johansen. 1988. *Groupware: Computer support for business teams*. The Free Press.
24. David W Johnson and Roger T Johnson. 1989. *Cooperation and competition: Theory and research.* Interaction Book Company.
25. Eirini Kalliamvakou, Daniela Damian, Kelly Blincoe, Leif Singer, and Daniel M German. 2015. Open source-style collaborative development practices in commercial projects using github. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press, 574–585.
26. Filippo Lanubile, Christof Ebert, Rafael Prikladnicki, and Aurora Vizcaino. 2010. Collaboration Tools for Global Software Engineering. *Software, IEEE* 27, 2 (March 2010), 52–55.
27. Charlotte P Lee and Drew Paine. 2015. 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*. ACM, 179–194.
28. Kyle Lewis. 2004. Knowledge and performance in knowledge-worker teams: A longitudinal study of transactive memory systems. *Management science* 50, 11 (2004), 1519–1533.
29. Bin Lin, Alexey Zagalsky, Margaret-Anne Storey, and Alexander Serebrenik. 2016. 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*. ACM, 333–336.
30. Mariel Miller and Allyson Fiona Hadwin. 2015. Scripting and awareness tools for regulating collaborative learning: Changing the landscape of support in CSCL. *Computers in Human Behavior* 52 (2015), 573–588.
31. Felipe F Oliveira, Julio CP Antunes, and Renata SS Guizzardi. 2007. Towards a collaboration ontology. In *Proc. of the Snd Brazilian Workshop on Ontologies and Metamodels for Software and Data Engineering*.
32. Judith S Olson and Gary M Olson. 2014. How to make distance work work. *interactions* 21, 2 (2014), 28–35.

33. Nancy Perry. 2013. Understanding classroom processes that support children'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).
34. Nancy E Perry and Ahmed Rahim. 2011. Studying self-regulated learning in classrooms. *Handbook of self-regulation of learning and performance* (2011), 122–136.
35. Toni Kempler Rogat and Lisa Linnenbrink-Garcia. 2011. Socially shared regulation in collaborative groups: An analysis of the interplay between quality of social regulation and group processes. *Cognition and Instruction* 29, 4 (2011), 375–415.
36. Walt Scacchi. 2002. Understanding the requirements for developing open source software systems. In *Software, IEE Proceedings-*, Vol. 149. IET, 24–39.
37. Walt Scacchi, Joseph Feller, Brian Fitzgerald, Scott Hissam, and Karim Lakhani. 2006. Understanding free/open source software development processes. *Software Process: Improvement and Practice* 11, 2 (2006), 95–105.
38. Amy Soller, Alejandra Martínez, Patrick Jermann, and Martin Muehlenbrock. 2005. From Mirroring to Guiding: A Review of State of the Art Technology for Supporting Collaborative Learning. *Int. J. Artif. Intell. Ed.* 15, 4 (Dec. 2005), 261–290. **http://dl.acm.org/citation.cfm?id=1434935.1434937**
39. Igor Steinmacher, Ana Paula Chaves, and Marco Aurélio Gerosa. 2013. Awareness support in distributed software development: A systematic review and mapping of the literature. *Computer Supported Cooperative Work (CSCW)* 22, 2-3 (2013), 113–158.
40. Margaret-Anne Storey, Leif Singer, Brendan Cleary, Fernando Figueira Filho, and Alexey Zagalsky. 2014. The (R) Evolution of social media in software engineering. In *Proceedings of the on Future of Software Engineering*. ACM, 100–116.
41. Margaret-Anne Storey and Alexey Zagalsky. 2016. Disrupting Developer Productivity One Bot at a Time. In *To appear at the 2016 International Symposium on the Foundations of Software Engineering*. ACM.
42. Margaret-Anne Storey, Alexey Zagalsky, Fernando Figueira Filho, Leif Singer, and Daniel M. German. 2016. How Social and Communication Channels Shape and Challenge a Participatory Culture in Software Development. *To appear in the Transactions on Software Engineering* (2016).
43. Gregory Trevors, Melissa Duffy, and Roger Azevedo. 2014. 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 (2014), 507–528. DOI: **http://dx.doi.org/10.1007/s11423-014-9343-8**
44. André Van Der Hoek, David Redmiles, Paul Dourish, Anita Sarma, Roberto Silva Filho, and Cleidson De Souza. 2004. Continuous coordination: A new paradigm for collaborative software engineering tools. In *Workshop on Directions in Software Engineering Environments*. Citeseer, 29–36.
45. Daniel M Wegner, Toni Giuliano, and Paula T Hertel. 1985. Cognitive interdependence in close relationships. In *Compatible and incompatible relationships*. Springer, 253–276.
46. Jim Whitehead. 2007. Collaboration in Software Engineering: A Roadmap. In *2007 Future of Software Engineering (FOSE '07)*. IEEE Computer Society, Washington, DC, USA, 214–225. DOI: **http://dx.doi.org/10.1109/FOSE.2007.4**
47. Philip H Winne and Allyson F Hadwin. 1998. Studying as self-regulated learning. *Metacognition in educational theory and practice* 93 (1998), 27–30.
48. Philip H Winne and Allyson F Hadwin. 2008. The weave of motivation and self-regulated learning. *Motivation and self-regulated learning: Theory, research, and applications* (2008), 297–314.
49. Zhi-Xue Zhang, Paul S Hempel, Yu-Lan Han, and Dean Tjosvold. 2007. Transactive memory system links work team characteristics and performance. *Journal of Applied Psychology* 92, 6 (2007), 1722.

Appendix

Reflection on the Goal Setting process

Continuous changes to the project prevents P1's team from creating long-term plans. Therefore, their Goal Setting process focuses on immediate tasks. In the case of P2 and P3, their Goal Setting process is defined at both high and low levels, and their working culture defines how to approach each aspect of their planning. Details about the interviewees' experiences for this process of regulation are presented in Table 4.

Interestingly, interviewees declared that their current working culture slowly emerged over time, and today, the shared agreements (e.g., quality standards, regular procedures) followed by all members make the work easier.

	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 greater focus on product quality rather than making a deadline.** *"If it takes an extra day, but you will feel happier with the quality and this is acceptable for the customers, go for it."* [P3]
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 *"at the higher levels, responsibilities are weighed 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]
Co-regulation	Using Slack, P1 checks on everyone's planning activities and has 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. 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 in which each participant discusses the tasks they completed the previous day and describes their plans for the current day. Tools: Slack, Google Hangouts. Formal documentation follows the same criteria as the Task Understanding process.	Each team leader creates a work plan, and in the weekly meeting each team member describes what they will get done during the week. Both P2 and P3 reported having defined quality standards which make their work flow a lot easier, however, *"coming into that standing order was a 3-hour debate back in the day."* [P2] Plans are divided into individual goals: Git issues are then created for each goal and later visualized using Waffle.io. Tools: GitHub issues, Waffle.io.

Table 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.

Reflection on the Enacting process

In both projects, interviewees highlighted that their team purpose is to always follow the plan, however, unexpected things usually come up. This situation was experienced more often by P1's team. Also, all interviewees reported actively using Slack to support individual work and their teammates' work. In both projects, participants use a specific Slack channel as an internal question-answer system, where anyone can get help or provide prompt feedback.

When discussing strategies for motivational engagement, P1 reported having a supportive team culture although this was never explicitly discussed among members. P2 and P3 mentioned that all team members openly contribute with strategies for maintaining motivation and engagement. Table 5 presents the Enacting work practices of both projects.

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 were aware of this fact and reported established practices they follow to make adaptations to their work, as shown in Table 6. Also, P2 and P3 said that documentation about changes is not always created, and if it is, the rationale of the change or adaptation is often omitted.

	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, things are sometimes modified during plan execution due to the dynamic nature of the software projects.** *"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]
Self-regulation	P1 tries to follow their work plan, however, requirements tend to change often (within hours), which also changes the expected output of their work.	P2 and P3 spend time each day making sure everyone on the team is on track, which is expected from their administrative positions.
Co-regulation	Team members use a 'help request' Slack channel to *"report issues if we are stuck and having troubles with the task we said we were going to complete that day."* [P1]	The team helps each other execute their individual tasks using a 'developer channel' Slack channel which was designed for *"Trying to encourage people to broadcast what they are doing and encourage feedback and cross-communication."* [P3]
Shared-regulation	Same as the co-regulation mode for Enacting - Plan execution.	Same as the co-regulation mode for Enacting - Plan execution.
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.***"Feeling everybody wants to help you is what makes it good [...] knowing that you're not given a task and sent alone to a room helps."* [P1]	**All team members contribute with strategies and support for motivational engagement.** *"People are very crucial—to help each other and even feel sympathetic when it is really shitty."* [P2]
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': *"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 a sudden you're like in the groove."* 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."*
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. *"It's uncompetitive and so people are very much willing to help and they care about each other's happiness."* [P3]
Shared-regulation	There is no collective discussion about motivational and engagement strategies, but all teammates help to maintain the supportive culture.	*"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] Team members also encourage a culture of mutual support: *"trying to use a bit of thanks and cheering upon the [Slack] channel when things are going well."* [P2]

Table 5. Enacting, 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.

	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 mode listed above.	If a change is required, it is implemented but documentation is created only if *"we think it's important and useful for others to know."* [P3]
Shared-regulation	New plans or adaptations are defined by the project managers. 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, *"sometimes they would not have 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 directly edit the original bit, which won't show the change."* [P1]	Adaptations are documented on Google Drive most of the time: *"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, the increasing team complexity (team size) is affecting how much documentation is produced: *"As we are growing, we are doing more documentation."* [P3]

Table 6. 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.