University of Victoria

Department of Computer Science

PhD Candidacy Proposal

Beyond Agile: Studying The Participatory Process in Software Development

Author:
Alexey Zagalsky

Supervisor:
Dr. Margaret-Anne Storey

Committee Members::
Dr. Arie van Deursen
Dr. Leif Singer

December 2, 2015

Preface

> *"Nothing in the world is worth having or worth doing unless it means effort, pain, difficulty."*
> – Theodore Roosevelt

Writing this research proposal was not easy for me, in fact it was quite challenging. Beyond overcoming the regular writer's block, I think the real challenge was actually attempting to piece all the "puzzle" pieces together—all the work done so far—and writing in a cohesive manner. I guess the scary part was not being certain if and how the pieces would connect to each other. By doing this process, I realized that writing the research proposal benefits me more than I initially anticipated. Not only it serves as a progress report to my supervising committee, but it forced me to stop, and reflect on the steps I've taken. To digest and re-examine the work I've done, enabling me to see that there is unique and real value to my work, and see what that value is.

Does this mean I got everything figured out and planned from now on forward? No, but now I have some ground to stand on, and a direction to follow.

Contents

1 Introduction

Software development is a knowledge-building process [11, 15]. Software is built with the *tacit knowledge* in the developers' head, and the *externalized knowledge* embodied in the development tools, channels, and project artifacts [11]. However, this knowledge-building process is evolving. The formation of communities of practice and the emergence of socially enabled tools and channels has led to a paradigm shift in software development [21] and the creation of a highly tuned *participatory culture*. Modern software development is collaborative, global, and large in scale (e.g., the *Ruby on Rails* project has more than 2800 contributors[1]). Many developers now contribute to multiple projects, and as a result, project boundaries blur, not just in terms of their architecture and how they are used, but also in terms of how they are authored. Indeed, we see a participatory culture [7] forming within many development communities of practice [27] where developers want to engage with, learn from and co-create with other developers. Developers care not just about the code they need to write, but also about the skills they acquire [18], the contributions they make and the connections they establish with other developers. These activities, in turn, lead to more collaborative software development opportunities.

We see that the collaborative and participatory nature of software development continues to evolve, shape and be shaped by communication channels that are used by developer communities of practice [8]—both by traditional communication channels (e.g., telephone, in-person interactions), as well as social features that may be standalone or integrated with other development tools (e.g., email, chat, and forums). Within a community of practice, software is a combination of the externalized knowledge (e.g., code, documentation, history of activities) as well as the tacit knowledge that resides in the community members' heads (e.g., experience of when to use an API, or design constraints that are not written down). Communication channels and development tools support developers in forming and sharing both externalized and tacit knowledge in a highly collaborative manner.

However, not much is known about the impact this participatory culture is having on software development practices nor on the knowledge-building processes. The research community has been studying the tools and communication channels used by developers [19, 21] in an effort to broaden our understanding. But, studying only the tools and channels can only provide a narrow perspective. We believe that software development has evolved beyond the agile process, into a *Participatory Process*—A knowledge building process which is characterized by the (1) **knowledge activities** and **actions**, (2) **stakeholder roles**, and (3) is enabled by socially enhanced **tools and communication channels** (as illustrated in Fig. 1). Thus, it is important to gain an understanding of each one of the components, and the way these components interact and shape each other. Beyond studying how software engineering is practiced, it is equally important to consider the software engineering **research methods and practices**, and software engineering **education paradigm**, as they feed and shape each other, and are part of the same ecosystem.

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

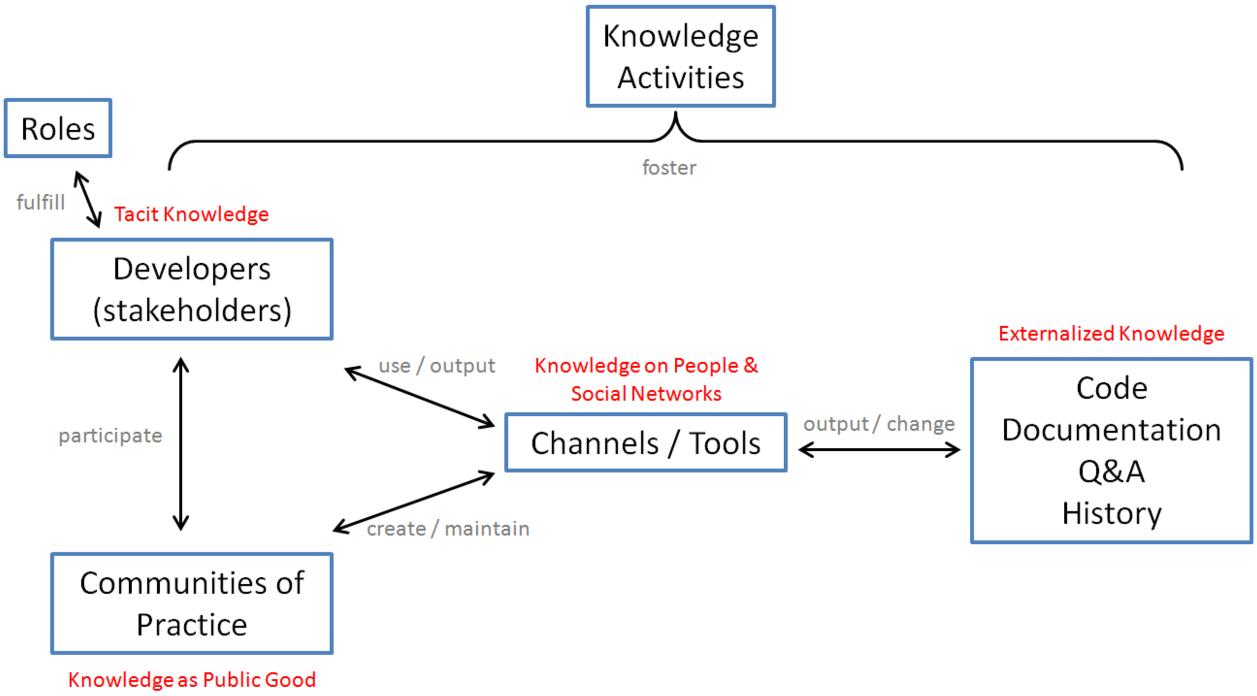


Figure 1: Knowledge theory in software development.

1.1 Research Goal

The overarching goal of the proposed thesis is to form a theory of knowledge in software development, similar to the work of Reinhardt *et al.* for knowledge workers [14, 13]. This theory emerges from our studies of the participatory process in software development. Figure 1 represents the knowledge theory in software development as we understand it now, and as this is a work in progress, the theory continues to develop and shape our research questions.

1.2 Research Questions

First, we want to gain an understanding of the involved components (e.g., tools and channels, activities) in the knowledge theory. An important yet challenging task, as these components work together and investigating them separately produces only a partial understanding. Thus, our first research question is:

RQ1 What characterizes a software development knowledge process?

A. What are the tools and channels developers use?

B. What are differentiable roles in software development?

C. What are typical knowledge activities and actions that developers perform?

Next, we aim to understand how the participatory process is affecting software development (enhances or hinders). The purpose is to use the components and terminology gained through RQ1. This leads to our second research question:

RQ2 How does the participatory process shape and challenge software development?

- **A.** How do communication channels shape development practices and activities?
- **B.** How do communication channels challenge developers?

And finally, we aim to demonstrate how a knowledge process, i.e., the participatory process, can be captured and investigated by using the resulting framework. This would serve as a validation of our work, and as a guide for other scholars. Thus, the third research question is:

RQ3 How can we capture the participatory process?

Answering and continuously refining these research questions will guide our work.

1.3 Expected Contributions

The proposed thesis aims to provide the following main contributions. First, we aim to characterize the participatory process in software development. This would help identify and define the involved components. Second, the proposed thesis will describe a *Participatory Process* framework, that could be used by researchers and practitioners for studying or supporting software development. The framework will encourage researchers to not only use a narrow "lens" (e.g., communication channels, roles, or activities), but rather consider all aspects. And lastly, we aim to demonstrate the use of the proposed framework by applying it to a specific case study (to be designed later).

2 Background

We would like to start with a discussion on software development as a knowledge building process, describing the different types of knowledge (depicted in Fig. 1). Then discuss communities of practice in software development as knowledge building communities and their role in the participatory culture.

2.1 Software Development as a Knowledge Process

In 1985 Naur [11] described programming as a *"(knowledge) theory building"* activity, suggesting that programming (in the loose sense) is not just about producing a program, but rather are the developers' insights, theory, and knowledge built in the process. By describing two real cases, Naur shows that programming involves not only the source code and the documentation (i.e., **externalized knowledge**), but also the knowledge in developers' heads (i.e., **tacit knowledge**). He further explains that tacit knowledge is essential to software support and modification—if a developer was to leave the project, the tacit knowledge allowing these activities would be lost (e.g., design reasoning). This paper of course was written before developers formed and worked in communities of practice, where knowledge can be generated and maintained by the community, preserving the knowledge even when individual members leave. Tacit knowledge can be further decomposed into **procedural knowledge** and **declarative knowledge** [15]. Procedural knowledge is dynamic and involves all the information related to the skills developed to interact with our environments. This knowledge is acquired from practice and experience. While declarative knowledge is based on facts, static, and concerned with the properties of objects, persons, and events and their relationships. Declarative knowledge is easy to describe and to communicate, as opposed to procedural knowledge.

The use of communication media in software development further extends the knowledge building process by enabling the transfer of knowledge between stakeholders. This facilitates individual learning and expression, as well as coordination and collaboration between members in a community. Wasko *et al.* [25] distinguish different theories of knowledge based on the kind of knowledge it helps capture or communicate: **knowledge embedded in people** that may be tacit or embodied within people's heads, with its exchange typically done one-on-one or in small group interactions; **knowledge as object** that exists in artifacts and can be accessed independently from any human being; and **knowledge as public good** that is "socially generated, maintained and exchanged within emergent communities of practice." We added a fourth type, based on our work [21], designed to capture **knowledge about people and social networks**.

2.2 Communities of Practice (i.e., Knowledge Building Communities)

Knowledge management in software development is an ongoing challenge [1]; it involves the creation, capture, sharing and distribution of knowledge across a community. Since the formation and growth of a community relies on the complex interactions of its members [21], it has been suggested that successful collaboration strongly depends on the social component of the community.

The core components of a **community of practice** [9, 25] are the **domain**, the **practice**, and the **community** [26]. The domain, or shared interest, defines the identity of the community. The practice identifies members of a community as **practitioners** that are constantly developing and sharing a set of resources (e.g., tools, documentation, histories, or experiences) to address recurring problems. And the community comprises the activities in which members engage in. Wenger [26] asserts that, given the proper structure, practitioners can be the best option for managing the construction of knowledge (e.g., Stack Overflow). Communities of practice have been the focus of study in other domains as well. In the learning sciences, researchers refer to it as a **Knowledge Building Community** *"that supports discourse and aims to advance the knowledge of the members collectively while still encouraging individual growth that will produce new experts and extend expertise within the community's domain"*.[2]

Empowered by the *"inexpensive and low barriers to publish, as well as the rapidly spreading peer-to-peer, large-scale communications made possible by social media"*, communities of practice have become an extensive part of software engineering [21]. The widespread adoption of socially enabled tools and channels facilitates virtual communities of practice and enables global software teams. Consequently, the availability of additional communication channels (e.g., micro-blogging) and a broad catalog of developer tools have increased the participatory culture among software developers. More importantly, the diversity of these tools and features posits a challenge in understanding which

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

may be the best composition or combination of tools that will enhance the *productivity* of collaborating developers—a challenge the proposed thesis aims to help with.

2.3 A Note on Terminology

This proposal follows the following rules regarding terminology: In software engineering, a *development process* (in some cases called a method or methodology) defines a set of development phases, activities and actions, roles, artifacts, and suggested tools and practices. For example, the agile process defines the actions (e.g., user research, design, coding, testing), the roles (e.g., integrator, product owner), and suggested practices (e.g., sprints, release early, daily stand-up meetings). Based on the activity theory [23] and the hierarchical model of human activity [4], we consider an *activity* as high level set of actions where the subject is the developer. Where as an *action* is a lower level entity, composed of *operations*. Each of which is driven by a different need: motive, goal (or purpose), and condition [23] (respectively). If we use Tell and Babar's provided example, an activity can be *"designing"*, and is consisted of *"analysis"*, *"synthesis"*, and *"evaluation"* actions. They further decompose actions into operations, e.g., *"evaluation"* consists of *"planing AE"* and *"preparing and managing results"*. In this research proposal, we discuss the participatory process at the activity level, however, when defining the participatory process framework (later in the thesis phase), we'll follow Tell and Babar's activity theory, and consider the full activity system: subject, object, rules, instrument, community, and the division of labor.

3 Understanding Communication Channels

In the following sections, we present what we know on the participatory process in software development and its main components (see Fig. 1)—communication channels, developers, and knowledge activities—thus answering **RQ1**. We highlight our work so far, and briefly describe the main findings. For each part, we indicate whether the work has been published, awaiting review, or partially completed. Here, we start with discussing how communication channels shape the participatory process in software development (**RQ1** and **RQ2**).

3.1 Communication Media Shapes the Participatory Process in Software Development

This work has been published in Proceedings of ICSE 2014, FOSE track

In this phase, we examine how communication media supports the flow of knowledge in software development [21]. We form a retrospective of the different communication channels since the late 1960, and categorized them based on the theories of knowledge provided by Wasko *et al.* [25].

Our findings reveal the transition has happened over time, showing the emergence of channels that are increasingly socially enabled, and a rise in the adoption of channels for exchanging community knowledge and social networking content (as illustrated in Fig. 2). The timeline highlights how we have entered a social era in terms of media use in software engineering. Although this provides us some insightful research findings about the implications of the paradigm shift in software engineering, we lack insights on how various media channels are used in combination to support software engineering activities.

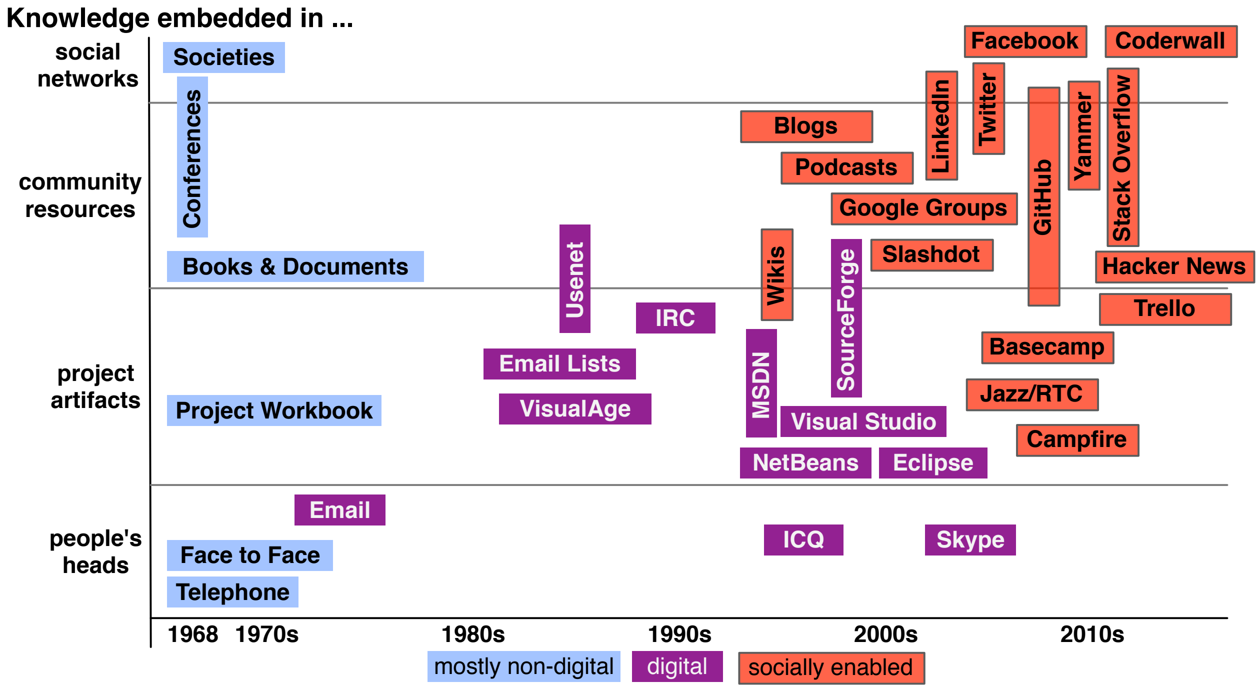


Figure 2: Media channels over time and how they support the transfer of developer knowledge. Note that some channels overlap multiple types of knowledge communication.

To try to understand how the newer generation of developers uses a broad range of social media channels, we conducted a large scale survey (1,516 responses) with members of a thriving participatory community through GitHub. The preliminary findings indicate that our respondents use a median of 12 unique channels (mean: 11.55, min: 1, max: 21). While 25% of the respondents use between 14 and 21 unique channels! The survey also unveils an initial set of challenges developers face in this new participatory culture—most mentioned challenges were: media literacy, anti-social behavior, trusting content, and vendor lock-in. We used this initial set of challenges as a starting point for a follow-up study (described in Section 4.1) where we investigated how communication media shapes and challenges a participatory culture in software development [22].

3.1.1 How Does This Fit in The Thesis

This study allows us to gain a retrospective of the tools and channels used to support software development, and understand their evolution (addressing **RQ1-A**). Moreover, this study lays the basis and served as a precursor for studying the challenges developers face (contributing to **RQ2-B**), and the activities they do (contributing to **RQ1-C**)—both are critical to understanding the knowledge building process.

3.2 Communication Channels Shape Developer Practices: Understanding How Software Teams Use Slack

This work is under review for CSCW 2016, Interactive Poster Submission

> *"We shape our tools and thereafter our tools shape us."*
>
> – Marshall McLuhan

Slack is playing an increasingly significant role in shaping the way software developers collaborate and communicate. In just two years, it has gained more than 1.1 million daily users and over 900k integrations[3]. Slack not only facilitates team messaging and archiving, it also supports a wide plethora of integrations to external services and bots (e.g., GitHub, Asana, Jira, Hubot). Moreover, Slack is disrupting and shaping developers' activities and practices within the modern, social, fast-moving and sometimes overwhelming development environment [21].

> *"If Slack is taken out, it's just like turning our lights off. We're blind."*
>
> – an anonymous developer

To understand how Slack impacts development team dynamics, we designed an exploratory study to investigate how developers use Slack and how they benefit from it. Understanding **how** and **why** developers use Slack to support their work is essential to gaining insights into the modern software development activities, and how these activities are being adapted to the participatory process.

3.2.1 Methodology

In this exploratory study, our goal is to understand how Slack and its integrations are used by software developers. In turn, allowing us to understand how modern communication tools are shaping development practices and activities (**RQ2-A** and contributing to **RQ1-A**). Our study is guided by the following secondary research questions:

SRQ1 What do developers use Slack for and how do they benefit?

SRQ2 What bots do developers use and why do they use them?

So far, we have deployed two surveys with open-ended questions to developers who adopted Slack. For the **first survey**, we targeted software developers who use Slack. We promoted it to 30 development-related Slack teams (that are publicly open) and also promoted it through Twitter. We received 53 responses to the first survey[4]. From the analysis of this survey, we realized that bots played a significant role in software development processes. Thus, for a **second iteration of the survey**, we refined some of the questions[5] and focused the distribution of it to developers that customized

[3] `http://techcrunch.com/2015/06/24/as-slack-hits-1m-daily-users-and-900k-integration-installs-it-hires-april-under`

[4] Survey used in the first phase: `http://goo.gl/forms/mnGhSZCNtY`

[5] Survey for the second phase: `http://goo.gl/forms/lZpGXPE6kH`

Slack bots—specifically we emailed 650 developers who forked or starred the "hubot-slack" project on GitHub[6]. We received 51 responses to this second survey.

3.2.2 What Do Developers Use Slack For and How Do They Benefit?

Our analysis reveals that developers use Slack for: personal, team-wide, and community-wide reasons.

Personal benefits: Developers use Slack as a gateway to **discover and aggregate news and information.** *"[I use slack for] RSS reader/bookmarking site for reliably interesting/relevant blogs (e.g. Signal vs. Noise, Rocky Mountain Institute, etc)".* They also use the instant messaging features to support **networking and social activities** with developers who share similar interests (e.g., Android developers) or have similar jobs. Surprisingly, they also use it for **fun**, as participants told us they use it for *"sharing gifs and memes"* (through Giphy, one of the most popular bots used) and for *"gaming"*.

Team-wide purposes: Slack's messaging feature is used widely for **communication**: *"[I use Slack for] communication with teammates (almost exclusively, we're a remote team)."* But, the way it is used varies. For some, it is used for remote meetings and note taking, for communication with other stakeholders (such as customers through live-chats) and for non-work topics. It further supports **team collaboration** through team management, file and code sharing, development operation notifications, and software deployments (i.e., **Dev-Ops**) and team Q&A.

Community support: Slack provides extensive support for **participation in communities of practice**, or special interest groups: *"[I use Slack for] keeping up with specific frameworks/communities".* Other participants mentioned using it for *"bouncing ideas off of other people in the community"* or use it for *"learning about new tools and frameworks for developing applications."*

3.2.3 How Does This Fit in The Thesis

Slack is the perfect motivation for building a knowledge theory, or a participatory process framework. This is a new tool/channel which is extensively used by software teams. However, we lack the tools and terminology to understand *why* and *how* it is used. Furthermore, Slack shapes and changes the behavior of developers (**RQ2-A**), while those in turn shape the design of the tool itself. To follow McLuhan's tetrad [10]: How does Slack **enhance** and support software development teams (e.g., collaboration and awareness)? What does it **make obsolete** (e.g., data is no longer siloed) ? And how does it flip when **taken to extreme** (e.g., what happens when Slack is down)? This thesis could help in answering those questions (**RQ2-B**).

3.2.4 What is Next?

This exploratory study was a preliminary phase. It allowed us a glimpse into **how** and **why** developers use Slack. However, our insights are somewhat superficial, as expected for an exploratory phase. Thus, the next step should aim for extracting deeper insights for the same research questions, e.g. through the use of interviews with developers. In fact, we have already moved forward

[6] `https://github.com/slackhq/hubot-slack`

by conducting an interview with developers from a local company (SendWithUs) that uses Slack, revealing interesting insights. But more importantly, we lack an appropriate framework or typology of knowledge activities in software development, and as a result it limited our ability in refining our research method and findings (e.g., frame the findings in terms of supported activities). We are currently working on building a **knowledge activity typology** (further described in Section 5.2). Additionally, we need to look at what happens when this new medium is taken to the extreme. What challenges does this new medium introduce? For example, some communities already object to using Slack[7].

As part of our study on how Slack is used by development teams, we noticed an interesting phenomena of *social bots* that support developers (and other stakeholders). Next, we discuss the natural extension from the *social programmer* to the use of *social bots* in supporting software development, and how it may affect knowledge flow.

3.3 The Rise of The Social Bots in Software Development

This work has been partially completed

Social media has dramatically changed the landscape of software engineering, challenging some old assumptions about how developers learn and work with one another. Bringing the rise of the *social programmer* who actively participates in online communities and openly contributes to the creation of a large body of *crowdsourced socio-technical content* [21]. This social developer is a **lifelong learner**, engaged in a continuous cycle of updating one's skills and further developing one's competencies in both formal and informal educational settings [14].

Developers engaging in the participatory culture have recently adopted a new and versatile channel—the **social media bots**[8]. Our exploratory study of how and why developers use Slack (see Section 3.2), revealed that developers use bots to retrieve or post messages (e.g., the Twitter bot reposts tweets to Slack), to integrate other communication channels such as audio, video, screen sharing (e.g., Screenhero), and email. Bots are also used for information acquisition through news aggregators. Some developers customize bots to fit their needs (e.g., to initiate coordination in the team, or to support trivial tasks). However, not much is known on how these bots affect software development and the knowledge building process.

3.3.1 How Does This Fit in The Thesis

In a similar manner to studying Slack, studying how social bots shape (or hinder) software development can guide researchers and practitioners (**RQ2-A**). Moreover, bots could perhaps become a separate entity in the knowledge theory diagram (see Fig. 1).

[7] `https://drewdevault.com/2015/11/01/Please-stop-using-slack.html`

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

3.3.2 What is Next?

The use of social bots in software development is an emerging phenomena, one that has not been studied so far. The use of bots changes and shapes the developers' practices and the activities they partake in. For example, developers interact with bots inside the communication channels (e.g., Slack), transforming these communication channels into a command-line interface. As a result, introducing benefits such as improved team awareness, and supporting the ability for non-technical stakeholder to interact directly with the system (e.g., deploying a new version through the use of a bot). However, what challenges does this introduce? (e.g., signal vs. noise issue). Thus, the following research questions should guide us in the next phase:

SRQ1 What knowledge activities developers use bots for? (contributing to **RQ1-C**)

SRQ2 What challenges does the use of social bots introduces? (contributing to **RQ2-B**)

SRQ3 How do bots affect the knowledge flow in software development?

As an initial phase, we have conducted an interview with developers in a local company, SendWithUs. The interview is being currently transcribed.

4 Understanding Developers and Their Roles

In this section, we present our work relating to the study of developers. We focus in this section on the human aspect of the participatory process, that is, we discuss studies aiming to understand the challenges developers face (**RQ2-B**), the way they collaborate (**RQ1-A**), and the roles they fulfill (**RQ1-B**). First, we start by examining challenges developers experience when using communication media to support software development.

4.1 How Communication Channels Challenge A Participatory Culture in Software Development

This work is under review for a TSE journal

> *"What does the medium flip into when pushed to extremes?"*
>
> – Marshall McLuhan

Software developers use many different communication tools and channels in their work. The diversity of these tools has dramatically increased over the past decade, giving rise to a wide range of socially-enabled communication channels and social media that developers use to support their activities. The availability of such social tools is leading to a participatory culture of software development, where developers want to engage with, learn from, and co-create software with other developers. However, the interplay of these social channels, as well as the opportunities and **challenges** they may create when used together within this participatory development culture, are not yet well understood. In this study, we investigated how the choice of communication channels shapes the activities that developers partake in within a participatory culture of development, as well as explore the challenges they may face.

4.1.1 Methodology

We designed and conducted a survey to learn how developers use tools to support their knowledge activities, to learn which media channels are important to them, and to determine which challenges they may face. Our secondary research questions were:

SRQ1 *Who is the social programmer* that participates in these communities?

SRQ2 *Which communication channels* do these developers use to support development activities?

SRQ3 Which communication channels are the most *important* to developers and *why*?

SRQ4 *What challenges* do developers face using an ecosystem of communication channels to support their activities?

We deployed the same survey during two different time periods: at the end of 2013 and at the end of 2014, and 1,516 developers responded (21% response rate).

4.1.2 The Challenges Developers Face Using Communication Channels

We found that on average developers indicated they use 11.7 channels across all activities, with a median of 12. We also asked developers to indicate the top three channels that are important to them (Figure 3 shows the number of responses given per channel). Surprised by the high number of channels, we asked ourselves what happens *"when the medium is pushed to the extreme?"*

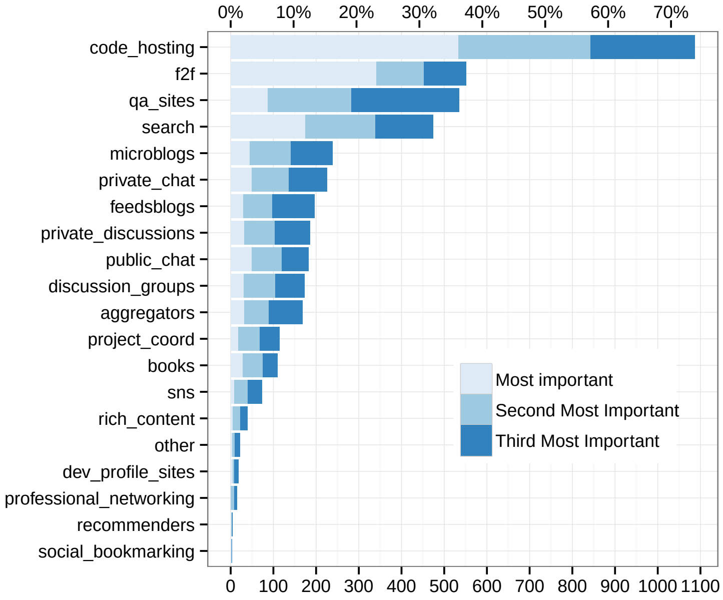


Figure 3: Number of responses per channel indicating the importance of each channel.

Thus, we would like to focus on research question **SRQ4** which inquires about the challenges developers face using communication channels. From our previous work [18, 19, 21], we were already aware that developers face challenges related to distractions, privacy and being overwhelmed by communication chatter using social media channels. Consequently, the survey asked specifically whether the respondents experienced these concerns. Fig. 4 shows the results from the Likert-style questions of the survey. We note that privacy is not a big concern for everyone, whereas being interrupted and feeling overwhelmed by communication traffic are issues for more developers.

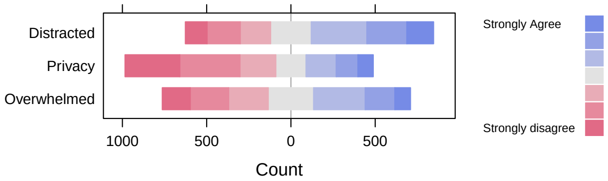


Figure 4: Frequency of responses to Likert questions probing on developer challenges.

We further asked the respondents to share with us any **additional challenges** they face through an open-ended text question. 432 respondents (356 to the 2013 survey, 76 to the 2014 deployment) either elaborated on the challenges mentioned above, or informed us about other challenges they face. A wide variety of challenges were reported and we coded, sorted, clustered and then categorized them using an open coding and iterative clustering technique. Postman, who studied the use of media in communities of practice extensively, refers to a *media ecology* and suggests we undertake the study of "the entire environments, their structure, content and **impact on people**." [12]. We note that the categories we found mirror the main areas of study suggested by Postman. The main categories of challenges that emerged from our analysis are as follows (we report these challenges in detail in the paper itself):

- Developer issues
- Collaboration and coordination hurdles
- Barriers to community building and participation
- Social and human communication challenges
- Communication channel affordances, literacy and friction
- Content and knowledge management concerns

4.1.3 How Does This Fit in The Thesis

Understanding developers and stakeholder roles is an essential part of building a knowledge theory and understanding the participatory process. Thus, the study of the challenges developers face was important. In fact, preliminary findings regarding the challenges relating to the developer activities shows similar challenge categories to the ones emerging from this study.

4.1.4 What is Next?

The challenges in this study focused on challenges when using communication channels to support software development. However, there is a need to also understand the challenges developers have in regard to the knowledge activities they partake in (something we already began investigating, and is discussed in Section 5.2).

4.2 Developer Roles in Software Development

This work has not been done yet

4.2.1 How Does This Fit in The Thesis

Developer (stakeholder) roles is an essential part of understanding the knowledge flow, and it would allow to identify different types of stakeholders and better understand the "expected behavior pattern" (**RQ1-B**).

4.2.2 What is Next?

This study hasn't been conducted yet. A main reason was that we wanted first to build the activity typology to guide us in this phase. The proposed methodology would be a mixed-methods methodology, consisting of a survey, interviews, and inspection of team data (e.g., Slack conversations, GitHub). The expected result is a typology of roles, describing the: roles, a description of each role, and typical knowledge activities and channels/tools associated to that role.

4.3 Studying Collaboration As Part of The Participatory Process

This work has been published as a preliminary work at CHASE 2015 workshop, and is under review for ICSE 2016, research track

Comprehending the synergetic nature of software development is difficult as collaboration is a complex activity that relies on awareness, communication, and coordination. However, gaining an understanding of collaboration in today's participatory development culture is even more challenging as there is a growing number of stakeholders involved in its development and feature needs evolve rapidly as software is continuously deployed. In our experience, the existing models of collaboration for software development (e.g., the 3C model [5]) are not powerful enough to capture the way different actors and stakeholders work together, nor do they capture how stakeholders have to respond to emergent requirements. These models focus heavily on task coordination, task completion, and stakeholder communication, but they do not adequately take into consideration knowledge co-construction by emergent stakeholders.

The challenge of modeling dynamic collaboration processes is not unique to software engineering and is shared by other domains that focus on knowledge work. To address the above mentioned shortcomings of the existing collaboration models, we worked on borrowing, adapting, and extending a model of regulated learning from the education domain known as the Model of Regulation [6]. *The Model of Regulation* describes distinct **regulation forms** (self-,co-, and socially shared-regulation), it defines **regulation processes** (task understanding, goal setting, enacting, large scale adaptation, and monitoring & evaluating), and it defines the **types of process support** provided by tools or channels (structuring support, mirroring support, awareness tools, and guiding systems). In this work, we aim to explore the viability of using the Model of Regulation to understand collaboration support in software development (e.g., in tools such as GitHub, Slack, Trello).

4.3.1 Methodology

Our research methodology consisted of a two-phase qualitative case study [16, 28] investigation. In the first phase, our purpose was *exploratory* as we investigated the feasibility of using the Model of Regulation as a way to understand the dynamics of the collaboration involved in the development of an open source project. The research question that guided this phase is:

SRQ1 Can the Model of Regulation be used to identify how regulation occurs in software development?

Phase I revealed new complexities in the Model of Regulation when multiple actor types with different investments—*Users* and *Contributors*—are involved in collaboration. Having adapted and refined the model, we then transitioned into phase II and investigated how regulation occurs in software development and how users participate in that regulation. We were guided by our second research question:

SRQ2 How do users participate in the regulation of an open source project?

Both phases of our study investigated the Neo4J[9] community. It is important to note that despite using the same *"case"* for both phases, they had different purposes (*exploratory* vs. *descriptive*) and relied on different processes and data.

4.3.2 Understanding The User's Role in the Regulation of an Open Source Project

By adapting and applying the Model of Regulation to a software project, we were able to gain an understanding in the user's role in the regulation of the project. Our findings show that: user participation **promotes task understanding** processes among contributors; user participation **contributes to the definition of project goals and standards**; close collaboration between users and contributors **helps them reach agreement** on the strategic selection of tools and resources to achieve their goals; and, users **play an important role in evaluating the project and supporting collaboration**;

These findings may not seem surprising, however, the main contribution of this work allows to identify and capture these instances of regulation and collaboration (examples and an extensive description of the Model of Regulation are described in detail in the paper itself). Figure 5 shows a representation of regulation process in an open source project (taken in a snapshot in time).

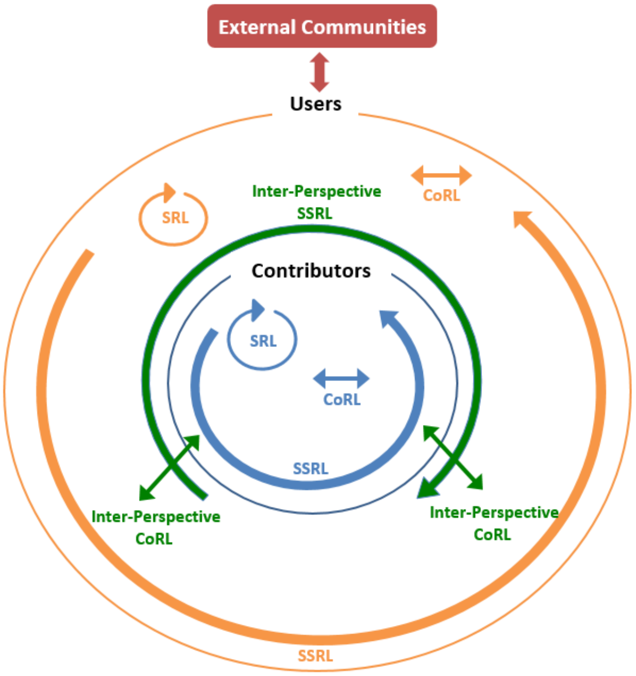


Figure 5: A representation of regulation forms in an open source project

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

4.3.3 How Does This Fit in The Thesis

The adapted Model of Regulation acts as a lens that allowed us to understand the specific types of regulation between users and project contributors in a collaborative software development project, a project that is an example of a participatory community. Furthermore, the Neo4J community showcases several knowledge activities, e.g., *co-authoring*, which can be supported by regulation process and thus allowing us to better understand these collaborative activities as part of the participatory process.

4.3.4 What is Next?

Next step should consider an analysis of the contributor's community and their use of communication tools to support their regulation processes. In this study, we studied communication in three different channels (namely GitHub issues, Google Groups and Stack Overflow) as a way to triangulate how regulation forms and processes from the Model of Regulation may be evident in collaborative software development. Future work should delve more deeply into the question of channel affordances for regulation, as well as consider other channels such as Slack (which the Neo4J group have recently started to use[10]). These insights about regulation and communication channel affordances should help practitioners decide which constellation of tools would give the best support for regulation in their collaborative projects.

[10] `http://neo4j.com/blog/public-neo4j-users-slack-group`

5 Understanding Knowledge Activities in Software Development

Next, we present our work relating to the study of knowledge activities in software development (**RQ1-C**). This work is a *cornerstone* in my thesis. We describe below our gradual progress achieved over several studies.

5.1 Communication Channels Developers Use to Support Their Development Activities

This work is under review for a TSE journal - we described part of it in Section 4.1

In the study described in Section 4.1 we also inquired on the purpose of using each channel—i.e., which channels used for what activity. Table 1 shows an overview of which channels developers said they use to support different activities. This is a preliminary list of knowledge activities based on previous studies and a literature review. In this table, we have grouped the channels according to *analog*, *digital* and *social+digital*. We can see from this table that there appears to be more reliance on communication channels that support social features.

	Face-to-face	Books	Web Search	Content Recommenders	Rich Content	Private Discussions	Discussion Groups	Public Chat	Private Chat	Feeds and Blogs	News Aggregators	Social Bookmarking	Q&A Sites	Prof. Networking Sites	Developer Profile Sites	Social Network Sites	Microblogs	Code Hosting Sites
	analog		digital							digital and socially enabled								
Stay Up to Date	50-60%	30-40%	70-80%	0-10%	20-30%	30-40%	40-50%	20-30%	30-40%	50-60%	50-60%	0-10%	50-60%	10-20%	10-20%	30-40%	30-40%	70-80%
Find Answers	40-50%	10-20%	80-90%	0-10%	0-10%	20-30%	40-50%	20-30%	20-30%	20-30%	0-10%	0-10%	80-90%	0-10%	0-10%	0-10%	10-20%	40-50%
Learn	40-50%	40-50%	60-70%	0-10%	20-30%	20-30%	30-40%	20-30%	20-30%	40-50%	20-30%	0-10%	60-70%	0-10%	10-20%	10-20%	20-30%	60-70%
Discover Others	30-40%	0-10%	30-40%	0-10%	10-20%	10-20%	20-30%	10-20%	10-20%	30-40%	20-30%	0-10%	30-40%	10-20%	10-20%	20-30%	30-40%	60-70%
Connect With Others	30-40%	0-10%	0-10%	0-10%	0-10%	30-40%	20-30%	20-30%	20-30%	10-20%	0-10%	0-10%	20-30%	10-20%	0-10%	20-30%	30-40%	60-70%
Get and Give Feedback	30-40%	0-10%	0-10%	0-10%	0-10%	30-40%	30-40%	20-30%	20-30%	10-20%	0-10%	0-10%	30-40%	0-10%	0-10%	10-20%	20-30%	60-70%
Publish Activities	10-20%	0-10%	0-10%	0-10%	0-10%	10-20%	10-20%	0-10%	10-20%	20-30%	0-10%	0-10%	10-20%	10-20%	0-10%	20-30%	30-40%	80-90%
Watch Activities	10-20%	0-10%	10-20%	0-10%	0-10%	0-10%	10-20%	10-20%	0-10%	30-40%	10-20%	0-10%	10-20%	0-10%	0-10%	20-30%	30-40%	80-90%
Display Skills/Accomplishments	10-20%	0-10%	0-10%	0-10%	0-10%	0-10%	0-10%	0-10%	0-10%	20-30%	0-10%	0-10%	10-20%	30-40%	10-20%	20-30%	20-30%	60-70%
Assess Others	20-30%	0-10%	10-20%	0-10%	0-10%	10-20%	10-20%	10-20%	10-20%	20-30%	0-10%	0-10%	20-30%	10-20%	0-10%	10-20%	20-30%	70-80%
Coordinate With Others	30-40%	0-10%	0-10%	0-10%	0-10%	50-60%	20-30%	20-30%	40-50%	0-10%	0-10%	0-10%	0-10%	0-10%	0-10%	0-10%	0-10%	70-80%

Legend:	0-10%	10-20%	20-30%	30-40%	40-50%	50-60%	60-70%	70-80%	80-90%	90-100%

(percentage of survey respondents mentioning a channel being used for an activity)

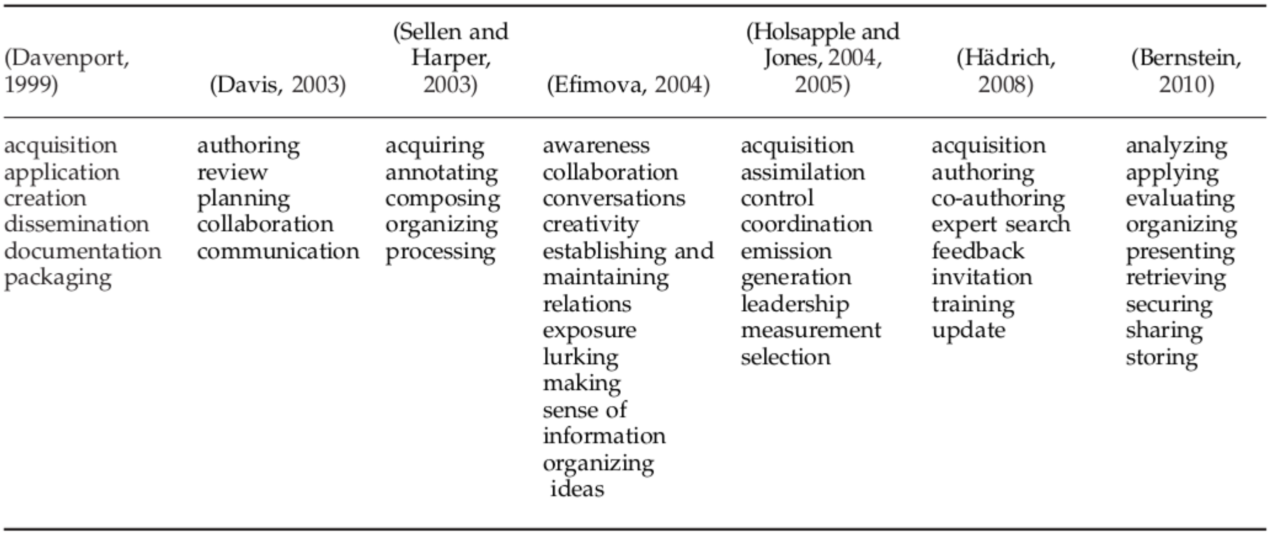
Table 1: Preliminary knowledge activities and the channels developers in our survey use for them.

In an effort to refine the preliminary knowledge activities (Table 1), we designed the a follow-up study (described next), focusing on the knowledge activities developer do. These activities in fact foster the knowledge flow in the development process (as shown in Fig. 1).

5.2 A Typology of Knowledge Activities in Software Development

This work has been partially completed

One of our main goals in the proposed thesis is to create a typology of knowledge activities in software development. Reinhardt *et al.* [14] have described a knowledge worker activity typology (which he called actions), that he based on previous work (see Figure 6) and an empirical study (involving a task execution study and a survey). Developers are a knowledge worker type (some even say developers are the knowledge worker prototype), however, Reinhardt *el al.*'s typology is insufficient in capturing the activities done in software development and so does other knowledge work activity taxonomies. As a result, we aim to form and validate a typology of knowledge activities for developers.



(Davenport, 1999)	(Davis, 2003)	(Sellen and Harper, 2003)	(Efimova, 2004)	(Holsapple and Jones, 2004, 2005)	(Hädrich, 2008)	(Bernstein, 2010)
acquisition application creation dissemination documentation packaging	authoring review planning collaboration communication	acquiring annotating composing organizing processing	awareness collaboration conversations creativity establishing and maintaining relations exposure lurking making sense of information organizing ideas	acquisition assimilation control coordination emission generation leadership measurement selection	acquisition authoring co-authoring expert search feedback invitation training update	analyzing applying evaluating organizing presenting retrieving securing sharing storing

Figure 6: Existing taxonomies for knowledge actions, as depicted by Reinhardt *et al.* [14].

Based on an iterative process of *data analysis—discussion—literature review*, we came up with the following initial activity typology. Note that the preliminary activities from the previous study are all represented below (e.g., *"Watch Activities"* is now *"Monitoring"*):

1. **Acquisition**: gathering of information with the goal of developing skills or project or obtaining an asset
2. **Authoring**: the creation code and related artifacts (externalized knowledge)
3. **Co-Authoring**: the collaborative creation of textual and medial content (externalized knowledge)

    (a) Communicating with others (including users)
    (b) Coordinate tasks with others

4. **Dissemination**: can be either of content (e.g., code or documentation) or of developer's activity (e.g., status reports)
    (a) Dissemination of content
    (b) Displaying activities
5. **Feedback**: includes both gaining and providing feedback
6. **Information Organization**
7. **Learning**: acquiring new knowledge, skills, or understanding during the execution of work or based on formalized learning material
    (a) Ask and answer questions
    (b) Using examples
8. **Monitoring**: includes the following
    (a) Staying up to date: stay up to date about new technologies, practices, trends and tools for software development
    (b) Keep track of one's development activities
    (c) Monitor one's collaborators' activities
    (d) Serendipitous discovery
9. **Networking**: creating and maintaining a (professional?) social network
10. **Searching**: looking up information, services, or experts

5.2.1 Methodology

As a first phase, we conducted in-depth interviews with developers asking **what** activities they do, **why** and **how**, what **channels and tools** they use for each activity, and what **challenges** they face. So far we interviewed 7 participants (with a 28% response rate) and transcribed 3 of these interviews. The interviews were 70-90 minutes long, and were conducted over VOIP (e.g., Skype, Google Hangouts). Note that this builds on the findings of previous studies (described in Sections 3.1 and 5.1), which helped form the initial set of knowledge activities. Thus, in this study we use interviews as their strength lies in the ability to gain a deeper understanding of each activity.

5.2.2 How Does This Fit in The Thesis

The knowledge activity typology for software development would allow us to categorize and map the knowledge flow, the channel use, and the developer roles in software development. The interviews conducted so far support the initial activity typology we built, in fact, none of the interviews revealed any new activities so far.

5.2.3 What is Next?

First we have to finish the first phase, conduct additional interviews with developers, and analyze these interviews (this work in under progress now).

The next phase will be to conduct observations of developer teams and recording the activities done in *the wild* (the subject of the observations is to be determined). This research method will complement our previous phase, as it would allow us to overcome the limitations associated with interviews (a possibility of bias and the risk of capturing only what we asked about).

6 Towards a Participatory Process in SE Research

Selecting suitable research methods for empirical software engineering research is ambitious due to the various benefits and challenges each method entails. One must regard the different theoretical stances relating to the nature of the research questions being asked, as well as practical considerations in the application of methods and data collection [2].

In this study [20], we described some of the characteristics of this emergent development culture and the tools that are used to support it. We also discussed the different tradeoffs and challenges faced in selecting and applying research methods for studying development in online communities of practice. Finally, we briefly explore how social tools could foster a more participatory culture in software engineering research and how that may help to accelerate our impact on software engineering practice.

In our studies of this participatory development culture, our goal has not been to investigate or experiment with new tools or ideas, but rather to understand the impact of the social tools developers already adopt on their development practices and on the community. We discuss two approaches, **mining software repositories** and **burrowing methods from social sciences**. We then describe how and why we **blend both** in our research.

Many software engineering researchers have already established excellent **social media literacy skills**. We have used our social graph to **gain research data and insights**, and to **form alliances** with developers and collaborations with other researchers. We also find it a useful avenue for **disseminating our research** and learning about related research in our field. It also becomes a useful **backchannel** for discussing research as it is being presented at conferences.

However, there are many challenges and risks from using social tools in our research community. We mentioned the need for researchers to develop improved **media literacy skills**, otherwise they may miss out on research developments. Another risk is that easier access to participants through public channels (such as GitHub) may lead to us **inadvertently spamming** our participants. Another issue we need to be aware of is to ensure that we do not somehow **cause harm to our participants**.

7 The Emergence of The Participatory Process in SE Education

This work has been published at CSCW 2015 conference, and a follow-up study is under review for ICSE 2016, SEET track

In this study, we examined how GitHub is emerging as a collaborative platform for education. We aim to understand how environments such as GitHub—environments that provide social and collaborative features in conjunction with distributed version control—may improve (or possibly hinder) the educational experience for students and teachers. We conducted a qualitative study focusing on how GitHub is being used in education, and the motivations, benefits and challenges it brings.

7.0.1 Methodology

To discover if and how GitHub is being used in education, our study used exploratory research methods [3, 17] and consisted of three phases of data collection (online methods [24], interviews, and a validation survey). We were guided by the following research questions:

SRQ1 How does GitHub support learning and teaching?
SRQ2 What are the motivations and benefits of using GitHub for education?
SRQ3 What challenges are related to the use of GitHub for education?

7.0.2 The GitHub Participatory Workflow

We found that GitHub can be compared to traditional learning management systems (LMS), however, GitHub allows going beyond the traditional LMS and provides many benefits when used to support teaching and learning. Benefits include: providing transparency of activity, encouraging participation, allowing reuse and sharing of knowledge (and course material), providing industry relevance, and provides a shared space for work (thus supporting a community of practice). We also noted challenges educators faced when using GitHub in the classroom: lack of a shared knowledge base, barriers to entry, and external restrictions (e.g., copyright).

7.0.3 The Student's Perspective

From our previous phase, we found that educators leverage GitHub's collaboration and transparency features to create, reuse and remix course materials, and to encourage student contributions and monitor student activity on assignments and projects. However, our previous research did not consider the student perspective. In this phase, we conducted a case study where GitHub was used as a learning platform for two software engineering courses. We gathered student perspectives on how the use of GitHub in their courses might benefit them and to identify the challenges they may face.

The research questions addressed in this work include:

SRQ1: How do students benefit from using GitHub in their courses?
SRQ2: What challenges do students face when GitHub is used by software engineering course instructors?
SRQ3: What recommendations can we provide software engineering instructors who wish to use GitHub for teaching?

The findings indicate that software engineering students do benefit from GitHub's transparent and open workflow. However, students were concerned that since GitHub is not inherently an educational tool, it lacks key features important for education and poses learning and privacy concerns. Our findings provide recommendations for designers on how tools such as GitHub can be used to improve software engineering education, and also point to recommendations for instructors on how to use it more effectively in their courses.

7.0.4 How Does This Fit in The Thesis

GitHub is on the brink of growing from a platform for software projects, and into a mainstream collaboration platform for other domains as well. An unexpected area where GitHubs collaborative workflow holds the potential to bring groundbreaking changes is education and learning by promoting a participatory culture. In fact, as we seen educators have already begun to use GitHub to support teaching and learning. Moreover, Slack is also being adopted in supporting education as well[11].

Software engineering education is an important part of the knowledge theory process in software development, and there is a bi-directional relation between the two. From one perspective, developers are on a life long journey of learning and knowledge acquisition. While on the other hand, the practice of software development (e.g., practitioners) feed back into software engineering educational system. In fact, communities of practice foster both sides of the eco-system (e.g., communities on Stack Overflow).

8 Discussion

We described an emerging theory of knowledge in software engineering, and the way we understand it through our studies of the participatory process in software development. Figure 1 represents the knowledge theory as we understand it now, and as this is a work in progress, we continue developing and shaping this theory. However, few questions come to mind.

8.0.1 Why Do We Need Another Theory

We have Wasko's [25] theories of knowledge, Reinhardt *et al.*'s [14, 13] knowledge worker theory, and Naur's theory [11]—why do we need another theory? These theories do not provide the sufficient tools and terminology to study the participatory process in software development. Moreover, we don't intend to replace these theories, but expand and build on their work. Bringing these theories and findings from the different studies is one of the contributions of the proposed thesis.

8.0.2 Is This A Theory or A Meta-Theory

A good question. For example, can this theory be used to analyze the Agile process as well (meta-theory) or does it only describe the participatory process (theory)? Perhaps yes. This knowledge theory has emerged from our study of the participatory process in software development, and the components we describe (Fig. 1) are needed in order to study the participatory culture. However, this work may be generalizable in the future, by separating between the components and the specific instance we study (i.e., the participatory process).

[11] `http://www.sqrlab.ca/blog/2015/11/29/using-slack-in-the-classroom/`

8.0.3 Limitations

We discuss limitations and threats to validity separately in each study (not described here), however, there is one concern that should be mentioned here. Three of the studies described earlier (Sections 3.1, 5.1, and 5.2) have used the same community to recruit participants, i.e., in all three studies we recruited participants that are active on GitHub (though **different** and **independent** participants each time). We used this community by choice, as GitHub is widely used by software teams. However, this may limit the generalizability of our findings, as perhaps developers who are active on GitHub may engage more in the participatory process. This concern may be addressed in the future, by expanding or changing the participation community.

9 Timeline

The work described in the previous chapters has been conducted in the first half of my thesis (2 years). The work described here was not done solely by myself, it has been done in collaboration with other researchers, and my contribution varies between the studies. Below is a tentative timeline for the completion of the activities leading to the final dissertation.

- Continue study on developer activities (incl. interviews) and form a knowledge activity typology - finish by August 2016.
- Build a framework of the Participatory Process in software development - by January 2017.
- Demonstrate the use of the framework on a developer community (e.g., SendWithUs or Neo4J) - by August 2018.
- Write up dissertation - by September 2018.

References

[1] Finn Olav Bjørnson and Torgeir Dingsøyr. Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used. *Information and Software Technology*, 50(11):1055 – 1068, 2008.

[2] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*, pages 285–311. Springer, 2008.

[3] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*, pages 285–311. Springer, 2008.

[4] Yrjo Engestrom. Learning by expanding. *Helsinki: Orienta-Konsultit Oy*, 1987.

[5] Hugo Fuks, Alberto B Raposo, Marco A Gerosa, and Carlos JP Lucena. Applying the 3c model to groupware development. *International Journal of Cooperative Information Systems*, 14(02n03):299–328, 2005.

[6] Allyson Fiona Hadwin, Sanna Järvelä, and Mariel Miller. Self-regulated, co-regulated, and socially shared regulation of learning. *Handbook of self-regulation of learning and performance*, 30:65–84, 2011.

[7] Henry Jenkins. *Confronting the challenges of participatory culture: Media education for the 21st century*. Mit Press, 2009.

[8] Filippo Lanubile. Social software as key enabler of collaborative development environments. `http://www.slideshare.net/lanubile/lanubilesse2013-25350287`, 2013.

[9] Jean Lave and Etienne Wenger. *Situated learning: Legitimate peripheral participation*. Cambridge University Press, 1991.

[10] Marshall McLuhan and Quentin Fiore. The medium is the message. *New York*, 123:126–128, 1967.

[11] Peter Naur. Programming as theory building. *Microprocessing and microprogramming*, 15(5):253–261, 1985.

[12] Neil Postman. The reformed english curriculum. In Alvin Christian Eurich, editor, *High school 1980: the shape of the future in American secondary education*. Pitman Pub. Corp., 1970.

[13] Wolfgang Reinhardt. *Awareness support for knowledge workers in research networks*. 2012.

[14] Wolfgang Reinhardt, Benedikt Schmidt, Peter Sloep, and Hendrik Drachsler. Knowledge worker roles and actionsresults of two empirical studies. *Knowledge and Process Management*, 18(3):150–174, 2011.

[15] Pierre N. Robillard. The role of knowledge in software development. *Commun. ACM*, 42(1):87–92, January 1999.

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

[17] Carolyn B. Seaman. Qualitative methods in empirical studies of software engineering. *Software Engineering, IEEE Transactions on*, 25(4):557–572, 1999.

[18] Leif Singer, Fernando Figueira Filho, Brendan Cleary, Christoph Treude, Margaret-Anne Storey, and Kurt Schneider. Mutual assessment in the social programmer ecosystem: An empirical investigation of developer profile aggregators. In *Proceedings of the 2013 conference on Computer supported cooperative work*, pages 103–116. ACM, 2013.

[19] Leif Singer, Fernando Figueira Filho, and Margaret-Anne Storey. Software engineering at the speed of light: How developers stay current using twitter. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 211–221, New York, NY, USA, 2014. ACM.

[20] Margaret-Anne Storey. Selecting research methods for studying a participatory culture in software development: Keynote. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, EASE '15, pages 2:1–2:5, New York, NY, USA, 2015. ACM.

[21] Margaret-Anne Storey, Leif Singer, Brendan Cleary, Fernando Figueira Filho, and Alexey Zagalsky. The (r)evolution of social media in software engineering. In *Proc. of the 36th Intl. Conf. on Software Engineering, Future of Software Engineering*, FOSE 2014, pages 100–116, New York, NY, USA, 2014. ACM.

[22] Margaret-Anne Storey, Leif Singer, Fernando Figueira Filho, Alexey Zagalsky, and Daniel German. How communication channels shape and challenge a participatory culture in software development. In *IEEE Transactions on Software Engineering*, Currently under review. IEEE, 2015.

[23] Paolo Tell and Muhammad Ali Babar. Activity theory applied to global software engineering: Theoretical foundations and implications for tool builders. In *Global Software Engineering (ICGSE), 2012 IEEE Seventh International Conference on*, pages 21–30. IEEE, 2012.

[24] Nina Wakeford and Kris Cohen. Fieldnotes in public: using blogs for research. *The Sage handbook of online research methods*, pages 307–326, 2008.

[25] Molly McLure Wasko and Samer Faraj. "It is what one does": why people participate and help others in electronic communities of practice. *The Journal of Strategic Inform. Systems*, 9(2-3):155 – 173, 2000.

[26] Etienne Wenger. Communities of practice: A brief introduction. 2011.

[27] Etienne C Wenger and William M Snyder. Communities of practice: The organizational frontier. *Harvard business review*, 78(1):139–146, 2000.

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