How Software Developers Mitigate Collaboration Friction with Chatbots

Carlene Lebeuf
University of Victoria
Victoria, BC, Canada
clebeuf@uvic.ca

Alexey Zagalsky
University of Victoria
Victoria, BC, Canada
alexeyza@uvic.ca

Margaret-Anne Storey
University of Victoria
Victoria, BC, Canada
mstorey@uvic.ca

Abstract
Modern software developers rely on an extensive set of social media tools and communication channels. The adoption of team communication platforms has led to the emergence of conversation-based tools and integrations, many of which are chatbots. Understanding how software developers manage their complex constellation of collaborators in conjunction with the practices and tools they use can bring valuable insights into socio-technical collaborative work in software development and other knowledge work domains.

In this paper, we explore how chatbots can help reduce the friction points software developers face when working collaboratively. Using a socio-technical model for collaborative work, we identify three main areas for conflict: friction stemming from team interactions with each other, an individual's interactions with technology, and team interactions with technology. Finally, we provide a set of open questions for discussion within the research community.

Author Keywords
Software Development, Collaboration, Chatbots, Bots, Conversational Interfaces, HCI, Socio-Technical Systems

ACM Classification Keywords
H.5.2 [User Interfaces]: Natural Language

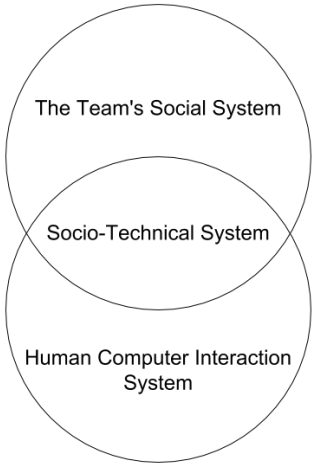

Figure 1: A simplified view of the intersection between domain models: the team's social system consists of collaborative work between people; the human computer interaction system focuses on individuals' interactions; and the socio-technical systems lies in the intersection of both.

Introduction

Many software projects have adopted team communication and collaboration tools such as Slack, Hipchat, Microsoft Teams, and Flowdock [10]. These IRC-like communication tools serve as a *central hub* for the organization, providing a fully searchable interface with persistent chat rooms, private group messages, and direct messages. Not surprisingly, distributed software teams rely on these team messaging platforms for the majority of their team communication, but collocated teams also use such systems to facilitate collaboration and promote a participatory culture. While software developers are at the forefront of the design and exploration of new tools—they can manipulate tools to fit their needs—other knowledge worker domains (e.g., journalism) are also adopting popular collaborative software development tools. Studying how software developers manage the complex constellation of collaborators in conjunction with their tools and practices provides valuable insights into understanding socio-technical collaborative work.

This widespread adoption of team-based communication platforms provides a breeding ground for new conversation-based tools and integrations. Many of these integrations are in the form of conversational bots (also referred to as chatbots). Although bots have been around for many years, the ability to easily integrate them with modern communication tools and access various APIs and data sources has created a recent explosion of new bots. These chatbots can be powered by rules, machine learning, or artificial intelligence, allowing users to interact via text or even spoken words (e.g., `MyVoiceBot` connects Amazon's Alexa to other chatbots inside Hipchat, giving the user voice control over typical text-based interactions).

Developers have been adopting these chatbots at a dizzying pace to support and fill many of the roles that software developers have traditionally filled: maintaining code quality, preforming testing, conducting development operations, supporting customers, and creating documentation [9]. Yet the research community struggles to understand how chatbots are supporting (or hindering) collaborative work, why some bots are more useful than others, and what risks they introduce.

Based on our past research, we realize that developers face friction points not only when using tools, but also when working with each other [8, 10, 3]. Recently, developers have been creating their own breed of chatbots to address the collaboration needs these friction points expose.

In this paper, we explore how chatbots help reduce collaboration friction points in software development. Specifically, we aim to answer the following research question:

RQ: What collaboration friction points can chatbots help mitigate in software development, and how?

Through a study of existing literature, an analysis of popular chatbots, our past studies [10, 8, 9], and our personal software development experiences, we synthesized a set of friction points software developers face when working collaboratively. This is not a comprehensive list of all possible areas of conflict, but rather a curated list focusing on common collaboration friction points.

We classify and present three categories of collaborative friction points inspired by the model of socio-technical systems [11]. Figure 1 shows a simplified view of the relationships between the systems. For each category, we found various chatbots being used by software teams, which we believe highlights the growing challenges in modern collaborative work environments.

Collaboration Friction Points in Software Development

Team Interactions
- ↯ Understanding team members' roles and expertise
- ↯ Adhering to team procedures and agreements
- ↯ Understanding and working towards team goals
- ↯ Coordinating team activities
- ↯ Managing trust and team cooperation

Individuals ⇔ *Technology*
- ↯ Distracting and interruptive technologies
- ↯ Maintaining awareness of new technologies
- ↯ Understanding channel affordances

Teams ⇔ *Technology*
- ↯ Information fragmentation and overload
- ↯ Adopting and understanding tool usage in the team's context
- ↯ Maintaining awareness of project activities
- ↯ Inadequate collaboration tooling
- ↯ Miscommunication on text-based channels

We conclude the paper with a set of research questions to spark discussion on the future use of chatbots in software development.

Background

We define a collaboration friction point (↯) as any resistance or conflict arising from a team's joint processes. These friction points may stem from the developers' mandatory collaborative needs not being fully satisfied.

Since friction can occur at various levels in a team's collaborative activities, we explored Whitworth's Model of Socio-Technical Systems [11] as a way to classify the types of friction developers experience. A socio-technical system is a complex social system evolving around a technical base, which includes the interplay of human, social, environmental, and technological factors. The lowest level contains the **hardware system**, or the collection of *physical* parts. The **software system** emerges from the hardware system and is based on *information*, the exchange of data, and code. With the addition of *personal* exchanges between the software and a human, the **human-computer interaction (HCI) system** is activated. The **socio-technical (ST) system**, the highest level, emerges from the HCI system with the addition of *community* exchanges.

We expanded the Model of Socio-Technical Systems (Figure 2) to encapsulate the societal and team social systems. We identified that collaboration breakdowns can occur at three levels: ↯ **team interactions** (team's social system), ↯ **individuals' interactions with technology** (HCI system), and ↯ **the team's interactions with technology** (ST system). The sidebar presents a high-level summary of the friction points, classified by their interaction level.

In the following sections, we detail the levels of collaborative friction experienced by developers, summarize select collaboration friction points, and then suggest popular `chatbots` that we believe can help mitigate the detrimental effects of this friction.

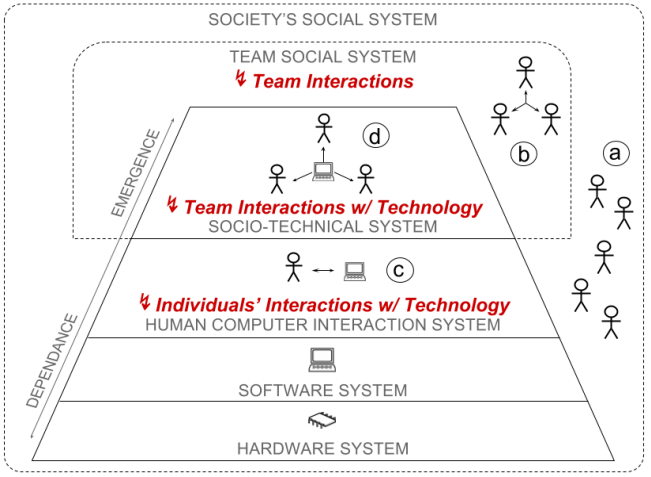

Figure 2: Socio-Technical Model for Collaborative Software Development including the (a) society's social system, (b) team's social system, and 3 categories of friction points: (b) ↯ team's interactions, (c) ↯ individual's interactions with technology, and (d) ↯ team's interactions with technology.

Friction in Team Interactions

This type of collaborative friction occurs as a result of interactions within the team's social system as they work together to realize and achieve shared goals. Although technology may facilitate these interactions, the friction stems from collaborative processes and would occur regardless of the use of technology.

↯ ***Understanding team members' roles and expertise.*** Understanding team members' roles and responsibilities can be difficult in large or distributed teams. Developers

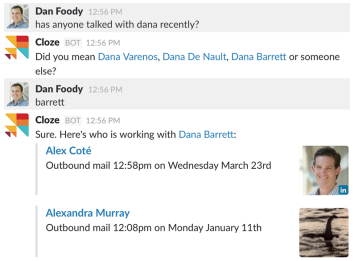

Figure 3: `Cloze` slackbot. Photo: https://www.cloze.com/

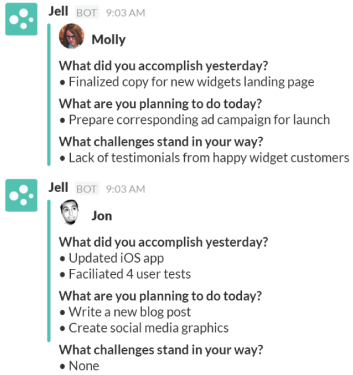

Figure 4: `Jell` slackbot. Photo: https://jell.com/

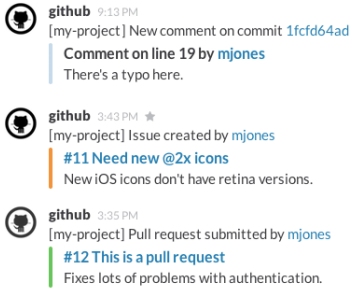

Figure 5: `Github` slackbot. Photo: https://twitter.com/SlackHQ/status/448572066808090624

often experience difficulty locating the best team member for mentorship [7], assistance on tasks requiring special expertise [6], or help when experience with certain aspects of the code base is needed, e.g., bug fixes [2].

To help maintain awareness of teammates' roles, developers use `WhoBot` and `Cloze`. Asking `WhoBot` "who knows about ..." returns a list of team members that frequently mentioned the topic, and `Cloze` consolidates everything you need to know about your contacts in one place: background information, a summary of interactions, and any follow-up items or notes.

Adhering to team's procedures and agreements.
Scripts, often referred to as working agreements, are a set of rules or guidelines that teams agree to follow to make themselves—and the team as a whole—more efficient [3]. Team members often forget or are unaware of the correct sequence of steps when completing tasks, particularly new members joining the team [3].

A development team at a large software company uses `Slackbot` to answer users' queries regarding the correct procedures for development activities, such as merging feature branches into the main code base. They also use `Slackbot` to monitor system status (e.g., website outages), notify the team, and provide a set of next steps.

Understanding and working towards team goals.
To collaborate successfully, all team members must understand the team's goals and the actions needed to achieve them [3]. They must also maintain awareness of the project's overall status by communicating what they are working on, their progress, and blockers they face [3].

To help the team maintain awareness of each other's activities, developers use chatbots like `WorkingOn` to broadcast real-time status updates to the group, and `Jell` to share their accomplishments, ongoing tasks, and blockers.

Coordinating team activities.
For large or distributed teams, coordinating multiple meetings, deadlines, and other activities can be a complex task. However, it can be even more difficult to ensure everyone is made aware of and remembers activities.

To help find a meeting time that works for everyone, developers turn to `Meekan`, a chatbot that searches through everyone's calendars, returns the times with the least conflicts, and allows team members to vote on their preferred options. Before a meeting, the `Solid` chatbot sends detailed reminders to team members. During the meeting, `Solid` keeps track of the time remaining and any incomplete tasks. After the meeting, `Solid` sends attendees the meeting's outcomes and any tasks they were assigned.

Managing trust and team cooperation.
Collaborating with others is difficult; some team members may have poor attitudes or lack the willingness to participate in a collaborative manner [10], while others may take a while to warm up to and begin trusting their teammates [1].

`Oskar` asks and shares how individuals are feeling to prevent isolation and allow teammates to offer support. To monitor team morale, `Ava Bot` privately checks in with team members to see how things are going and raises issues to management when appropriate.

Friction in Interactions with Technology

Two categories of friction points stem directly from technologies not adequately supporting software developers' collaboration needs: friction resulting from **individual** and **team** interactions with technology.

Glossary of Chatbots

`Ava Bot`
http://zeal.technology
`BitBucket`
https://slack.com/apps/A0F7VRDPE-bitbucket
`Convergely`
https://www.convergely.com
`Cloze`
https://www.cloze.com/app/connect/slack
`Digest.ai`
https://digest.ai/
`Github`
https://github.com/integrations/slack
`WhoBot`, `T-Bot`
https://www.onmsft.com/news/microsoft-teams-introduces-t-bot-and-who-bot
`Jell`
https://jell.com/slack
`Knelfbot`
http://www.knelf.com/slack.html
`Meekan`
https://meekan.com
`MyVoiceBot`
http://demo.softserveinc.com/
`Oskar`
http://oskar.hanno.co
`Screenfully`
http://screenful.com/guide/slack-integration
`SlackBot`
https://slack.com/apps/A0F81R8ET-slackbot
`Solid`
https://getsolid.io/slackbot
`Subversion`
https://slack.com/apps/A0F827LTA-subversion
`WorkingOn`
https://www.workingon.co/integrations/slack

Friction resulting from **individuals' interactions** with technology arises when an *individual's needs* are not fulfilled by the technologies they are using, which impedes their ability to collaborate effectively. Although an overview of these friction points is provided in the sidebar, for the sake of brevity, we focus on team interactions with technology.

Friction resulting from **team interactions** with technology arises when the *community's needs* are not fully satisfied by the technologies they are using.

Information overload.
Attempting to enable developers to work more productively, teams often adopt more tools in their workflow, resulting in information and channel overload [10]. As the team's knowledge gets spread over the growing number of channels, issues with information fragmentation and quality begin to emerge [10].

To deal with this ever growing flow of knowledge artifacts, developers use chatbot integrations for many of their everyday tools [8, 4, 9]. Chatbots also provide them curated overviews of channels they may not be actively following: `Digest.ai` creates daily recaps of team discussions, and `TLDR` generates summaries for long messages.

Adopting & understanding tool usage in the team's context.
Friction occurs when team members refuse to adopt the tools required for their job, which is often due to a lack of technical knowledge[5]. Even with successful adoption, team members still need to understand how to use these tools within their team's social context.

To help teams learn to use communication tools, `Slackbot` and `T-Bot` teach teammates how to preform common actions, such as creating a new channel. Developers also use chatbots to "bring technology into the conversation" and make complex tools or tasks (e.g., development operations) accessible to the entire team though text-based commands in their communication platforms[1].

Maintaining awareness of the team's technology-dependent activities.
With the surge of social development tools, developers must maintain awareness and coordinate their development activities with those of their teammates [10]. For successful collaboration to occur, they need to understand how to use the tools within the context of others using them as well.

Chatbot-style integrations for existing tools like `GitHub`, `BitBucket`, and `Subversion` notify the team when changes are made to the codebase, helping developers maintain awareness of the team's collaborative activities.

Discussion Points for the Workshop

We believe that chatbots have immense potential for supporting developers' collaboration needs. However, we first need to understand the benefits and possible risks these new, virtual teammates are bringing to the software development teams that are so openly embracing them.

To conclude, we propose a set of research questions to spark discussion on the future of chatbots and their ability to facilitate collaboration in software development:

1. How should we study chatbots? Can existing models and theories of collaboration help explain how chatbots are being used?

2. What other collaborative friction points can be addressed with new or existing chatbots?

[1] https://youtu.be/IhzxnY7FIvg

3. With rapid progress being made in the fields of artificial intelligence, machine learning, and speech interfaces, how might this change the use of chatbots in the future?

4. What risks are introduced by adopting chatbots in software development?

References

[1] B. Al-Ani, M.J. Bietz, Y. Wang, E. Trainer, Be. Koehne, S. Marczak, D. Redmiles, and R. Prikladnicki. 2013. Globally Distributed System Developers: Their Trust Expectations and Processes. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*. ACM, New York, NY, USA, 563–574. DOI: http://dx.doi.org/10.1145/2441776.2441840

[2] J. Anvik, L. Hiew, and G.C. Murphy. 2006. Who Should Fix This Bug?. In *Proceedings of the 28th International Conference on Software Engineering*. ACM, New York, NY, USA, 361–370. DOI: http://dx.doi.org/10.1145/1134285.1134336

[3] M. Arciniegas-Mendez, A. Zagalsky, M.A. Storey, and A.F. Hadwin. 2017. Using the Model of Regulation to Understand Software Development Collaboration Practices and Tool Support. In *Proceedings of 20th ACM Conference on Computer-Supported Cooperative Work and Social Computing*. IEEE, Portland, OR, USA, 17. DOI: http://dx.doi.org/10.1145/2998181.2998360

[4] F. Calefato and F. Lanubile. 2016. A Hub-and-Spoke Model for Tool Integration in Distributed Development. In *Proceedings of International Conference on Global Software Engineering*.

[5] E. Kalliamvakou, D. Damian, K. Blincoe, L. Singer, and D.M. German. 2015. Open Source-style Collaborative Development Practices in Commercial Projects Using GitHub. In *Proceedings of the 37th International Conference on Software Engineering - Vol 1*. IEEE Press, Piscataway, NJ, USA, 574–585. http://dl.acm.org/citation.cfm?id=2818754.2818825

[6] A. Moraes, E. Silva, C. Da Trindade, Y. Barbosa, and S. Meira. 2010. Recommending Experts Using Communication History. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*. ACM, New York, NY, USA, 41–45. DOI: http://dx.doi.org/10.1145/1808920.1808929

[7] I. Steinmacher, I.S. Wiese, and M.A. Gerosa. 2012. Recommending Mentors to Software Project Newcomers. In *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering*. IEEE Press, Piscataway, NJ, USA, 63–67. http://dl.acm.org/citation.cfm?id=2666719.2666734

[8] M.A. Storey. 2012. The Evolution of the Social Programmer. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*. IEEE Press, Piscataway, NJ, USA, 140–140. http://dl.acm.org/citation.cfm?id=2664446.2664469

[9] M.A. Storey and A. Zagalsky. 2016. Disrupting Developer Productivity One Bot at a Time. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, New York, NY, USA, 928–931. DOI: http://dx.doi.org/10.1145/2950290.2983989

[10] M.A. Storey, A. Zagalsky, F. Filho, L. Singer, and D. German. 2016. How Social and Communication Channels Shape and Challenge a Participatory Culture in Software Development. *IEEE Transactions on Soft. Eng.* PP, 99 (2016), 1–1. DOI: http://dx.doi.org/10.1109/TSE.2016.2584053

[11] B. Whitworth. 2009. *The social requirements of technical systems*. Vol. 3. IGI Global.