Disrupting Developer Productivity One Bot at a Time

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

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

ABSTRACT

Bots are used to support different software development activities, from automating repetitive tasks to bridging knowledge and communication gaps in software teams. We anticipate the use of Bots will increase and lead to improvements in software quality and developer and team productivity, but what if the disruptive effect is not what we expect? Our goal in this paper is to provoke and inspire researchers to study the impact (positive and negative) of Bots on software development. We outline the modern Bot landscape and use examples to describe the common roles Bots occupy in software teams. We propose a preliminary cognitive support framework that can be used to understand these roles and to reflect on the impact of Bots in software development on productivity. Finally, we consider challenges that Bots may bring and propose some directions for future research.

CCS Concepts

•Human-centered computing → Interaction paradigms; Collaborative content creation;

Keywords

Human computer interaction; computer supported collaborative work; productivity; software engineering

1. INTRODUCTION

Improving the productivity and effectiveness of developers is a key concern faced by practitioners and researchers alike. One way to help developers be more productive and effective is to provide them with better and smarter tools—tools that automate or streamline the development process—so that they can work together on larger and more complex systems in a more efficient manner. We now see Bots—also referred to as ChatBots or ChatOps—playing a prominent role in many software development contexts. In general, bots are seen as applications that automate repetitive or predefined tasks. In software development they are used to help developers make smarter decisions and to support developers that need to communicate and coordinate with others. The micro-services that Bots provide are not new, but the way they are presented to developers, through a conversational UI embedded in developer chat channels is changing how tools are integrated in the developer's tool suite.

In their basic form, Bots serve as a conduit or an interface between *users* and *services*, typically through a conversational user interface (UI)[1], and are further enhanced by adding personalization and a memory. They can be designed to operate in *pull* mode where the user initiates the interaction, *push* mode where the Bot initiates the interaction, or a combination of both. Most commonly Bots are used for automating tasks (e.g., running tests when certain conditions are met) or for gluing tools together. Bots may leverage AI or machine learning techniques, or they may capture or analyze data generated by other tools and Bots.

Outside of software development, Bots and conversational UIs are seeing mainstream adoption. Facebook, Amazon, Microsoft, and Google[2] are all investing heavily in what is hailed by some[3] as the new *"Universal UI"* and seen by many as a *paradigm shift* for user interaction that is more natural for humans to use. Some companies use Bots for extending services to better provide for the user's needs (e.g., Google's conversational search assistant), while others, e.g., Facebook, aim to replace apps *"one bot a time"*. Bots are seeing adoption across many different industries from retail and ecommerce to government.[4]

Software engineering has already seen an adoption of Bots at a dizzying pace. Bots reportedly help developers become more productive by automating tedious tasks, by helping developers stay aware of important project or community activities, and by reducing interruptions. At the team level, Bots smooth and improve the efficiency of every phase of the software development life cycle, including coding, testing, operations, and managing user relations. Team communication tools such as IRC, Slack, and HipChat offer platforms for integrating services through Bots. Developers communicate and "listen to" these Bots in the same style as the

FSE'16, November 13–18, 2016, Seattle, WA, USA
© 2016 ACM. 978-1-4503-4218-6/16/11...$15.00
http://dx.doi.org/10.1145/2950290.2983989

[1] http://www.wired.com/2015/06/future-ui-design-old-school-text-messages/
[2] http://www.macworld.co.uk/feature/iosapps/cortana-vs-siri-google-now-amazon-echo-alexa-what-is-best-ai-voice-assistant-3511811/
[3] http://techcrunch.com/2016/02/16/on-chatbots/
[4] https://medium.com/botness/learnings-from-the-first-botness-survey-dbeba3f89fbc?source=latest

conversational UI they use to collaborate and monitor other developers on their teams! Yet little is known about the impact these Bots have in terms of the benefits they bring to individual or team productivity, or in improving quality. Moreover, the challenges and risks that may arise from developers and teams using these "new virtual team members" has barely been considered.

In this paper, we aim to shed light on the prominent role that Bots are starting to play and encourage researchers to study the impact (positive and negative) of Bots on software development. We propose a *cognitive support framework for how Bots can support software development.* The framework first describes some of the more common **Bot roles** that support different phases of the software development life cycle as well as how Bots can help developers be more **efficient** and more **effective** in meeting their goals. We suggest that this framework can help in describing, designing and evaluating development Bots. This is followed by a discussion of the **challenges** and risks that Bots may spawn as well as some important **future research directions**.

2. DEVELOPER BOT ROLES

Like the many roles software developers can fill, we see a variety of Bots participating in every phase of the software development process. While Slack and HipChat are two popular services that provide a Bot habitation platform, Bots are available through many different tools. In the following, we describe some of the main categories of Bots we see in software development. Some of these categories are inspired by Sven Peters' talk about Bots in the Atlassian tool ecosystem[5], while other categories stem from our preliminary research on this topic as well as a brief survey we conducted with developers that use Bots integrated through Slack [3]. Using these categories, we give examples of Bots that fill or may soon fill these roles.

2.1 Code Bots

There are a variety of Bots that help make coding activities more efficient and effective. One challenge that developers face during development is that they often have to *synchronize tasks* across two or more separate tool workflows. For example, committing code changes and fixing bugs on GitHub, while also updating work items and tasks on Trello. Doing all the required steps is tedious and developers frequently forget to update both tools, but a Bot (e.g., Hubot) can be programmed to automate some of these steps and offload some of the memory overhead.

BugBot is a Slack app for working with GitHub issues, allowing users to maintain awareness and create bug reports through Slack[6]. BugBot *automates* previously manual tasks and *integrates* tools developers use. Tools like Slack can be integrated with issue tracking systems such as GitHub and Trac, and through these integrations, developers can *gain awareness* of commits without necessarily being interrupted, potentially improving their productivity as they can stay in the "flow" [4]. Developers also create custom Bots for supporting the peer review process in their team[7], allowing to automatically check GitHub Pull Requests (PRs) for peer reviews, and label and merge these PRs appropriately.

To help developers find answers to questions, Slack can be integrated with Stack Overflow.[8] It contributes to the team's collective knowledge by allowing developers to ask questions in the same channels used for team communication. This can be further improved by Bots like Brisby, a knowledge management Bot that automatically answers questions by learning from responses to similar questions previously discussed on the team.

Most of the Bots mentioned so far are rather rudimentary and do only what a developer has programmed them to do. Perhaps they don't really make the developer any *smarter* or more effective—they simply remove friction. But there are more sophisticated Bots, such as those in the Atlassian tool chain that watch what happens when a developer breaks a build and then automatically create a branch from that build. Atlassian Bots can also merge changes across different branches and recommend reviewers based on previous commits. In the future, one can imagine Bots that take more sophisticated actions based on the code context [1].

2.2 Test Bots

Bots also play an integral role in testing. Many static code analysis tools, that had their roots in research but were cumbersome to adopt, are now more accessible through a Bot. In the Atlassian ecosystem, the Freud Bot automatically runs static analysis tools such as FindBugs, CheckStyle, and PMD. If one of these tools indicates a code quality concern, Freud generates a Pull Request with the corresponding suggested code change. This not only saves on code review time but also provides constructive suggestions for improving the code. Another Atlassian Bot, Dr. Code, keeps an eye on "technical debt" by tracking and visualizing project health over time. Atlassian also has a Compare Bot that notices changes in screenshot images, indicating potential user interface bugs. When a new version of a UI is approved, the Compare Bot uses this new version as its oracle.

In terms of saving time, Atlassian uses a bot to detect flaky tests—tests that fail on occasion and make developers consume valuable resources trying to figure out the reasons. This bot quarantines flaky tests and allows the build to continue, but creates an issue so that the quarantined test can be investigated by developers at a later time. Another Atlassian Bot, Hallelujah, saves time during testing by balancing tests across machines. For the NPM ecosystem, the Greenkeeper.io Bot generates a pull request if a new dependency update for a library or API requires a test to be updated.

2.3 DevOps Bots

DevOps Bots (often called ChatOps because of the integration in Chat) are used to speed up code deployment or address slow feedback loops between developers, infrastructure and operations personnel. For instance, PagerDuty is a tool that automatically creates an issue when a service fails, notifying the right people and reducing "alert noise" and communication overload (as the correct personnel are specified in the description of the service). Pagerbot[9], a bot developed at Stripe, allows team members to easily manage and coordinate their PagerDuty on-call schedules and incident response efforts. In another example, Jason Hand

[5]https://svenpet.com/talks/rise-of-the-machines-automate-your-development/
[6]http://smallwins.today/bugbot
[7]www.felixrieseberg.com/a-peer-review-bot-for-github/
[8]https://github.com/karan/slack-overflow
[9]https://stripe.com/blog/pagerbot

discusses how MTTR (mean time to repair) can be reduced through the use of feedback loops mediated by Bots[10].

Teams also use Bots (e.g., DeployBot[11]) to build, manage, deploy, and monitor their build deployments directly from their chosen communication tool (e.g., Slack, Campfire, Hipchat). *"Chatting with your infrastructure might seem strange at first but it's easy to see the benefits. A timeline of who's deploying what and deployments that are so easy anyone can trigger them."*[12] Allowing any stakeholder to deploy helps reduce the feedback loop and bridges the technical-knowledge gap for many stakeholders on the team.

2.4 Support Bots

Support Bots help bridge the communication gap that often exists between users and developers. One key challenge for developers is dealing with messages or reports from an extensive user base. Bots can automate interactions with users and customers by answering frequently asked questions by consulting (and updating) a knowledge base.

Customer support services like ZenDesk, Intercom, and Smooch all have integrations (a basic version of Bots) with tools like Slack, allowing for and assisting with direct communication with users, capturing customer feedback, and in some cases, automatically providing suggested answers. Over time, Bots might reduce team communication overload (e.g., filtering non-helpful reviews, assigning the right people, or suggesting solutions without involving humans at all), while providing better support and a much improved user experience. Similar Bots are used in other domains. For example, with MOOCs in the education space, Bots can be used to mimic the role of a teaching assistant to answer frequently asked student questions.[13]

2.5 Documenting Bots

Documentation authorship is always a challenge for developers. For example, authoring release notes for a new version can be quite tedious. In the Atlassian ecosystem, Bots author release notes by aggregating information from code commits and issue comments. A side benefit is that developers are likely to write better messages knowing that they will be used for release notes. Documentation could also be automatically generated in different languages through the use of Translation Bots.[14] We also suspect that future Bots will automate the generation of documentation from resources such as Stack Overflow, or generate reports and dashboards by integrating analytics and visualization services.

3. HOW BOTS CAN IMPROVE DEVELOPER PRODUCTIVITY

The categorization of Bots by Role is useful in understanding the landscape of Bots for supporting software development, but it does not help one reflect on how Bots benefit developer productivity and project quality. Recently, Meyer *et al.* studied developers' perceptions of productivity [4], and found that individual developers have very different views on productivity but completing tasks and having clear goals, reducing interruptions and distractions, and holding fewer meetings are factors that make developers feel more productive. Their recommendations are that: individuals should avoid interruptions and set clear goals; teams should use toolchains that support flow; and organizations should streamline their communication. Interestingly, Bots can be used to meet many of these recommendations.

Productivity is a concept of interest in other domains[15] and in general it is important to distinguish *effectiveness* (completing tasks related to meaningful goals) from *efficiency* (completing tasks more quickly). Here we suggest a number of ways that Bots can provide cognitive support for improving efficiency and effectiveness of developers through a set of concrete **design elements** (shown in bold) that are clustered according to the *user goals* that they satisfy (shown in italics).

3.1 Efficiency (do things faster)

Automate tasks:

Many Bots are used to **automate tedious tasks** (e.g., comparing different user interface screens after a change), or to **automate repetitive tasks** (such as merging changes across different branches), or Bots can **make tasks redundant** (e.g., by automatically answering user questions).

Help developers stay in flow:

It is important for developers to maintain a state of "flow" especially when programming. One way is to **reduce interruptions and distractions** whereby Bots can be aware of the developer's context and defer interruptions and notifications until a more suitable time [1]. Some interruptions are unavoidable, but Bots should **provide support for context switching**. For example, a Bot could save the developer's state. Another way to help developers stay in flow is to **integrate tools** reducing friction from tool switching or by gathering information that was formerly scattered across various tools.

3.2 Effectiveness (towards meaningful goals)

Improve decision making:

In addition to supporting faster completion of tasks, Bots can help in decision making by **capturing and analyzing data relevant to decisions**. For example, capturing information about user requirements and insights on suitable reviewers can improve design and code reviewing activities. Just as it is important to capture data, it is also important to **share knowledge** with other members, and Bots can help with information dissemination (e.g., documentation generation from release notes).

Support team cognition:

Mary Poppendieck [6] discusses how the "team with the most situational awareness wins", thus Bots that **provide situational awareness** will help teams be more effective. Many Bots already help team members know when commits were made, which tests and services failed, and when other services or builds are deployed. But Bots are also needed to **support team communication**, either by initiating it when it is needed, or by making it unnecessary.

[10] www.jasonhand.com/infrastructure-as-conversation/
[11] https://deploybot.com/
[12] https://skillsmatter.com/skillscasts/7629-devops-for-slackers-deploying-code-with-a-chat-bot
[13] http://www.wsj.com/articles/if-your-teacher-sounds-like-a-robot-you-might-be-on-to-something-1462546621
[14] http://botsfortelegram.com/project/translate-bot/
[15] http://www.productiveflourishing.com/a-general-theory-of-productivity/

Regulate individual and team tasks and goals:
Productive and effective developers and teams will carefully regulate their goals and tasks, and monitor and visualize their own and team activities [2]. Bots can **initiate and track reminders**, as well **support coordination across tasks**, help to **monitor and visualize progress and team culture**, and then even possibly **adapt** when things are not as expected.

4. DISCUSSION AND FUTURE WORK

Developers use many different Bots and with fully programmable Bots like Hubot and Lita, and frameworks such as BeepBoop and Microsoft's Bot Framework, developers can customize and build Bots that fit their needs. We already see that some of the challenges developers face when using social and communication channels are reduced by the use of Bots (e.g., communication overload, information fragmentation, maintaining team awareness) [8].

Some may correctly point out that the technical support offered by Bots is not new. Plugins, scripts, and architectures that support micro-services have been around for some time. But what is new, as Sean Regan from Atlassian mentions, is that the integration of micro-services in a conversational UI leads to a *"collaboration model that connects developers, tools, processes and automation into a transparent workflow"*. And as the role of conversational UIs is further combined with powerful AI techniques, access to big data, and natural language processing, the use of Bots is likely to increase. The usage scenarios presented in this paper are probably only the tip of the iceberg. The anticipation is that Bots will lead to an improvement in software quality and developer and team productivity. But will this positive impact address only what Fred Brooks calls "accidental complexities" rather than the "essential complexities"? And if Bots do disrupt developer productivity, what if that disruption has negative consequences in addition to certain benefits?

Indeed, there are many possible **negative consequences**. Bots may cause team members to spend less time together, reducing the chances for serendipitous learning and discovery. Another potential issue revolves around acquired culture— some Bots can acquire and embody company culture, but what happens when that culture changes? Will the sentiment analysis that Bots use be sensitive to culture and personal nuances? Bots are also often used to avoid interruptions or distractions, but they may bring other interruptions or distractions that are not as obvious initially. Also, automatically generating documentation or release notes may be desirable, but if a programmer knows they are generated, will they be as likely to read and trust them? [5] If some Bots are not obviously "machines", are there ethical issues that should be considered? In sum, it is not clear **how Bots should be designed**. There are guides on how to program Bots that suggest best practices[16] and discuss why personality matters[17] but these best practices do not address ethical, social, or long-term impacts of Bot usage.

How to study Bots is furthermore unclear. Through our suggested preliminary framework, we categorized Bots according to their role, but as Reinhardt discusses there are other knowledge worker roles [7] such as Controller, Linker, Organizer and Sharer, that Bots may play a role in filling. Our framework also suggests cognitive support design elements for enhancing developer and team productivity. But productivity may not be the only goal. Keeping developers happy may be critical in terms of retention. Interestingly, there are Bots, such as Oskar[18], that can track user happiness. Perhaps, in the future Bots can be used to study the impact of Bots and then self adapt to avoid negative consequences. But first we need to be able to anticipate and recognize the benefits and possible negative consequences if Bots are to satisfy our intended human based goals.

[16]https://medium.com/slack-developer-blog/slack-bot-onboarding-3b4c979de374#.wzlfz6jr9

[17]http://venturebeat.com/2016/07/07/bots-need-a-personality-not-a-brain-like-a-vending-machine/

[18]http://oskar.hanno.co/

5. ACKNOWLEDGMENTS

We thank Chris Parnin for his feedback and insightful comments that contributed to this work.

6. REFERENCES

[1] M. P. Acharya, C. Parnin, N. A. Kraft, A. Dagnino, and X. Qu. Code drones. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pages 785–788, New York, NY, USA, 2016. ACM.

[2] M. Arciniegas-Mendez, A. Zagalsky, M.-A. Storey, and A. F. Hadwin. Using the model of regulation to understand software development collaboration practices and tool support. In *Proceedings of the 20th ACM Conference on Computer Supported Cooperative Work & Social Computing (to appear)*. ACM, 2017.

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

[4] A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann. Software developers' perceptions of productivity. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 19–29. ACM, 2014.

[5] A. Murgia, D. Janssens, S. Demeyer, and B. Vasilescu. Among the machines: Human-bot interaction on social q&a websites. In *CHI Conference on Human Factors in Computing Systems*, CHI Extended Abstracts, pages 1272–1279. ACM, 2016.

[6] M. Poppendieck and T. Poppendieck. *Lean Software Development: An Agile Toolkit: An Agile Toolkit*. Addison-Wesley, 2003.

[7] W. Reinhardt, B. Schmidt, P. Sloep, and H. Drachsler. Knowledge worker roles and actions – results of two empirical studies. *Knowledge and Process Management*, 18(3):150–174, 2011.

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