2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)

A Chatbot for Conflict Detection and Resolution

Elahe Paikari
University of California, Irvine
Irvine, CA USA
epaikari@ics.uci.edu

JaeEun Choi
KyungHee University
Seoul, Republic of Korea
mody123@khu.ac.kr

SeonKyu Kim
Kookmin University
Seoul, Republic of Korea
kimsk9410@kookmin.ac.kr

Sooyoung Baek
Hanyang University
Seoul, Republic of Korea
qortndud97@hanyang.ac.kr

MyeongSoo Kim, SeungEon Lee, ChaeYeon Han, YoungJae Kim
Kookmin University
Seoul, Republic of Korea
{myron89, js03110, cyhan1004, corea_kyj}@kookmin.ac.kr

KaHye Ahn
Sogang University
Seoul, Republic of Korea
sghk@sogang.ac.kr

Chan Cheong
Dankook University
Seoul, Republic of Korea
j94chan@gmail.com

André van der Hoek
University of California, Irvine
Irvine, CA USA
andre@ics.uci.edu

***Abstract*—We present Sayme, a chatbot that we are developing to address the detection and resolution of potential code conflicts that may arise in parallel software development. Sayme is designed to operate both proactively, informing developers when they engage in activities that create conflicting changes, and reactively, responding to user inquiries regarding the state of different developers' work and how it may overlap. We introduce our motivation for developing Sayme, present its design and features, and offer an outlook at our future work.**

Keywords—Chatbots; code conflicts; collaborative development; configuration management; awareness

I. INTRODUCTION

In any software development project, developers are faced with potential problems when they work on the same code base in parallel. It is well-known that conflicts may occur when they change the same artifact simultaneously or when they change different artifacts, but those artifacts exhibit dependencies (e.g., [1]). The former is typically called a direct conflict, the latter an indirect conflict [2].

While the typical software configuration management system provides features for either avoiding conflicts (by locking the files a developer plans to work on) or resolving them when they do occur (through merge tools), it has been observed that these features are insufficient: some 10%-20% of all check-ins (or pull requests in more modern terminology) lead to a conflict [3] and developers report wanting to avoid these at all costs [4, 5]. In response, various tools have been developed that offer awareness of parallel work by informing developers of potential conflicts in real-time, so that they can adjust, either autonomously or by coordinating with the other developer involved [6, 7].

To date, nearly all these tools provide visual information concerning conflicts, either directly presented in the integrated development environment or appearing in a pop-up window. In this paper we explore a different direction, namely one relying on integrating the information regarding conflicts in the developer's chat channels, for two reasons:

- With the prolific rise of Slack and other social channels for communication in software engineering [8], we believe it is important to explore whether alerting software developers of conflicts through this familiar channel could potentially be more effective than the typical visualization in the IDE. That is, does the medium matter from the perspective of the developer and overall project?
- While today's chatbots in software engineering are limited in their functionality [9], more advanced chatbots in other fields have begun to show the potential power of conversation (e.g., [10, 11]). We believe conflict detection and resolution represent excellent targets for exploring if chatbot can be intelligent participants in the conversations among developers when they are faced with conflicts.

In this paper we present our first steps into this exploration. Specifically, we introduce our prototype, Sayme, which detects direct and indirect code conflicts, notifies developers of emerging conflicts, and responds to a variety of requests to help developers understand the state of other developers' work.

II. BACKGROUND

A. Conflict Detection Tools

A host of different tools have been implemented to enhance developers' awareness of parallel changes and especially potential conflicts among those changes. These tools monitor all the changes developers make in their local workspaces and, upon detecting potential conflicts among those changes, send notifications to the involved developers. The kinds of conflicts they address has evolved over time, with early tools such as Elvin [12] and Syde [13] supporting direct conflicts only, later tools such as Palantír [7] and CollabVS [14] supporting indirect conflicts as well, and a recent tool such as Crystal [15] taking a different tact in distinguishing textual, build, and test conflicts.

How these tools inform developers is equally varied. Elvin uses a uni-directional tickertape, Palantír small iconic annotations in the file viewer of the integrated development environment, and CollabVS a separate window entirely. FastDash [6] is different, providing a holistic view of the entire shared workspace. Lighthouse [16] expands on this idea through spheres of influence that show the potential impact of ongoing changes.

While these tools are important, a downside is that they are passive. They pass information about possible code conflicts to the developer, but it is up to the developer to then respond using whatever means they find suitable (e.g., abandon their own changes, use IM to coordinate with the other developer).

B. Chatbots

While both operate on communication (chat) channels such as Slack or Microsoft Teams, chatbots are different from bots. Bots automate repetitive tasks and are invoked or provide output on chat channels [17]. Chatbots also operate on chat channels, but engage in conversation to accomplish their tasks [9]. The use of chatbots has recently become popular for all sorts of purposes

978-1-7281-2262-5/19/$31.00 ©2019 IEEE
DOI 10.1109/BotSE.2019.00016

in all sorts of fields[1], though it should be acknowledged that chatbots have been explored for years [18] and have strong roots in conversational agents [19]. The emergence of Google Assistant[2], Amazon Alexa[3], Slack, and other platforms, together with frameworks such as Hubot[4] and wit.ai[5], has propelled chatbots into everyday design and use.

Recently, the software engineering research community has begun to explore chatbots. For example, Devy [20] enables developers to use voice commands to execute low-level workflows, 'talking back' if it needs further information. As another example, Socio [21] supports a team of developers in creating domain models out of requirements that developers express in natural language on an instant message channel.

Some have begun to study existing chatbots in order to build an understanding of their overall landscape. Wessel et al. [22], for instance, surveyed developers about their projects' use of chatbots on GitHub, finding that, although chatbots are considered useful for assistance in maintenance tasks, they are not sufficiently intelligent to fully support even just these tasks in ways that developers actually would prefer to work. As another example, Lebeuf [17] developed a taxonomy of observable properties and behaviors of chatbots, including details about their structure and implementation. Finally, in our framework [9], we characterize chatbots on underlying design properties, finding that most current chatbots are quite similar in being passive in their conversations or unable to learn from existing interactions.

III. APPROACH

Sayme is a chatbot that provides information about potential direct and indirect conflicts and helps developers in resolving these conflicts. Its primary function is to proactively detect potential conflicts among developers working in parallel on the same set of code artifacts, so it can notify them through Slack channel before emerging conflicts become too big. Sayme's secondary function is to respond to a variety of requests that help developers understand the state of other developers' work. We designed Sayme with several distinguishing factors in mind.

- *Initiative*: A key design decision of Sayme is that it acts proactively as well as reactively. In this way, Sayme 'feels' more like a full partner to developers, given that it offers them useful functionality on its own but also responds to their needs.
- *Knowledge of context*: Sayme maintains a working history of its past conversations, which allows it to adjust its messages relative to past messages. This is useful, for instance, to indicate if a conflict still exists or has been resolved.
- *Private channels*: Sayme uses private channels to talk with each developer in order to avoid information overload and targeted messages becoming lost in a sea of notifications. It also allows Sayme to personalize the message when two or more developers are involved in a conflict.
- *Shorthand*: In some cases, Sayme not only presents information to developers, but also provides them Slack buttons to act on this information. For instance, when Sayme lets a developer know about a conflict, the message is accompanied with a *git diff* button so a developer can inspect it.
- *Alternative vocabulary*: For each functionality, Sayme has on average five different ways of phrasing a notification or response. This is to appear more personable and relatable.

A. Proactive Features

As discussed, direct conflicts may occur when two or more developers work on the same file simultaneously, and indirect conflicts may occur when one developer edits one file in ways that may be incompatible with changes by a second developer to another file (in our case, we scope this to a method invocation being added in one file and changes to that method being made in its actual source code; overall, however, our approach is generalizable to the broader set of conflicts covered by, for instance, Crystal). Note that we explicitly state 'potential' conflicts: awareness tools work by notifying developers of what might be, instead of exactly what is. It is entirely possible, for instance, that two developers make significant changes to a file, but those changes are non-overlapping. It is also possible they make small changes each, but those changes are highly incompatible. Rather than precisely analyzing ongoing changes continuously, which is computationally expensive, awareness tools give developers a 'sense' of what is happening that hopefully is sufficient for them to act if need be.

As an awareness tool, then, Sayme's basic functionality is to inform developers when they are working on the same file, or when one alters a method that another invokes in their code. If this is the case, Sayme issues a message to both developers informing them of the situation. The developers can choose to then act on this information, for instance by abandoning their changes or by deciding to chat with the other developer to find out about their planned changes.

Sayme continues to monitor potential conflicts and will issue another message under two circumstances: if it detects that the potential conflict no longer exists (because one developer reverted their file through GitHub or undid their changes manually in the editor) or after 30 minutes of the conflict being in existence and still not having been resolved. In the former case, the developers are informed the conflict is gone; in the latter they are reminded it is still there.

Consider a scenario in which Sooyoung, Kathryn, and Sun are all adding features to a project. Sooyoung starts modifying file *A.py*, but is informed by Sayme about a potential direct conflict with Sun (left side of Fig1, ①). Sun receives a similar notification (right side of Fig1, ①). Since his changes are already significant, he opens a direct channel with Sooyoung and sends her a message that asks her to undo her work (separate channel not shown). Sooyoung checks her changes and decides that they are pretty minor; she makes a sticky note to do them next time and therefore she chooses to undo her modifications. Sayme notices and informs both of them that the conflict disappeared (Fig1, ②). Since Sun is working on a really complicated task, he does not want to be in the same situation again; he therefore asks Sayme to lock the file for an hour (Fig1, ③). Sayme informs the rest of the team that this file is now locked by Sun to avoid possible conflicts in future (see Fig1, ④⑤).

Meanwhile, Kathryn wants to update *B.py* to insert a call to a method in *A.py.* Careful as she is, she checks with Sayme that she is the only one working on *B.py* (Fig1, ①②). After this,

[1] https://thereisabotforthat.com/

[2] https://assistant.google.com/

[3] https://developer.amazon.com/alexa

[4] https://hubot.github.com/

[5] https://wit.ai/

she opens the file and adds the method call. However, because Sun, in his work on *A.py*, modified the body of this particular method, Sayme flags the potential conflict (Fig1, ③). Sun uses the button to view the diff and decides that his changes are not actually creating a conflict for Kathryn, so he sends her a direct message and confirms that they both can continue (message not shown in figure).

B. Reactive Features

The previous scenario already incorporated two of the reactive features of Sayme: locking a file and checking who may be working on a file. We implemented several other such features where the developer initiates the interaction, with the full list shown in Table 1. Note that we merely include examples of how these features are invoked, as Sayme uses natural language processing to interpret what the developer may mean.

TABLE 1. REACTIVE FEATURES IN SAYME.

Feature	Example
Check working status	What @Sun is working on?
Check file status	Who is editing A.py now?
Lock file	Sayme, could you please lock A.py?
Unlock file	Unlock A.py.
Check lock status	Do you know who locked A.py?
Get code history	Tell me who wrote lines 33-37 in A.py.
Help	Show me your features.
Greetings	Hi. (See you.)

The first two reactive features allow a developer to get information as to what files a particular developer is working on or who is editing a particular file. The next three support locking files, though we note that Sayme does not actually lock the files in GitHub; rather, it treats the locks as soft locks, to stay with its philosophy of being an awareness tool. Thus, if a developer starts editing a file that another developer has locked, Sayme informs the new developer, for instance by telling them that "Sun placed a lock on that file for another hour. I'll let you know when they unlock it." If a developer chooses to ignore it, both are informed of this fact, and they will also start receiving the usual proactive conflict messages. Finally, Sayme supports developers in checking the history of specific lines of code, in learning which features it has, and having an informal greeting mechanism.

IV. IMPLEMENTATION

Sayme is implemented using Python, MySQL, and Google Cloud Platform. It interfaces with Git as the software configuration management system and uses Slack as the chat channel through which it communicates with developers. Its overall architecture is shown in Fig3.

For its proactive features, Sayme operates autonomously by initiating conversations with developers based on information it collects from Git. Specifically, Sayme monitors when developers save files to then behind the scenes use Git commands to automatically extract the files being changed and at which lines they are being changed. It then parses all of the files using the Abstract Syntax Trees (AST)[6] library (to detect possible indirect conflicts across files, as based on incompatible changes to method declarations and invocations). It then stores the repository name, the file names, the number of modified lines of codes, and the actual modified code that each developer is currently working on in a MySQL database. It is through this database that Sayme detects direct and indirect conflicts by issuing queries.

Sayme operates on a request-response basis for its reactive features, waiting for a developer to initiate a conversation. We explicitly chose not to limit Sayme to a fixed set of commands that it would recognize in the Slack channel, instead opting to use natural language processing to interpret the requests of the developer. Sayme uses Dialogflow[7] as the underlying engine, which we seeded with between 28 to 54 training phrases, depending on the specific feature. These training phrases were authored by us to cover common ways in which developers interacted with Sayme during early testing. Upon understanding what feature a developer requests, Sayme again uses the database to either find the information needed (e.g., who is editing a particular file) or store additional information (e.g., a lock being placed). Only in the case of a developer requesting the history of certain lines of code does Sayme invoke GitHub once more.

Note once more that each feature of Sayme is supported by multiple output phrases, some of which are parameterized. This allows Sayme to generate messages with detailed information in it as to who is in conflict, for which file(s), and where.

V. CONCLUSION AND FUTURE WORK

Chatbots are emerging in software engineering as a promising direction of research. Our work seeks to add to the explora-

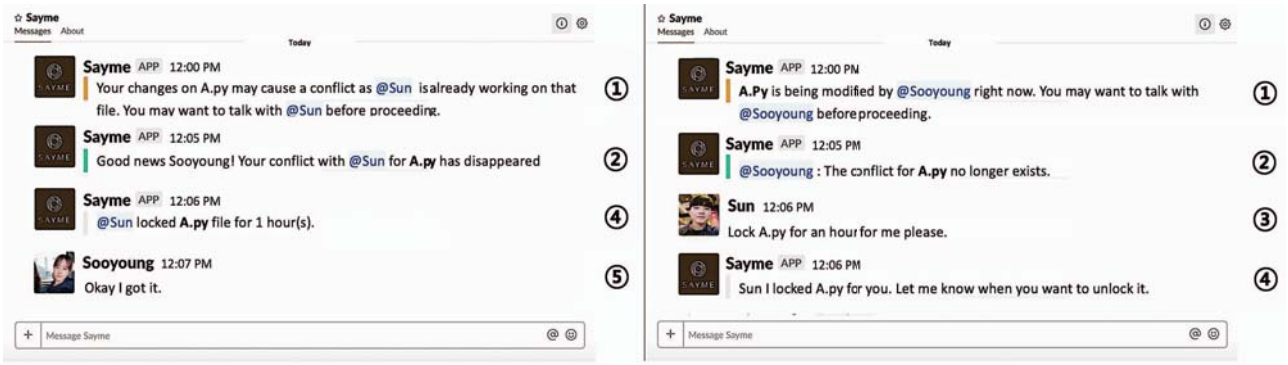
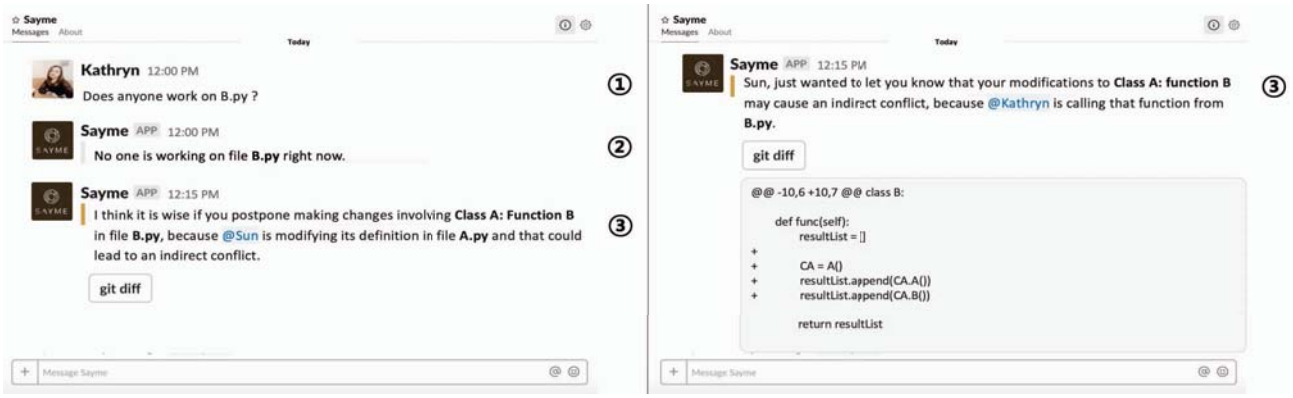


Fig. 1. Example of interaction with Sayme (direct conflict).

[6] https://docs.python.org/3/library/ast.html

[7] https://dialogflow.com/



tion of chatbots in software engineering by focusing on a well-known problem, conflict detection and resolution. We are especially interested in whether using chatbots as an alternative medium to the typical visualization in the IDE can reduce the number of conflicts and, for conflicts that do emerge, help developers in detecting and resolving them earlier. Beyond, however, we want to explore if the conversational nature of chatbots can be leveraged to offer more advanced support yet.

So far, we have implemented a prototype version of Sayme that offers a base layer of functionality, namely the detection of potential direct conflicts as well as a limited form of indirect conflicts. Our next steps are twofold. First, we plan to conduct a comparative laboratory study in which we compare use of Sayme versus a traditional awareness tool (and a combination of both) to examine the relative effectiveness of each approach.

Second, we want to explore the fundamental raison d'être of chatbots: conversations. While Sayme is both proactive and reactive, and in some cases offers buttons for developers to easily respond to its answers, Sayme is not a conversational chatbot like Juji[8] or Hipmunk[9]. We believe that being able to hold such conversations can significantly enhance the value of Sayme. One example is conflict prediction: leveraging the DOI/DOK work [23], we plan to enhance Sayme to predict where possible conflicts may arise, even if the developer has not yet begun editing those files. As this is a prediction, Sayme can ask for confirmation of a developer's plan and then adjust its advice accordingly. As another example, we believe Sayme can be an active participant in conflict resolution, making suggestions based on detailed code analysis and engaging with developers in conversations how to best handle each situation.

ACKNOWLEDGMENT

We thank the I-SURF program for supporting the exchange students that made this project happen.

REFERENCES

[1] S. M. Easterbrook, E. E. Beck, J. S. Goodlet, L. Plowman, M. Sharples, and C. C. Wood, "A Survey of Empirical Studies of Conflict," *CSCW: Cooperation or Conflict?*, Springer, 1993, pp. 1–68.
[2] A. Sarma and A. van der Hoek, "Palantír: coordinating distributed workspaces," 26th Annual International *Computer Software and Applications Conference*, 2002, pp. 1093–1097.
[3] G. G. L. Menezes, L. G. P. Murta, M. O. Barros, and A. van der Hoek, "On the Nature of Merge Conflicts: a Study of 2,731 Open Source Java Projects Hosted by GitHub," *IEEE Transactions on Software Engineering*, pp. 1–1, 2018.
[4] D. Berlin and G. Rooney, *Practical Subversion*. Apress, 2006.
[5] P. M. Duvall, S. Matyas, and A. Glover, *Continuous integration: improving software quality and reducing risk*. Pearson Education, 2007.

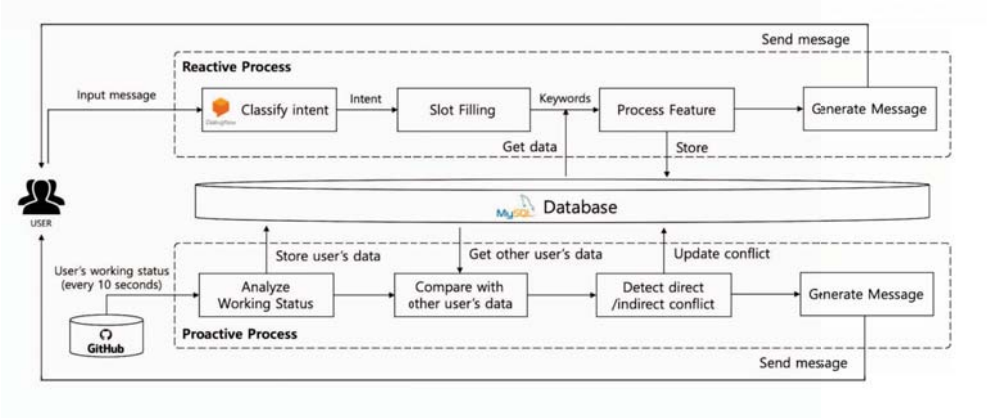


Fig. 3. Architecture of Sayme.

[8] https://juji.io/

[9] https://www.hipmunk.com/hello

[6] J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson, "FASTDash: A Visual Dashboard for Fostering Awareness," *Software Teams - SIGCHI conference on Human Factors in Computing Systems*, 2007, pp. 1313–1322.

[7] A. Sarma, D. F. Redmiles, and A. van der Hoek, "Palantír: Early Detection of Development Conflicts Arising from Parallel Code Changes," *IEEE Transactions on Software Engineering*, 38 (4), pp. 889–908, 2012.

[8] M.-A. Storey, A. Zagalsky, F. F. Filho, L. Singer, and D. M. German, "How Social and Communication Channels Shape and Challenge a Participatory Culture in Software Development," *IEEE Transactions on Software Engineering,* 43 (2), pp. 185–204, 2017.

[9] E. Paikari and A. van der Hoek, "A framework for understanding chatbots and their future," *11th International Workshop on Cooperative and Human Aspects of Software Engineering*, 2018, pp. 13–16.

[10] "A Robot-Companion for Senior People and Patients with Alzheimer's Disease," *EnduranceRobots*. [Online]. Available: http://endurancerobots.com/azbnmaterial/a-robot-companion-for-senior-people-and-patients-with-alzheimer-s-disease/. [Accessed: 05-Feb-2019].

[11] "Roof - AI Assistant for Real Estate," *Roof AI*. [Online]. Available: https://roof.ai. [Accessed: 05-Feb-2019].

[12] G. Fitzpatrick, T. Mansfield, S. Kaplan, D. Arnold, T. Phelps, and B. Segall, "Augmenting the Workaday World with Elvin," Springer, Dordrecht, 1999, pp. 431–450.

[13] L. Hattori and M. Lanza, "Syde: a tool for collaborative software development," *32nd ACM/IEEE International Conference on Software Engineering*, 2010, 2, pp. 235–238.

[14] R. Hegde and P. Dewan, "Connecting Programming Environments to Support Ad-Hoc Collaboration," *23rd IEEE/ACM International Conference on Automated Software Engineering*, 2008, pp. 178–187.

[15] Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin, "Proactive detection of collaboration conflicts," *19th ACM SIGSOFT symposium and 13th European conference on Foundations of software engineering*, 2011, pp. 168–178.

[16] I. A. da Silva, P. H. Chen, C. Van der Westhuizen, R. M. Ripley, and A. van der Hoek, "Lighthouse: coordination through emerging design," *2006 OOPSLA workshop on eclipse technology exchange*, 2006, pp. 11–15.

[17] C. R. Lebeuf, "A Taxonomy of Software Bots: Towards a Deeper Understanding of Software Bot Characteristics," MSc dissertation, 2018.

[18] J. Weizenbaum, "ELIZA—a Computer Program for the Study of Natural Language Communication Between Man and Machine," *Communications of the ACM*, 9 (1), pp. 36–45, 1966.

[19] A. P. Chaves and M. A. Gerosa, "Single or Multiple Conversational Agents?: An Interactional Coherence Comparison," *2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–13.

[20] N. C. Bradley, T. Fritz, and R. Holmes, "Context-aware conversational developer assistants," *40th International Conference on Software Engineering*, 2018, pp. 993–1003.

[21] S. Pérez-Soler, E. Guerra, and J. de Lara, "Collaborative Modeling and Group Decision Making Using Chatbots in Social Networks," *IEEE Software*, 35 (6), pp. 48–54, 2018.

[22] M. Wessel *et al.*, "The Power of Bots: Characterizing and Understanding Bots in OSS Projects," Proceedings of the *ACM on Human-Computer Interaction*, 2018, 2, pp. 1–19.

[23] T. Fritz, J. Ou, G. C. Murphy, and E. Murphy-Hill, "A degree-of-knowledge model to capture source code familiarity," *32nd ACM/IEEE International Conference on Software Engineering*, 2010, 1, pp. 385–394.