2019 ACM/IEEE 14th International Conference on Global Software Engineering (ICGSE)

Supporting A Flexible Grouping Mechanism for Collaborating Engineering Teams

Georgios Kanakis, Stefan Fischer, Djamel Eddine Khelladi and Alexander Egyed
Institute of Software Systems Engineering
Johannes Kepler University
Linz, Austria
georgios.kanakis@jku.at, djamel_eddine.khelladi@jku.at, stefan.fisher@jku.at, alexander.egyed@jku.at

***Abstract*—Most engineering tools do not provide much support for collaborating teams and today's engineering knowledge repositories lack flexibility and are limited. Engineering teams have different needs and their team members have different preferences on how and when to collaborate. These needs may depend on the individual work style, the role an engineer has, and the tasks they have to perform within the collaborating group. However, individual collaboration is insufficient and engineers need to collaborate in groups. This work presents a collaboration framework for collaborating groups capable of providing synchronous and asynchronous mode of collaboration. Additionally, our approach enables engineers to mix these collaboration modes to meet the preferences of individual group members. We evaluate the scalability of this framework using four real life large collaboration projects. These projects were found from GitHub and they were under active development by the time of evaluation. We have tested our approach creating groups of different sizes for each project. The results showed that our approach scales to support every case for the groups created. Additionally, we scouted the literature and discovered studies that support the usefulness of different groups with collaboration styles.**

***Index Terms*—collaboration, collaborating groups, software engineering, change propagation**

I. Introduction

A multitude of engineering tools exist, covering the diverse needs of engineers, during the different phases of the software and systems engineering life cycle. These needs are spread throughout every phase of the software engineering process. There are tools for requirement engineering, architecture & design modeling, implementation, and beyond covering other engineering disciplines [1]. However, nearly all engineering tools are tailored to the needs of individual engineers and rarely to the needs of a collaborating group of engineers. Yet, collaborative engineering is the norm and the group of engineers play a vital role in this engineering process.

The importance of collaboration for software engineering groups has been recognized very early and has been an active research field since the beginnings of the 90's [2]. Collaboration has been described in taxonomies [3], [4] - within the software engineering domain and with other engineering disciplines (e.g., between mechanical engineers and software engineers) [5]. However, today's engineering tools usually focus on individual engineers who capture and handle artifacts locally on their workstation. In doing so, these tools fail to address many collaboration needs: such as the sharing of artifacts or changes at arbitrary times with arbitrary engineers in the same collaborative group.

As a consequence, explicit collaborative tools have emerged to address the lack of tooling support. However, all mainstream approaches, such as Git [6], SVN [7], support a single style of asynchronous group collaboration. They have limited means of sharing artifacts with some engineers (i.e., as desired in feature-driven engineering [8]) or no means of sharing artifacts' changes continuously (i.e., as required by pair programming [9], [10]). A flexible collaboration approach, as we discuss in this paper, could help to overcome these limitations. Furthermore, a systematic mapping study by Portillo Rodriguez et al. showed that there is a lack of tools supporting collaboration in groups with different modes [11].

Our experience, supported by literature [12], suggests that the collaboration among engineers rarely follows limited groups, tools, timing intervals. Moreover, collaborating engineers may use the same tools (e.g., a group of programmers using the same programming tool) or different tools (e.g., a designer and a programmer working together). There are hardly any restrictions on who needs to collaborate with whom, when, and how. Thus, the collaborative groups can apply uniform artifact sharing mode or they may require different modes from each group member. An extension with a grouping mechanism to our previous flexible collaboration framework is proposed and discusses in this paper.

The contributions of this paper are as follows: First, we present scenarios that engineering groups can use to collaborate and propagate their changes. Second, we present a collaboration language that allows engineers to specify their desired collaboration style, both among individual members and in addition to collaborate within their group. Third, we present a change propagation algorithm that enforces the engineers' desired collaboration's parameters in respect to change propagation within groups. Fourth, a cloud-based prototype demonstrates a working implementation. There, the cloud serves as a change propagation facilitator. Lastly, to ensure the principles of the collaboration as defined by the language, we created a model using Alloy. The model verifies that the restrictions and parameters required for the data propagation stand for the applied change propagation algorithm.

We assessed the feasibility and scalability of our approach using four open source projects found on Github. The projects are: Google's SyzKaller, Amazon Web Service Javascript SDK, Facebook's RockSDB and Microsoft Visual Studio. We implemented them using our proposed collaboration language and tested them using their collaboration history (as available through commits). We performed experiments subdividing the engineers of these projects into groups of different sizes. For both collaboration modes, synchronous and asynchronous, we found that the number of engineers in a group had an impact on the time required to propagate all the changes. Furthermore, we compared these propagation times, from collaborating in groups, with scenarios in which engineers collaborate directly with each other without groups. Our experiments showed that the collaboration in groups propagated the same amount of changes significantly quicker than when all engineers collaborate individually with each other.

The rest of the paper is structured as follows: Section II provides the motivating examples and their challenges. Section III discusses our approach and a mixed collaboration scenario for group collaboration. Later, Section IV presents the evaluation of our approach, Section V explores the related work and lastly Section VI has our concluding remarks and future work.

II. Motivating Examples

Collaboration among engineers is inevitable in modern engineering. Furthermore, it is essential for engineers to form groups within a project. Imagine a number of engineers that need to create two different groups. Naturally, each engineer has his own private workspace which contains his own artifacts. The engineer has also his/her own view of the artifacts within his/her tool's view. This information needs to be shared among the other four engineers in different ways based on the role of each engineer. This can occur through a shared workspace that every group member can have access to.

The core of this section presents three examples of engineers collaborating in groups using two different collaboration modes. The first mode is synchronous collaboration where a change made by an engineer is propagated instantly to the group and the collaborating engineers. The second mode is asynchronous which requires trigger actions of the engineers to propagate their changes to the group and the collaborating engineers. Finally, we describe when these two different collaboration modes are being mixed together.

A. Synchronous Group Collaboration

In this example, we use the synchronous collaboration mode, from now on, we call it *Instant* mode to present the collaboration among the different engineers within groups.

Figure 1 presents three engineers form a collaborating group (Alice, Bob and Charlie). These engineers are collaborating by sharing artifacts to their group workspace instantly. Imagine Alice creating changes in her private workspace, then these changes are instantly shared to the group workspace, from where they are instantly propagated to the private workspaces

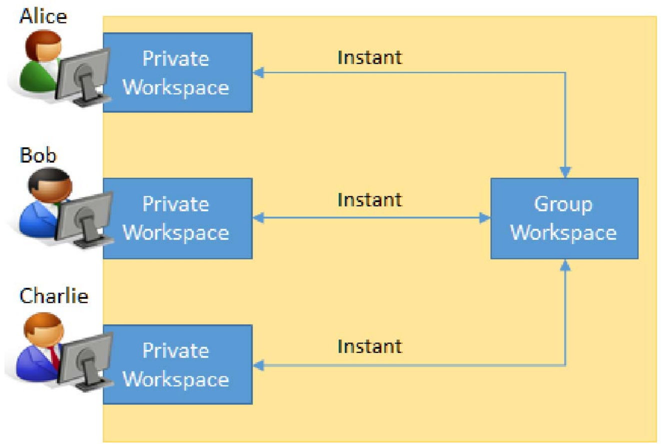

Fig. 1. *Instant* Group Collaboration Scenario

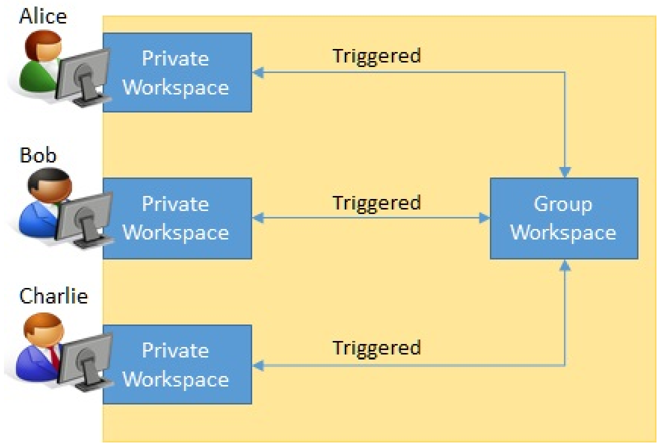

Fig. 2. *Triggered* Group Collaboration Scenario

of Bob and Charlie. Similarly, Bob could make another change in his private workspace and share it instantly with the group workspace allowing all engineers to receive the changes instantly. This scenario is used for engineers that want to collaborate in a similar way as in GoogleDoc.

B. Asynchronous Group Collaboration

In this example, we use the asynchronous collaboration mode, from now on we call it *Triggered* mode. We will apply change propagation with the use of trigger actions between the private workspaces and the group workspace and vice versa. Figure 2 presents the scenario where an arbitrary number of engineers forming a group that shares information asynchronously. As in the previous scenario, Alice creates a change in her private workspace at one point, then she needs to perform a trigger action to propagate her changes to the group workspace. Similarly, Bob and Charlie are required to perform a trigger action on their side to receive Alice's changes to their

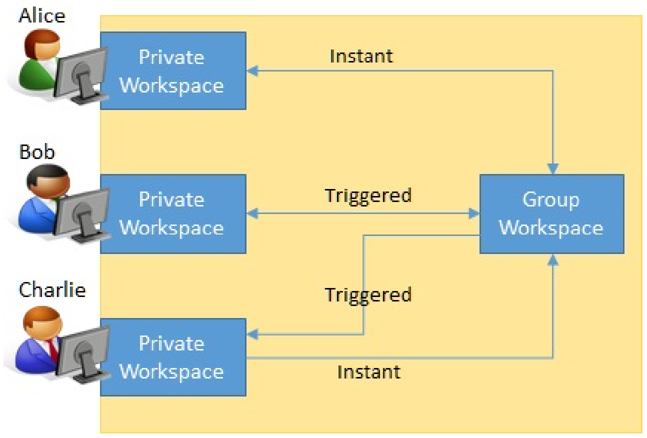

Fig. 3. Mixed Mode Group Collaboration Scenario

private workspace. This asynchronous collaboration scenario is used in all well known version control systems like GIT and SVN.

C. Mixed Use of Collaboration Modes

This motivating example is more complex since it uses both collaboration modes, instant and triggered in the same group collaboration. This scenario assumes that engineers would like to apply different collaboration modes per engineer.

Figure 3 presents different engineers with different collaboration preferences working together in the same collaboration group. In this scenario, Alice is the main developer of the project and she wants to share her changes instantly to the group as well as instantly receive theirs. While Bob is responsible for component integration and wants to receive only production ready artifacts and then share the completed integration work to the group thus prefers only triggered collaboration. Charlie is a test engineer and want to receive production ready code for black box testing while he shares his unit tests immediately with the group for Alice to receive these tests.

Hence, we see that different engineers and roles within group require different collaboration modes. Varying engineer's needs come also with different challenges in order to provide a collaboration framework to support them.

D. Challenges and Objectives

To date, tools supporting group collaboration usually follow asynchronous (GIT, SVN) or synchronous (Collabode [13], [14], GoogleDocs) collaboration mode. At most, some tools may support the change from one collaboration mode to the other (Cloud9 [15], ATCoPE [16]). The main challenge of a collaboration framework is to be able to support different collaboration modes within the same group of collaborators. An even more advanced challenge is not only supporting different collaboration modes for each engineer but also for each direction from and towards the group workspace as presented in Section II-C.

Furthermore, when a collaboration framework succeeds in providing mixed collaboration modes, it has to ensure that the changes are propagated correctly according to the collaboration styles. For example, an engineer with triggered collaboration mode should not receive a change unless he/she makes the necessary triggering actions to receive the change. Next, this framework should ensure that all data is transferred to the collaborating private workspaces regardless of the collaboration modes selected by the engineers. This requires all data to be able to be transferred instantly to these engineer selecting instant collaboration mode and the same data must be available for those engineer which selected triggered mode.

III. Method

This section presents our approach for a collaboration framework that supports group collaboration. The work presented in this paper focuses on the propagation of changes between collaborating engineers within groups. Services beyond the sharing of artifacts, for instance communication between engineers, are out of the scope of this work.

Our collaboration framework provides instant and triggered mode and also the ability to mix these modes based on the individual engineers needs. Additionally, our framework is tool independent since it can be applied to different kinds of artifacts. It allows to define the change propagation between the group workspace and the individual private workspaces (for the collaborations between individual engineers). Our approach consist of i) the *configuration language* to define the collaboration, and ii) the *algorithm* to propagate the changes within a group according to the defined collaboration.

A. Configuration language

The configuration language defines the syntax and semantics on how engineers are allowed to configure their collaboration. It allows engineers to select Instant or Triggered mode and the collaborating parties. In our approach the collaboration is established between two workspaces and it is uni-directional between these workspaces. Figure 4 illustrates the configuration language as it is represented in a meta model representation.

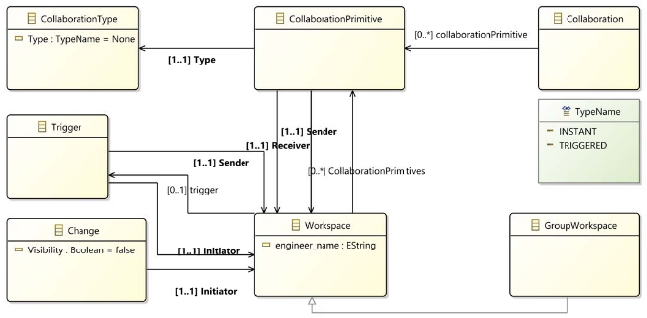

Fig. 4. High Level Model For The Configuration Language

The main elements of the configuration language are the **CollaborationPrimitive** which contains the information that is needed to establish the collaboration between the workspaces, the **CollaborationType**, the **Trigger**, the **Change**, the **Workspace**.

We define the **CollaborationPrimitive** as a tuple C = {T,S,R} such that

- T is the **CollaborationType**.
- S is the sending **Workspace**.
- R is the receiving **Workspace**.

We define **CollaborationType** (Ct) as an object such that Ct.Type $\in$ {**Instant**, **Triggered**}.
Workspace is defined as a tuple Ws = {Owner, CollaborationPrimitives, Changes, Trigger} such as

- Owner is a string.
- CollaborationPrimitives is a set of **CollaborationPrimitive**.
- Changes is a set of **Change**.
- Trigger is an action for starting change propagations when type=Triggered.

To establish the group collaboration, we define the **Group Workspace** as an extension to the workspace element that inherits the owner field but it does not set it with a value(i.e., remains NULL).

Finally, we define a **Change** as an element to represent the changes made to an artifact by an engineer through a given tool (i.e., described by a workspace). The engineer who made the change is considered the change initiator.

The configuration of the group collaboration is the same as the configuration of individual collaboration since the group workspace is an extension of the workspace element. Based on the definition of the collaboration primitive, the engineer defines a sender and a receiver of the change propagation and the type of the collaboration. This allows the engineer to define the mixed mode of collaboration even between his/her relationship to the group.

As an example consider the scenario presented in Figure 3. Using our configuration language we would define six collaboration primitives, three with the engineers' workspaces as senders and the group workspace as receiver and three with the group workspace as sender and receivers each one of the three engineers' workspace. For each one of the primitive, we can define its type separately. Figure 5 presents an instance of our configuration language for the mixed scenario with six collaboration primitives for three engineers and the group workspace. We see that ”Charlie” can define his collaboration with the group differently when he makes a change and when a change is propagated from the group workspace. Furthermore, Alice and Bob can work with their own individual styles within the same collaborative group.

B. Change Propagation

The behavioral aspect of the configuration language is the change propagation algorithm which is depicted in Algorithm 1. The algorithm takes as input either a change or a trigger (usually not both). The changes correspond to creations, modifications, and deletions of artifacts' elements within tools. The triggers denote events when engineers want to propagate these changes to and from a group. They are relevant in case of triggered collaborations only and ignored otherwise. In case a change is passed, the algorithm checks for each collaboration primitive defined by the change initiator (workspace), whether the collaboration primitive is instant and, if true, forwards the change to the defined receiver. In case the receiver is a group workspace, the same propagation algorithm is called, so the changes are propagated through all the collaboration primitives of type instant that have the group workspace as sender. As was discussed above, the receiver then synchronizes these changes with its respective tool normally a tool adapter, completing the change propagation. In case a trigger is passed, the algorithm checks for collaboration primitives in the sender of the trigger (also a workspace), that are of type triggered, and forwards the change to the defined receiver.

Imagine Alice making a change to her artifacts. The propagation algorithm is called (line 1). Automatically, the algorithm takes the change and retrieves its initiator collaboration set (algorithm line 3). It checks collaboration primitives that are of type instant and it will recover the collaboration primitive with the group workspace propagating the change to it (lines 4-5). Then, the group workspace initiates the propagation algorithm and checks its list of collaborations for instant collaborations (lines 6-11). In the scenario depicted in Figure 3 there are not any available. Thus, the algorithm terminates.

Later, Charlie wants to make a change to his code. Therefore, he needs to receive the latest code shared in the group to avoid introducing possible conflicts. He initiates the change propagation algorithm with a trigger that set the sender being the group workspace (line 1 and lines 12 - 15). The group workspace recovers its set of collaborations that are triggered and selects the one that has the engineer who triggered the propagation (i.e., Charlie) as receiver (line 17). When found, the changes submitted by Alice will be passed to Charlie (Line 18). The sender is the group workspace but the initiator of the trigger is an engineer by creating and setting the trigger's sender and initiator field.

Similar, actions will occur when Bob needs Alice's changes. To call the change propagation algorithm from the group workspace, the trigger is instantiated with sender being the group workspace and Bob as its initiator. Then, Bob would be able to receive the changes made by Alice. The lines 22 to 24 are applied when a trigger has propagated changes to the group workspace from a private workspace and the group has collaborations that are of type instant with other private workspaces. In our scenario, if Bob wants to share his changes into the group. He needs to initiate the change propagation algorithm by sending a trigger. Then, the algorithm will retrieve his defined collaboration primitive with the group and it will propagate his changes to the group workspace.

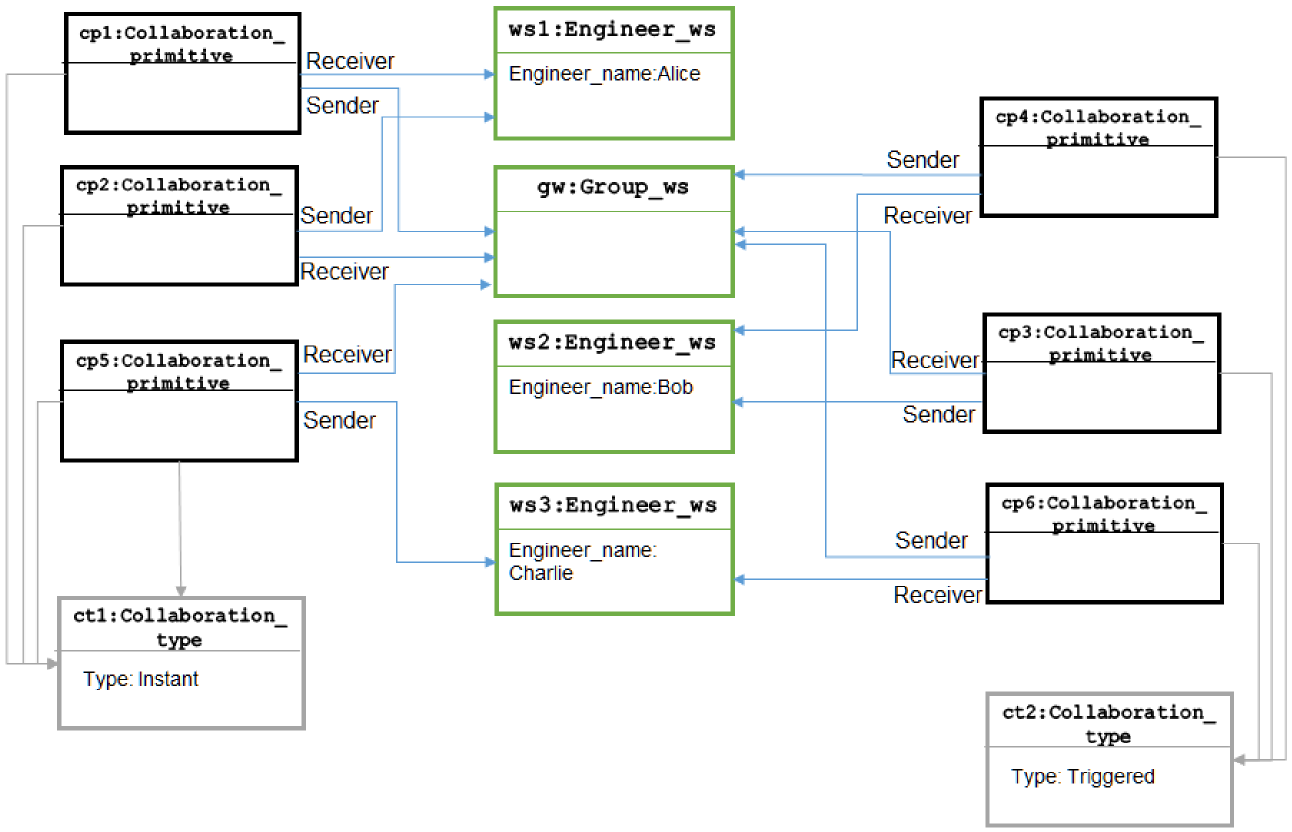

Fig. 5. Illustration of Mixed Scenario with Our Approach as Model Instance

C. Method's Verification

In order to verify our collaboration framework, we also built an Alloy model using the configuration language and the change propagation algorithm 1. In the Alloy model, we define the required predicates based on the requirement of the change propagation algorithm. The Alloy model allowed us to verify the soundness of our approach in respect to the required change propagation among defined collaboration primitives within Alloy model's scenarios.

D. Implementation

As a proof of concept, we have implemented our collaboration framework (configuration language and change propagation algorithm) in our platform called DesignSpace [17]. We have set up change propagation in triggered and instant mode using both JUnit tests as well as actual tools to propagate changes among the engineers workspaces. To interact with the DesignSpace platform, we have built tool adapters for Eclipse, Microsoft Visio [18] and others. The Alloy model has verified a number of properties with its implementation. However, we decided also to test this in the proof of concept implementation making changes to the defined collaboration primitives. During operation, we changed the receiver, in another case, we switched the collaboration primitive's type. After these changes, we observed the behavior of the system. The behavior of the system was as expected and verified by the Alloy model.

IV. Evaluation

This section presents the evaluation of our approach for group collaboration between the different engineers. A main concern that the propagation algorithm may rise is its ability to scale. Thus, this section will answer that question. Does the approach scales with real projects? We also present our exploration for the usefulness of this approach to the different collaboration styles proposed by the literature and applied from different tools.

A. Evaluating Scalability

To evaluate the scalability of our approach we took real projects of different sizes and emulated their sizes in terms of engineers and average changes per engineer. We evaluated the scalability of our approach using four open source projects that are actively developed on GitHub. The selected projects are:

1) Google SyzKaller Project [1]

[1]Syzkaller's GitHub page: https://github.com/google/syzkaller

Algorithm 1 Change Propagation in Synchronous and Asynchronous collaboration style

```
 1: function PROPAGATE CHANGES(c:change, t:Trigger)
 2:     if c != null then
 3:         for all cp:CollaborationPrimitive $in$ c.Initiator.CollaborationPrimitives do
 4:             if cp.type = instant then                                ▷ Check for Instant collaboration type
 5:                 cp.Receiver.addChange(c)
 6:                 if cp.Receiver = group workspace then
 7:                     c.Initiator := cp.Receiver
 8:                     PropagateChanges(c)
 9:                 end if
10:             end if
11:         end for
12:     else if t != null then
13:         for all cp:CollaborationPrimitive $in$ t.Sender.CollaborationPrimitives do
14:             if cp.type = triggered then                            ▷ Check for Triggered collaboration type
15:                 if cp.Sender = t.Sender then
16:                     if t.Sender = group workspace then
17:                         if t.Initiator = cp.Receiver then
18:                             cp.Receiver.addChange(cp.Sender.getChanges())
19:                         end if
20:                     else
21:                         cp.Receiver.addChange(cp.Sender.getChanges())
22:                         if cp.Receiver = group workspace then  ▷ Case where the group receives triggered changes but has
    also instant collaborations to propagate
23:                             c = cp.Sender.getChanges()
24:                             c.Initiator := cp.Receiver
25:                             PropagateChanges(c)
26:                         end if
27:                     end if
28:                 end if
29:             end if
30:             t.terminate()                                          ▷ The execution of the trigger completes
31:         end for
32:     end if
33: end function
```

2) Amazon Web Services SDK for Javascript language [2]
3) Facebook RockSDB Project [3]
4) Microsoft Visual Studio [4]

These projects have different sizes concerning the number of engineers that collaborate in them. The smallest, Google Syzkaller, has 73 contributors as by 07/01/2019 and the largest has 790 as by 07/01/2019. We consider these projects adequate for the scalability test since they are large enough and contain a significant number of engineers working on them. We also chose these projects since they are active and growing projects where a large number of changes occurred by their collaborating engineers.

For each project, we examined a set of 70 commits and we extracted the average number of changed files within these commits. This gives us the average number of changes an engineer performs for our framework to replicate these projects' collaboration behavior using our group collaboration framework. However, since we cannot deduct from GitHub the group sizes of collaborating engineers within these projects, we decided to emulate different sizes of group for every project. The groups sizes we decided to emulate are 5, 10, 20, 50, 100 and all engineers of a project in a group.

For the scalability study, we performed the measurements within a Windows Core-I7, 8GB ram computer, using the Eclipse platform to build unit tests. Each unit test is considered a test case where we spawned the required number of collaborators and we filled their workspaces with instances of changes. Then, within the test, we started the data exchanges. The measurements were performed within the JUnit environment and the number of runs were five per test case. From these five runs, we report their average time in the tables.

Table I summarizes the sizes of the projects selected for the scalability testing of our approach. The number of engineers per project are reported in the second column, the average

[2]SKD-JS's GitHub page: https://github.com/aws/aws-sdk-js

[3]RockSDB's GitHub page:https://github.com/facebook/rocksdb

[4]Visual Studio's GitHub page: https://github.com/Microsoft/vscode

TABLE I
SELECTED PROJECTS SIZE

Projects	Engineers	Avg Changes per Engineer	Total Changes Propagated
Google SyzKaller	73	3	219
AmazonWS SDK-JS	112	14	1568
Facebook RockSDB	414	5	2070
Microsoft Visual Studio	790	3	2370

changes an engineer performs in the project is reported in the third column. The forth column provides the total number of changes that will be propagated within the project. In each project, we are going to create groups of different sizes ranging from 5 engineers per group until all engineers belonging to one group. The last group may have a slightly less engineers, those remaining, than the rest of the groups. We measured the time required for all groups to transfer all changes among the group engineers. The time is measured within the JUnit [19] framework of Eclipse IDE [1].

Table II shows the time required for the propagation of changes among the collaborating engineers in all different groups in instant mode. The second column of Table II presents the time required to propagate all changes created by the engineers in a group of five engineers for all created groups. We see that regardless of the project's size the groups of five engineers require the same amount of time, as expected since the sizes of the groups are the same despite the total number of engineers. The third column presents the time required to propagate the changes engineers made within groups of 10 engineers. We see that doubling the size of the group does not seem to have an impact on the time required to propagate their changes remaining at 16ms. Only when the size of the groups becomes 20 or 50 engineers per group does the time increase to 30ms as seen in columns four and five in Table II. For the project Google Syzkaller, we could not perform the test with 100 engineers per group since there are not enough engineers in total in this project. For the rest of the projects, we see that adding 100 engineers in a group does not have significant impact in the time measurement for instant mode. When we created a single group with all engineers for the test case projects, we see that for the two smaller projects the time remains around 20ms. However, when the number of collaborating engineers in a single group is getting larger, the time also increases from 20ms in the Amazon Web Services (112 engineers) to 60ms for the Microsoft Visio project (790 engineers). The Facebook RockSDB project (414 engineers) requires 40ms time in the single group with all engineers.

From the measurements of Table II, we can conclude that the number of created groups does not really affect the time of propagation for the changes engineers are making in instant mode. However, the number of engineers per group is the key factor that affects the time any number of changes to be propagated among the collaborating engineers.

Table III shows the time measurements for the test cases of our approach using the triggered mode of collaboration. Similarly, with the instant mode, we performed tests dividing the engineers into groups with different sizes. Note, that in our experiments we performed the triggering sequentially from all engineers of a group in order to assess the worst possible case of change propagation. We observed that the number of groups existing does not create significant changes to the time required to propagate the changes. The smallest project requires 16ms while the largest project requires 32ms for the group with size of five engineers. The time is two times higher between the two projects for the groups of 5 engineers but the number of engineers in the largest project is 10 times higher than the smallest project (73 engineers - 790 engineers). This difference on the number of engineers, requires almost 10 times more groups in the largest project but only two twice the time to propagate the changes among all groups.

In contrast to the small effect of the number of groups required, we see that the number of engineers per group has a strong effect on the time measurement. In Table III, we observe that for the triggered mode of collaboration, the larger the group is, the more time it requires to propagate the changes of all members of the group to the remaining engineers in the group. This behavior is expected since the number of triggers that are required to be executed is larger. Again for the smallest test project, the Google SyzKaller, we could not perform the 100 engineers per group test. In almost all cases the time is smaller than a second. The cases with all engineers in one group for the two largest projects (Facebook RockSDB and Visual studio projects) are exceptions with times four second for the RockSDB and 21 seconds for the Visual studio.

Additionally, we see that the number of changes each engineer performs has an impact on the result but does not significantly affect the scalability of the approach. We presented the average changes per engineer in Table I where we see Amazon Web services project has 14 changes per engineer while the Google SyzKaller only three changes per engineer. Even though, the difference of the number of engineers between the two projects is only 39 engineers, the time measurements were twice as high which is explained by the different number of propagated changes.

B. Comparing Groups to Single Collaborations

We also consider the performance benefit of using groups in our approach instead of creating individual collaboration between each engineer. Thus, we performed also tests creating collaboration primitives between the workspaces of each engineer. We used the same test case projects and the same number of changes as presented in Table I.

Table IV presents the time measurements to propagate a number of changes without using group workspaces at the time of writing of this paper. We only create collaborations among the engineers of each project. The results for the instant collaboration mode using the created collaborations is presented in column five while column six presents the same set up using triggered collaboration mode. Column one to three contains the projects and their characteristics again to remind the reader the corresponding sizes.

TABLE II
TEST CASES USING INSTANT COLLABORATION MODE

Project Synchronous Collaboration	5 Members	10 Members	20 Members	50 Members	100 Members	All Engineers
Google SyzKaller	0.016s	0.016s	0.03s	0.03s	-	0.016s
Amazon Web Service SDK	0.016s	0.016s	0.03s	0.03s	0.03s	0.02s
Facebook RockSDB	0.016s	0.016s	0.03s	0.03s	0.03s	0.04s
Microsoft Visual Studio	0.016s	0.016s	0.03s	0.03s	0.04s	0.06s

TABLE III
TEST CASES USING TRIGGERED COLLABORATION MODE

Project Asychronous Collaboration	5 Members	10 Members	20 Members	50 Members	100 Members	All Engineers
Google SyzKaller	0.016s	0.031s	0.038s	0.057s	-	0.069s
Amazon Web Service SDK	0.040s	0.047s	0.064s	0.120s	0.249s	0.334s
Facebook RockSDB	0.032s	0.037s	0.060s	0.135s	0.273s	4.099s
Microsoft Visual Studio	0.032s	0.040s	0.064s	0.145s	0.300s	21s

TABLE IV
TEST CASES USING WORKSPACE TO WORKSPACE COLLABORATION

Test Case	Engineers	Avg Changes per Engineer	Total Changes Propagated	Synchronous Propagation Time	Asynchronous Propagation Time
Google SyzKaller	73	3	219	0.038s	0.08s
Amazon Web Service SDK	112	14	1568	0.05s	0.451s
Facebook RockSDB	414	5	2070	0.063s	7.483s
Microsoft Visual Studio	790	3	2370	0.247s	47.115s

We observe that using a group among all engineers is significantly faster than creating individual collaborations among the different engineers of the project. We see that the benefit of applying groups increases, with the number of collaborators in a project. In the smallest project, without a group collaboration, we measured around 0.04s for instant and 0.08s for the triggered collaboration mode. The same project have around 0.02s and 0.07s time to propagate changes when the group collaboration was applied. For the largest project, we measured 0.25s for the instant collaboration mode and around 47s for triggered mode while using a group collaboration the measurements are 0.06s and 21s respectively.

Similarly, we observe benefit when using the group collaboration for the two intermediate projects, Amazon WS SDK and Facebook RockSDB. Thus, we can conclude that the use of groups can improve the change propagation within the team when applied compared to establishing individual collaborations among the team members.

C. Usefulness of Group Collaboration

Until this point, we demonstrated that our approach is scalable using real open source projects to evaluate it. The discussion about the usefulness will take part in this subsection. Even though, we have not explicitly done an experiment to evaluate the usefulness of group in collaboration, we researched for other works that have evaluate the usefulness of group collaboration. Cook et al. [20] presented a user evaluation of collaborative tools for software engineering. They used a post experiment questionnaire to evaluate the experiments qualitative aspects. The experiment consisted of tools with asynchronous/triggered collaboration mode and synchronous/instant collaboration mode. The usefulness of both modes in the distributed setting was 15.7 out of 20 in the survey climax with 20 being highest and 1 being lowest. The instant mode scored 14.8 and the general source code sharing scored 16.4 out of 20. Furthermore, the rapid use of Git in developing new open source projects as described in Barr et al. [21] is a good indication of the usefulness of groups within software engineering. They also performed a semi-structured interview with six people in four different projects to assess the need and usefulness of branches and revision control systems. The interviews revealed that engineer in collaborating groups require cohesion and isolation during development but also easy merge of work. These features made GIT a successful tool among developers. Therefore, as we argue that we can simulate the GIT collaboration style, we provide the benefits of GIT to collaborating engineers in a group.

D. Limitations

This work is limited to explore the information exchange between the collaborating parties and does not focus on the collaboration awareness and communication between the collaborating parties. The goal of our evaluation was to provide the evidence for the scalability of our approach. For the awareness among collaborating parties, we have built a graphical user interface. It is linked to our cloud platform and adds collaboration management and awareness functionality to our framework.

V. Related Work

Mcguire et al. [22] had proposed a framework called SHADE that tried to be flexible in exchanging the different engineering knowledge among defined producers and consumers. SHADE framework did not have any selection or suggestion to allow engineers to define the synchronous or asynchronous mode of the knowledge exchange. Our approach has these modes in the center of its implementation.

Steinfield, Jang and Pfaff [23] had proposed the TeamSCOPE system to support the needs of distributed team of a project. TeamSCOPE implements features to support communication awareness and artifact sharing. The artifact sharing feature is a stand-alone application which has a preset way for the members of the team to exchange their artifacts. In our approach, engineers can define their mode of collaboration based on their individual needs. Also, our approach is not a specific tool but a framework that can be implemented in different tools and for different artifacts.

Li et al. [24] proposed on their work a permission driven collaboration scheme. The permission scheme requires any sharing of information to be explicitly defined with a permission role towards the accessed data. Their work utilizes virtual groups to address problems within multi-domain environments where different engineering knowledge must be shared among different groups. Their work differs from ours since they do not discuss the change propagation in respect to the different collaboration modes but they describe a mechanism to manage the distribution of members among the groups.

M. Pasqual and O. Weck [25] present on their work the notion of change propagation and a model to efficiently propagate change requests to the different groups of engineers within a project. This work or any related to it differs significant from our work since they discuss the propagation of change as a change request from external source. Whereas we discuss the change propagation as a change made by the engineers to the artifacts one created.

Fylaktopoulos et al. [15] presented an overview of the platforms for cloud based development that most of them support groups with instant or triggered collaboration mode but none of them support the possibility of mixing these modes for the individual engineer. In some cases, the engineers can switch the group collaboration from one mode to the other but they cannot choose individually within the group or per direction, the mode they prefer.

VI. Conclusion

In today's software engineering landscape, collaboration is an essential part of the software engineering process. It enables engineers to overcome the limitations of capabilities of single humans and to resolve complex and difficult tasks. Teams are the core form of collaboration among engineers. Collaborative teams can vary on the how, when and what engineers collaborate with. Tools exist that provide group collaboration. However, they provide limited options for flexibility to engineers to apply their preferred styles. Our approach proposed a framework that can allow engineer form collaborating groups the way they decide. Simultaneously within the group, we can have engineers collaborating instantly or with triggered mode allowing them to apply their individual work style or the style that their role in the project requires. We ensure that these conditions are holding within the group by formally verifying our collaboration language and its change propagation algorithm using Alloy. Later, we present evidence of the scalability of our approach using four real large open source projects. These projects were found on GitHub and were actively developed at the time of writing. These projects had between 73 and 790 engineers working on them. We divided these engineers into different groups ranging in size from five engineers, and increased these numbers until all engineers were in one group. Furthermore, we compared the group usage with the individual collaboration among all engineers in the projects. This allowed us to observe the benefits of using the group collaboration over the individual interlinked collaboration. We plan to examine the ability to extend our framework and implement workspace and group hierarchies to provide further collaboration possibilities.

Acknowledgment

The research leading to these results was funded by the Land of Upper Austria - Joint International PhD Program in Informatics under the grant number Wi-2014-202477/6.
Funded by the JKU Linz Institute of Technology (LIT) the state of Upper Austria, grant no. LIT-2016-2-SEE-019 and Pro2Future, a COMET K1-Center of the Austrian Research Promotion Agency (FFG), grant no. 854184.

References

[1] S. Holzner, *Eclipse: A Java Developer's Guide*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2004.

[2] J. Grudin, "Groupware and social dynamics: Eight challenges for developers," *Communications of the ACM*, vol. 37, no. 1, pp. 92–105, 1994.

[3] ——, "Computer-supported cooperative work: history and focus," *Computer*, vol. 27, no. 5, pp. 19–26, May 1994.

[4] A. Cruz, A. Correia, H. Paredes, B. Fonseca, L. Morgado, and P. Martins, *Towards an Overarching Classification Model of CSCW and Groupware: A Socio-technical Perspective*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 41–56. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33284-5_4

[5] J. Whitehead, "Collaboration in software engineering: A roadmap," in *2007 Future of Software Engineering*. IEEE Computer Society, 2007, pp. 214–225.

[6] J. Loeliger and M. McCullough, *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development, 2nd Edition*. O'Reilly Media, Inc, 2012.

[7] B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato, *Version Control with Subversion*. O'Reilly Media, 2009.

[8] C. R. Turner, A. Fuggetta, L. Lavazza, and A. L. Wolf, "A conceptual basis for feature engineering," *Journal of Systems and Software*, vol. 49, no. 1, pp. 3–15, 1999.

[9] K. Beck and E. Gamma, *Extreme programming explained: embrace change*. addison-wesley professional, 2000.

[10] R. C. Martin, *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002.

[11] J. Portillo Rodriguez, A. Vizcaino, M. Piattini, and S. Beecham, "Tools used in global software engineering: A systematic mapping review," *Information and Software Technology*, vol. 54, no. 7, pp. 663–685, 2012. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84860230057&partnerID=40&md5=51bb5546c00da14778c525859163dcb7

[12] C. M. Harvey and R. J. Koubek, "Cognitive, social, and environmental attributes of distributed engineering collaboration: A review and proposed model of collaboration," *Human Factors and Ergonomics in Manufacturing & Service Industries*, vol. 10, no. 4, pp. 369–393, 2000.

[13] M. Goldman, G. Little, and R. C. Miller, "Collabode: Collaborative coding in the browser," in *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, ser. CHASE '11. New York, NY, USA: ACM, 2011, pp. 65–68. [Online]. Available: http://doi.acm.org/10.1145/1984642.1984658

[14] ——, "Real-time collaborative coding in a web ide," in *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '11. New York, NY, USA: ACM, 2011, pp. 155–164. [Online]. Available: http://doi.acm.org/10.1145/2047196.2047215

[15] G. Fylaktopoulos, G. Goumas, M. Skolarikis, A. Sotiropoulos, and I. Maglogiannis, "An overview of platforms for cloud based development," *SpringerPlus*, vol. 5, no. 1, pp. 1–13, 2016. [Online]. Available: http://dx.doi.org/10.1186/s40064-016-1688-5

[16] H. Fan, C. Sun, and H. Shen, "Atcope: any-time collaborative programming environment for seamless integration of real-time and non-real-time teamwork in software development," in *Proceedings of the 17th ACM international conference on Supporting group work*. ACM, 2012, pp. 107–116.

[17] A. Demuth, M. Riedl-Ehrenleitner, A. Nöhrer, P. Hehenberger, K. Zeman, and A. Egyed, "Designspace: an infrastructure for multi-user/multi-tool engineering," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 2015, pp. 1486–1491.

[18] S. A. Helmers, *Microsoft Visio 2016 step by step*. Microsoft Press, 2015.

[19] V. Massol and T. Husted, *JUnit in Action*. Greenwich, CT, USA: Manning Publications Co., 2003.

[20] C. Cook, W. Irwin, and N. Churcher, "A user evaluation of synchronous collaborative software engineering tools," in *12th Asia-Pacific Software Engineering Conference (APSEC'05)*, Dec 2005, pp. 6 pp.–.

[21] E. T. Barr, C. Bird, P. C. Rigby, A. Hindle, D. M. German, and P. Devanbu, "Cohesive and isolated development with branches," in *Fundamental Approaches to Software Engineering*, J. de Lara and A. Zisman, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 316–331.

[22] J. G. Mcguire, D. R. Kuokka, J. C. Weber, J. M. Tenenbaum, T. R. Gruber, and G. R. Olsen, "Shade: Technology for knowledge-based collaborative engineering," *Journal of Concurrent Engineering: Applications and Research (CERA*, vol. 1, 1993.

[23] C. Steinfield, C.-Y. Jang, and B. Pfaff, "Supporting virtual team collaboration: The teamscope system," in *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work*, ser. GROUP '99. New York, NY, USA: ACM, 1999, pp. 81–90. [Online]. Available: http://doi.acm.org/10.1145/320297.320306

[24] Q. Li, X. Zhang, S. Qing, and M. Xu, "Supporting ad-hoc collaboration with group-based rbac model," in *2006 International Conference on Collaborative Computing: Networking, Applications and Worksharing*, Nov 2006, pp. 1–8.

[25] M. C. Pasqual and O. L. de Weck, "Multilayer network model for analysis and management of change propagation," *Research in Engineering Design*, vol. 23, no. 4, pp. 305–328, 2012.