Improving Integration Process Efficiency through Pull Request Prioritization

Agustín Olmedo, Gabriela Arévalo, Ignacio Cassol, Christelle Urtado, Sylvain Vauttier

Improving Integration Process Efficiency Through Pull Request Prioritization

Agustín Olmedo[1][a], Gabriela Arévalo[2], Ignacio Cassol[1][b], Christelle Urtado [3][c] and Sylvain Vauttier[3][d]

[1] *LIDTUA (CIC), Facultad de Ingeniería, Universidad Austral, Buenos Aires, Argentina*

[2] *DCyT (UNQ), CAETI (UAI), CONICET, Buenos Aires, Argentina*

[3] *EuroMov Digital Health in Motion, Univ. Montpellier & IMT Mines Ales, Ales, France*

Keywords: Distributed Version Control System, Distributed Software Development, Pull-based Development, Pull Request, Software Merging, Merge Conflicts.

Abstract: Pull-based Development (PbD) is widely used in software teams to integrate incoming changes into a project codebase. In this model, contributions are advertised through Pull Request (PR) submissions. Project administrators are responsible for reviewing and integrating PRs. Prioritizing PRs is one of the main concerns of project administrators in their daily work. Indeed, conflicts occur when PRs are concurrently opened on a given target branch and propose different modifications for a same code part. We propose to consider the integration process efficiency (IPE) as the fact that for a given integration cost (*i.e.,* number of conflicts to be solved) the highest gain is reached (*i.e.,* the largest number of PRs are integrated). The goal of this work is to optimize the IPE through PR prioritization. We propose a process that provides a sequence of unconflicting PR groups. This sequence minimizes the number of conflict resolutions and defines an optimized integration order according to the efficiency of the integration process. We apply our proposal to seven representative historical integration sequences from an open source project. In all seven cases, the IPE obtained by our proposal is higher than the historical IPE from 28.73% to 156.52%.

1 Introduction

Many software development teams rely on a Distributed Version Control System (DVCS) to manage concurrent editing of their projects' source code (Brindescu et al., 2014) (Rodríguez-Bustos and Aponte, 2012). Since their appearance in 2001, DVCSs –notably Git (Chacon and Straub, 2014)– have transformed collaborative software development. Each developer has a personal local copy of the entire project history. Changes are first applied locally in the developer copy and are later integrated into a new shared version. Conflicts between parallel changes need to be solved during this integration process (Mens, 2002).

The pull-based development (PbD) model is widely used in collaborative software development (Gousios et al., 2014)(Gousios et al., 2015). In this model, contributions are advertised through *Pull Request* (PR) submissions. Members of the project's core team (aka project administrators) are responsible for reviewing and integrating PRs.

In an open-source context, the PbD model encourages contributions since anyone can propose changes to be integrated into the project, thereby increasing the burden on project administrators who decide whether to integrate contributions into the main development branch or not (Gousios et al., 2015)(Pham et al., 2013). In large projects, the volume of incoming PRs is quite a challenge (Gousios et al., 2015)(Tsay et al., 2014a). Prioritizing PRs is one of the main concerns of project administrators in their daily work (Gousios et al., 2015).

Since contributions can be submitted concurrently, more than one PR can propose changes that modify the same part of the code (*i.e.,* same lines of the same file). When two PRs are concurrently opened on a given target branch, proposing different modifications for identical code parts, a *pairwise conflict* exists. In such case, only the first one can be integrated automatically, while the second requires a conflict resolution.

A PRs *integration sequence* is the chronological

[a] https://orcid.org/0000-0003-2844-6816
[b] https://orcid.org/0000-0002-6309-7503
[c] https://orcid.org/0000-0002-6711-8455
[d] https://orcid.org/0000-0002-5812-1230

order in which the project administrator integrates the opened PRs. The integration sequence defines the integration process. According to Van Der Veen (Van der Veen, 2015), there is an integration sequence in which the number of conflict resolutions is reduced. Although the actual number of conflicts is not reduced, they can be concentrated in a particular PR integration action, thus reducing the number of conflict resolutions. The integration process may be therefore not as efficient as possible because the number of conflict resolutions could be smaller or because the integration of less conflicting PRs (PRs easier to integrate) are inexplicably delayed.

The advantage of an efficient integration process is that the codebase is kept as up-to-date as possible. This is very valuable for the software development process because efficient continuous integration processes limit future integration conflicts (Beck, 2000). In addition, since conflict resolutions are carried out at the end of the integration process, conflicts are solved on an updated codebase, applying the most appropriate resolutions to the latest state of the software.

We propose to consider the integration process efficiency (IPE) as the fact that for a given integration cost (*i.e.,* number of conflicts to be solved) the highest possible gain is reached (*i.e.,* the largest number of PRs are integrated). Therefore, to achieve an optimized IPE, the number of conflict resolutions should be minimized and the least conflicting PRs should be integrated earlier.

We pose the following research questions:

- **RQ1.** *Can the efficiency of the integration process be improved by prioritizing PRs according to conflicts?*
- **RQ2.** *To what extent does our proposal improve the IPE as compared to the historical IPE measured from a real project history?*

The goal of this work is to evaluate the feasibility of IPE optimization through PR prioritization. We propose a process that provides a sequence of unconflicting PR groups that minimizes the number of conflict resolutions and defines an optimized integration order according to the efficiency of the integration process.

In order to answer the research questions, we apply our proposal to the sets of PRs corresponding to seven representative historical integration sequences from an open source project.

The remainder of this paper is structured as follows. Section 2 includes background knowledge and definitions. Section 3 presents our proposed approach to optimize IPE for a set of PRs. Section 4 explains how we evaluate our proposal and gives an insight on the case study that will be used throughout the paper. Section 5 contains results and threats to validity analyses. Related works are included in Section 6 before providing our conclusions and perspectives for this work.

2 Background & Definitions

In DVCS, versions are created when a developer commits the changes currently staged to the project. Versions derive from each other and a derivation relation then links a version to (one of) its predecessor(s). The graph composed of versions and their derivation relations is called a version history. This history can be linear, tree-like (several versions can be derived from the same one, to model software variants for example), or, in the most general case, forms a direct acyclic graph (versions can also be merged). This last and more generic case is the appropriate framework for collaborative development. We consider such histories in the remainder of this study.

A branch is the version history up to a specific version and defines an independent line of development. To enable collaborative or concurrent development, but also to preserve the contents of the main development branch, developments are typically performed on new branches, forked from the latest version currently available on the main development branch (Bird and Zimmermann, 2012). This branch is sometimes called a feature branch as its role in the development process is to bring up a new feature to the project.

Tasks that have to be dealt with (to correct, maintain or improve the software) are stored as issues in an issue tracking system (IST) such as Jira [1]. Contributors choose or are provided a ticket to deal with. To do so, they create a dedicated feature branch in their local repository from the latest software version (fetched from the team repository), check-out this version to get a work copy and make the necessary changes. These changes can then be committed on the feature branch in the local repository so as to enact them. When the development is achieved, the versions locally commited on the feature branch are checked-in (pushed) to the remote team repository. A last step is to submit a Pull Request (PR) to ask the project administrator to include the changes into the project's main development branch.

A PR is a request to the project administrator(s)

[1] https://www.atlassian.com/en/software/jira

to pull versions from a branch to another one. So, a PR contains a source branch from which versions are pulled from and a target branch to which versions are pushed. Project administrators must review and integrate opened PRs in the project. As a result of the review process, PRs can be closed as accepted or rejected. PRs can be accepted with or without changes. In case of being accepted with changes, the contributor must apply the changes suggested by the project administrator by committing a new version in the source branch of the PR, so that the PR is automatically updated.

The accepted PRs are integrated, that is, the changes of the versions of the source branch since it was forked from the target branch are incorporated into the target branch. If versions have been commited on the target branch since the source branch of the PR was forked, there may be merge conflicts since the head version of the source branch do not derive anymore from the head version of the target branch. We consider the changes as text file modifications as Git does. A change is the modification of a text chunck in a file or the addition or deletion of a file. Therefore, merge conflicts between changes are due to the fact that the same line(s) of the same file are modified in both branches.

When there are merge conflicts between the changes of the source branch and the changes of the target branch of a PR, the PR is considered as a conflicting PR. Conflicting PRs require a conflict resolution to integrate their changes into the target branch whereas for unconflicting PRs, the changes are automatically integrated. In the PbD approach, a pairwise conflict exists between two PRs when the integration of a PR would entail a future conflict when integrating the other one to their shared target branch (and reciprocally).

The integration of all opened PRs implies that all conflicts are solved. The final cost of the integration process is therefore the same regardless of the integration sequence. Even though, the IPE can be improved since it is possible to propose an integration sequence that provides greater gain earlier at a lower cost, as compared with other integration sequences. Particularly, the integration process is more efficient when all unconflicting PRs are integrated first because, up to that point in the integration process, the IPE is optimal since these PRs are integrated automatically.

3 Our approach: IPE Optimization

Our approach takes as its inputs a set of opened PRs and generates a sequence of PR groups that defines an optimized integration process. The proposal takes as an hypothesis that all the pairwise conflict resolutions have the same mean cost (the study of an accurate cost model is a perspective for this work). Thus, we do not consider the elementary conflicts (like conflict between files or line blocks, depending on the considered detail level) but only pairwise conflicts.

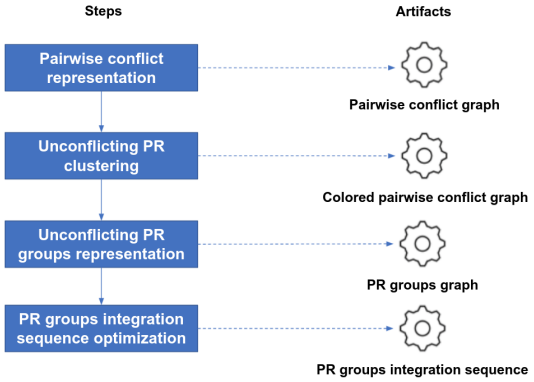

Figure 1: IPE optimization steps with their corresponding artifacts.

PR groups are composed of unconflicting PRs. These can thus be integrated altogether in an automatic manner. In addition, PR groups are ordered so as to integrate first the less conflicting PR group, *i.e.,* the group with the best trade-off between the number of PRs in the group and the number of pairwise conflicts that need to be solved to integrate the PRs of the group. Our approach lies on a four stepped process shown in Figure 1.

Pairwise conflict representation. It processes the set of PRs to obtain the existing pairwise conflicts. Conflicts between PRs are obtained by performing an in-memory merge[2] of the latest version of the source branch of each PR. A pairwise conflict graph is generated where nodes represent each individual PR and edges represent the existence of merge conflicts between PRs pairs.

Unconflicting PR clustering. It processes the pairwise conflict graph to generate a set of groups that each contains unconflicting PRs. Thus, the integration of each PR group into the target branch is optimized because its PRs can be automatically integrated as a single composite PR, resulting in a single conflict resolution process. Among the set of possible solutions to this clustering problem, our proposal gives preference to those ones that have a smaller number of the biggest possible groups. In this way, we minimize the number of conflict resolutions to be actually handled (one for each group).

Our approach solves this clustering problem as a

[2]https://git-scm.com/docs/git-merge

graph coloration problem (Kubale, 2004). The graph coloring algorithm colors nodes that are not adjacent with the same color. In the pairwise conflict graph, two nodes are adjacent when the PRs represented by those nodes have conflicts. Therefore, the non-adjacent nodes are those that do not conflict with each other and these are the ones that are colored with the same color. In this way, unconflicting PRs are grouped together. In addition, the coloring algorithm tries to find the solution with the minimum number of colors, thus the minimum number of unconflicting PR groups.

We model the graph coloration problem as a Constraint Satisfaction Problem (CSP) as follows:

- A variable, *colors*, associated with the definition domain $[0, \#PRs]$, that models the set of colors available to color the graph. We use as an upper bound the number of nodes (the trivial coloring solution).
- A set of variables v_i, associated with the definition domain $[0, \#PRs]$, that model the color affected to each node.
- A set of constraints $v_i < colors$, to model that nodes cannot use more than the available colors.
- A set of constraints $v_i \neq v_j$, for any couple of nodes (v_i, v_j) connected by an edge, to model that nodes connected by edges cannot have the same color,
- An objective that is $minimize(colors)$, so the solver will try to find not only a solution but the best solution and will use the objective to prune some search branches.

Unconflicting PR groups representation. It is quite straightforward from the output of 2nd step. It consists in representing a graph whose k nodes represent colors (unconflicting PR groups) from previous step and whose edges correspond to integration conflicts between PR groups. As any pair of PR groups is in conflict (otherwise they would have the same colour and have been regrouped), the resulting graph is complete. The edges of the graph are labelled with the number of pairwise conflicts that exist between the members of the connected PR groups.

PR groups integration sequence optimization. This step provides an ordered set of PR groups which constitutes our calculated optimized PR group integration sequence according to the IPE. This order is calculated by traversing the PR group graph to search for the best trade-off that maximizes gain (integrate as much PRs as possible) while minimizing cost (avoid conflict resolutions as much as possible). The traversing algorithm is shown in Algorithm 1. It starts with the biggest node (line 10). In each step (lines 12-29), it looks for the best neighbor node, which is the one with the best trade-off between node size and the cumulative cost of the paths from the node to the nodes already visited (*i.e.,* the total cost of integrating the group assuming the already integrated groups).

```
   Input: G is a PR group graph
   Output: S is a sequence of nodes
 1 S = []
   /* select biggest node of G                          */
 2 V = null
 3 maxNodeSize = 0
 4 for node in G.nodes do
 5 |   if node.size > maxNodeSize then
 6 |   |   V = node
 7 |   |   maxNodeSize = node.size
 8 |   end
 9 end
10 S.add(V)
   /* traverse graph nodes starting with the
      biggest node                                      */
11 while length(S) < length(G.nodes) do
12 |   neighbors = G.adjacentsOf(V) - S
13 |   bestNeighbor = null
14 |   bestTradeoffValue = 0
15 |   for neighbor in neighbors do
16 |   |   nodeSize = neighbor.size
17 |   |   EdgeAccumWeight = 0
18 |   |   for s in S do
19 |   |   |   E = G.edgeBetween(s, neighbor)
20 |   |   |   EdgeAccumWeight =
   |   |   |     EdgeAccumWeight + E.weight
21 |   |   end
22 |   |   tradeoffValue = nodeSize /
   |   |     EdgeAccumWeight
23 |   |   if tradeoffValue > bestTradeoffValue
   |   |     then
24 |   |   |   bestNeighbor = neighbor
25 |   |   |   bestTradeoffValue = tradeoffValue
26 |   |   end
27 |   end
28 |   V = bestNeighbor
29 |   S.add(V)
30 end
31 return S
```

Algorithm 1: Get PR group integration sequence

PR ID	Integration timestamp
2052	2017-10-21 16:19:55
2058	2017-10-21 16:20:28
2055	2017-10-21 16:20:57
1978	2017-10-21 16:38:02
2062	2017-10-21 19:00:49
2032	2017-10-21 19:26:30
2057	2017-10-21 19:26:35
2063	2017-10-21 19:47:26
2011	2017-10-21 19:53:54
2064	2017-10-21 19:55:32
1996	2017-10-21 19:58:33
1983	2017-10-21 20:00:03
1955	2017-10-21 20:13:49
1974	2017-10-21 20:16:24
1917	2017-10-22 15:52:31
2033	2017-10-23 16:25:40
2070	2017-10-23 21:43:22
2077	2017-10-25 20:16:46
2072	2017-10-27 15:26:04
2073	2017-10-27 15:26:48
2074	2017-10-27 15:27:10
2075	2017-10-27 15:27:47
2076	2017-10-27 15:28:15
2067	2017-10-27 15:28:57
2068	2017-10-27 15:29:27
1951	2017-10-27 17:45:27
1990	2017-10-27 22:54:29
2083	2017-10-29 16:18:09
1930	2017-10-29 22:57:07

Table 1: Historical integration sequence from October 16, 2017 to October 30, 2017 of Antlr4 project.

4 Evaluation on a case study

In this section we explain how we evaluate our proposal. We set up a case study that consists in extracting integration sequences from the history of a software project and applying our proposal to the set of PRs for each integration sequence.

4.1 Case study definition

Project selection. We selected the project Antlr4 [3] since it (i) is publicly available in GitHub, (ii) has more than ten thousand stars, (iii) has more than two thousand forks, and (iv) follows a PbD process. In addition, its history has more than a thousand PRs, which is an adequate number to evaluate the proposal. Therefore, the project is suitable because it is popular and has a good number of contributors and contributions.

Integration sequence extraction. Based on a time window (t_1, t_2), we extract the integration sequence from the project history. The range of the selected time windows in the case study is 2 weeks based on the usual sprint length in agile methodologies such as Scrum (Diebold et al., 2015) or the size of the merge window used in the Linux kernel project (German et al., 2016). Opened PRs within the time window (t_1, t_2) can be integrated or rejected within or after the time window or remain open indefinitely. Figure 2 shows all these PR types for PRs that are created before the time window and those that are created within the time window. Our case study includes all the PRs integrated within the time window to obtain the integration sequence. In Figure 2, these PRs correspond to PR types 5 and 6. Rejected PRs (types 1, 4, 7 and 9) and those that are never closed (types 2 and 10) are not incuded in the case study because those PR types are not integrated into the project history. PR types 3 and 8 are not included because they are not integrated within the considered time window. Therefore, we obtain an integration sequence by extracting integrated PRs within the time window (t_1, t_2) and sorting them by their integration timestamp. This sequence is the *historical integration sequence*. Table 1 shows an example of a *historical integration sequence* of the Antlr4 project corresponding to a time window from October 16, 2017 to October 30, 2017.

[3] https://github.com/antlr/antlr4

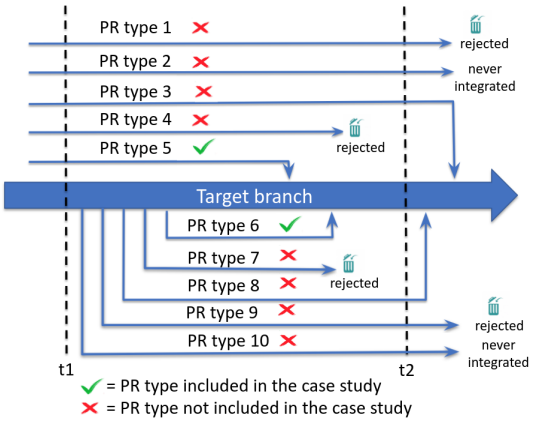

Figure 2: PR types according to their project life cycle for a time window. Type 5 and 6 are targets of our case study because those PRs are integrated into the defined time windows (t_1, t_2).

4.2 Historical IPE

We evaluate the case study by calculating the IPE as a function of the cumulative cost and gain of each integration step where the cost is the number of pairwise conflicts to solve and the gain is the number of integrated PRs.

Table 2 shows the cumulative cost and gain values for each integration step corresponding to the historical integration sequence shown in Table 1. The pairwise conflicts corresponding to this integration

Integration step	PR ID	Cumulative Cost (x)	Cumulative Gain (y)
1	2052	0	1
2	2058	0	2
3	2055	0	3
4	1978	0	4
5	2062	0	5
6	2032	0	6
7	2057	0	7
8	2063	0	8
9	2011	0	9
10	2064	0	10
11	1996	7	11
12	1983	7	12
13	1955	7	13
14	1974	8	14
15	1917	8	15
16	2033	8	16
17	2070	8	17
18	2077	8	18
19	2072	8	19
20	2073	8	20
21	2074	8	21
22	2075	8	22
23	2076	8	23
24	2067	8	24
25	2068	8	25
26	1951	8	26
27	1990	20	27
28	2083	20	28
29	1930	22	29

Table 2: Cumulative cost and gain values of each integration step for the historical integration sequence shown in Table 1.

sequence can be seen visually in Figure 4. We start considering that the first 10 PRs of Table 1 do not conflict with each other. They are integrated at zero cost since project admininstrator does not have to solve any conflict to integrate them. Since PR #1996 conflicts with 7 already integrated PRs (#2052, #2058, #2055, #2062, #2057, #2063 and #2064), it is required to solve conflicts with those PRs to integrate it. Therefore the cost of integrating PR #1996 is 7. As PRs #1955 and #1983 do not conflict with any PR already integrated, then they are integrated at zero cost and the cumulative cost is still 7. PR #1974 conflicts with PR #2011, so the cumulative cost is 8. The following 12 PRs (#1917, #2033, #2070, #2077, #2072, #2073, #2074, #2075, #2076, #2067, #2068, #1951) do not conflict with the previously integrated PRs, thus they are integrated at zero cost and the cumulative cost is still 8. As PR #1990 conflicts with 12 previously integrated PRs (#2052, #2058, #2055, #2062, #2057, #2063, #2064, #1996, #2070, #2077, #2067, #2068), the cost is 12 and the cumulative cost is 20. PR #2083 does not conflict with any of the previously integrated PRs, so it is integrated at zero cost and the cumulative cost is still 20. Finally, PR #1930 conflicts with PRs #1951 and #2083, thus the integration cost is 2 and the cumulative cost is 22.

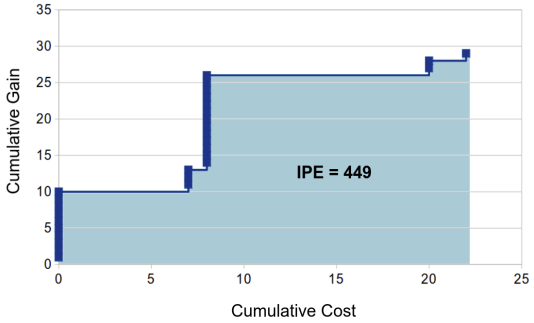

Figure 3: Integration trajectory corresponding to the historical integration sequence shown in Table 1. The area under the trajectory represents the IPE.

Figure 3 maps the cost/gain to a trajectory where axis x is the cumulative cost, axis y, the cumulative gain. Each (x,y) coordinates is an integration step and the line between points is the integration step cost. The trajectory models the integration sequence. The area under the trajectory represents the IPE because a larger area means that a higher gain is achieved at a lower cost. The IPE corresponding to the historical integration sequence represented on Figure 3 is 449. In addition, the number of conflict resolutions is 4, corresponding to the integration of PRs #1996, #1974, #1990 and #1930.

4.3 Optimized IPE

We apply our approach to the set of PRs corresponding to the historical integration sequence shown in Table 1. Figure 4 shows the pairwise conflicts graph resulting from applying the first step of the proposal to the set of PRs of the historical integration sequence. Figure 5 shows the corresponding colored graph after applying the second step of the proposal. The chromatic number k is 3 meaning that PRs can be grouped into three groups where PRs do not conflict with each other. Figure 6 shows the PR group graph obtained after applying the third step of the proposal to the colored pairwise conflict graph of Figure 5. Lastly, applying the fourth step of the proposal to the PR group graph results in the following PR group integration sequence: $G0 \rightarrow G1 \rightarrow G2$. So, 25 PRs can be automatically integrated into the target branch and the remaining 4 PRs require manual conflict resolution. The three PRs corresponding to $G1$ do not conflict with each other but 14 pairwise conflicts need to be solved to integrate them into the target branch. Finally, it is needed to solve 8 pairwise conflicts to integrate the PR belonging to $G2$ to the target branch.

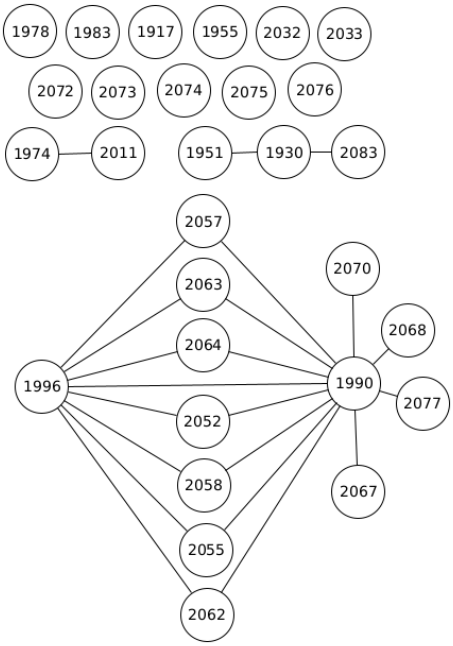

Figure 4: Pairwise conflict graph obtained from the set of PRs of Table 1.

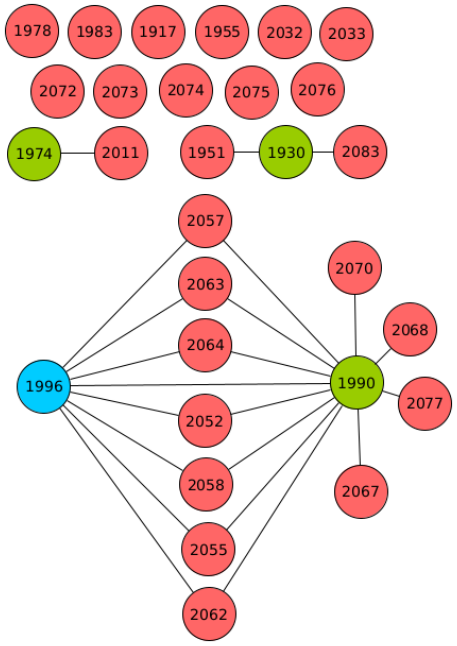

Figure 5: Colored pairwise conflict graph resulting from applying the second step of the proposal to the pairwise conflict graph of Figure 4.

Integration step	Group	Cumulative Cost (x)	Cumulative Gain (y)
1	G0	0	25
2	G1	14	28
3	G2	22	29

Table 3: Cumulative cost and gain values of each integration step for the proposed PR group integration sequence.

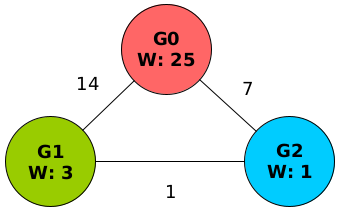

Figure 6: PR group graph obtained from the colored pairwise conflict graph of Figure 5.

We evaluate the integration sequence optimized by our proposal by calculating its IPE. Figure 7 shows both the historical and our optimized integration sequences. The IPE corresponding to our optimized integration sequence is 578 (28.73% higher than the historical one). In addition, its number of conflict resolutions is 2, corresponding to the integration of the $G1$ and $G2$ groups.

5 Experimental results

In this section, we report and discuss the main results achieved in our work. We extract PR information from the *GHTorrent* dataset[4] (Gousios and Zaidman, 2014). It should be noted that we obtain pairwise conflicts from the project history considering the head version of the PRs on the submission date.

We apply our proposal to seven representative historical integration sequences of the Antlr4 project. The selection criterion for these sequences is the number of unconflicting PR groups obtained by applying the coloring algorithm to the pairwise conflict graph corresponding to the integration sequences.

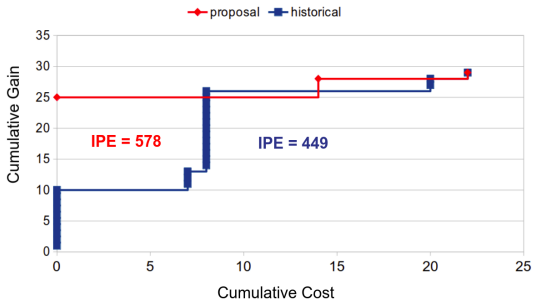

Figure 7: Comparative graph between the historical integration trajectory and the PR group integration trajectory.

[4]We download the latest backup dated 2019-06-01 from https://ghtorrent.org/downloads.html

Table 4 shows information about the pairwise conflict graphs corresponding to the selected integration sequences. The number of PRs (# PRs) and the number of pairwise conflicts (# Pairwise Conflcits) corresponds to the number of nodes and edges of the pairwise conflict graph respectively. The number of potential conflict resolutions (# Potential conflict resolutions) corresponds to the worst case of the number of conflict resolutions for the case study. Finally, the number of unconflicting PR groups (# Unconflicting PR Groups) corresponds to the number of groups (colors) that we obtained by applying the coloring algorithm explained in Section 3.

RQ1. *Can the efficiency of the integration process be improved by prioritizing PRs according to conflicts?*

Table 5 shows the comparison between the historical IPE and the optimized IPE obtained by our proposal for each integration sequence shown in Table 4. We observe that for all selected integration sequences our approach improves the historical IPE, which means that less conflicting PRs are integrated earlier in the proposed PR group integration sequence. In addition, the number of conflict resolutions resulting in the proposed PR group integration sequence is less than or equal to the historical one. Therefore, we can conclude that prioritizing PR according to conflicts can improve the IPE.

RQ2. *To what extent does our proposal improve the IPE as compared to the historical IPE measured from a real project history?*

Table 5 shows a percentage of improvement of the IPE between 28.73% and 156.52% as compared to the historical IPE. We can thus conclude that our proposal can help project administrators significantly by optimizing their integration work when the number of pairwise conflicts is proportionally large with respect to the number of PRs and the number of potential conflict resolutions is close to the optimal.

It should be noted that our solution allows project administrators to systematically reduce the number of conflict resolutions but not the number of conflicts to be solved.

Threats to validity. The criteria defined in order to extract conflicts between PRs already integrated into the project history could be further discussed. Another identified validity threat could be related to the fact that the proposal was evaluated on a limited number of historical integration sequences. On the other hand, we consider they are representative and include a wide variety of cases. Finally, it would be argued that the evaluation was not performed on distinct software projects. In this way, we prioritized the selection of the case studies in order to ensure real different scenarios more than the inclusion of many projects that could contain no significant differences.

6 Related work

In recent years, researchers have conducted several studies to better understand the PbD model usage and to improve the development process in this model (Tsay et al., 2014a) (Tsay et al., 2014b) (Gousios et al., 2014) (Gousios et al., 2015) (Gousios et al., 2016). We can distinguish two processes within the PbD model that are closely related: the review and integration processes. The review process reviews the code to meet the project's quality standards and decides to accept or reject a contribution. The integration process integrates the accepted PRs by solving merge conflicts for conflicting PRs.

From the review process viewpoint, works that study latency factors are (Gousios et al., 2014), (Zhang et al., 2014), (Yu et al., 2015), (Yu et al., 2016b) and (Kononenko et al., 2018) that propose different quantitative / qualitative analyzes to identify latency factors in the PR review process. These studies inspired our proposal to study and improve the latency of the PR integration process. Other works study the PR acceptance factors (Tsay et al., 2014a), (Rahman and Roy, 2014), (Gousios et al., 2015), (Zampetti et al., 2019), (Legay et al., 2018) identifying and analyzing the social and technical factors behind PR acceptance or rejection. Gousios et al. (Gousios et al., 2014) indicates that 27% of rejected PRs are conflicting PRs. Therefore, our work could also improve the acceptance rate by reducing the number of conflicting PRs. Some works recommend the most suitable project administrator for reviewing a given PR (Thongtanunam et al., 2014), (Ying et al., 2016), (Yu et al., 2016a), (Jiang et al., 2017) and (Jiang et al., 2019).

Among the works that study PR prioritization we find some that prioritize PRs by response and acceptance likelihood (Van Der Veen et al., 2015), (Azeem et al., 2020a) and (Azeem et al., 2020b). The goal of these prioritization strategies is minimizing the time required to obtain updates from contributors on the most useful PRs. Zhao et al. (Zhao et al., 2019) propose a learning-to-rank approach to prioritize PRs that can be quickly reviewed by project administrators in order to review more PRs in a period of time or be able to review any PR when they have a few minutes.

Case Study	Time Window Start Date	Time Window End Date	# PRs	# Pairwise Conflicts	# Potential Conflict Resolutions	# Unconflicting PR Groups
1	2017-02-12	2017-02-26	32	23	22	2
2	2017-10-16	2017-10-30	29	22	15	3
3	2018-07-14	2018-07-28	10	7	4	4
4	2018-10-28	2018-11-11	16	29	11	5
5	2017-10-01	2017-10-15	16	47	14	6
6	2017-10-08	2017-10-22	30	74	25	7
7	2018-11-05	2018-11-19	36	63	15	8

Table 4: Information about selected historical integration sequences

Case Study	Historical		Optimized		IPE improvement (%)
	# Conflict Resolutions	IPE	# Conflict Resolutions	IPE	
1	3	503	1	690	37.17
2	4	449	2	578	28.73
3	3	23	3	59	156.52
4	7	294	4	387	31.63
5	11	480	5	632	31.66
6	17	1138	6	1978	73.81
7	11	1065	7	2014	89.11

Table 5: Comparison between the historical and the optimized IPE obtaining by our proposal for each integration sequence shown in Table 4.

Recently, Saini and Britto (Saini and Britto, 2021) use a Bayesian Network to prioritize PRs based on acceptance probability, change type (i.e. bug fixing, new feature, refactoring) and presence or absence of merge conflicts. Their main goal is to decrease the overall lead time of code review process along with helping to reduce the workload of project administrators. The work presented in this paper also prioritizes PRs using a strategy very similar to that proposed by Zhao et al. (Zhao et al., 2019), but with the aim of improving the integration process instead of the review process.

From the integration process viewpoint, works that study pairwise conflicts (Ma et al., 2017), (Zhang et al., 2018) propose an analysis of the frequency and difficulty of resolution. The tool proposed by Van Der Veen et al. (Van Der Veen et al., 2015) offers an alternative view to Github's PR interface, which allows developers to sort opened PRs by their number of pairwise conflicts. This information is useful for project administrators to be aware of the impact of integrating a PR in the integration process. Other works are focused on the detection of duplicate PRs (Li et al., 2017), (Ren et al., 2019), (Wang et al., 2019), (Li et al., 2021) by comparing the textual similarity of title and description or by the similarity between code changes. Duplicate PRs are not considered in this paper.

7 Conclusions

In this paper, we optimize the efficiency of the integration process through PR prioritization. In particular, our approach is able to automatically calculate a sequence of unconflicting PRs groups that minimizes the number of conflict resolutions and orders the integration of the groups with the best trade-off between the number of PRs and the number of conflicts still to be solved. We evaluate the effectiveness of our approach applying it to seven representative historical integration sequences of the Antlr4 project. Results of our study show that the IPE of the integration sequence calculated by our approach is higher than the historical IPE from 28.73% to 156.52%.

It should be noted that applying the approach in a real environment could have side effects on the work habits of the people involved in the project.

We plan to conduct an empirical study on a large number of projects that use PRs intensively. We also plan to propose a tool integrated into platforms that support PbD (*e.g.,*GitHub, Bitbucket, etc.), in order to (i) further evaluate the usefulness of our proposal, (ii) study the side effects on the work habits when using our proposal, and (iii) discover additional factors that can be used to improve the IPE performed by our proposal. The study of accurate models for the evaluation of the cost and the gain of PR merges is also a perspective for our work.

REFERENCES

Azeem, M. I., Panichella, S., Di Sorbo, A., Serebrenik, A., and Wang, Q. (2020a). Action-based recommendation in pull-request development. In *Proceedings of the International Conference on Software and System Processes*, pages 115–124.

Azeem, M. I., Peng, Q., and Wang, Q. (2020b). Pull request prioritization algorithm based on acceptance and response probability. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, pages 231–242. IEEE.

Beck, K. (2000). *Extreme programming explained: embrace change*. addison-wesley professional.

Bird, C. and Zimmermann, T. (2012). Assessing the value of branches with what-if analysis. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, pages 1–11.

Brindescu, C., Codoban, M., Shmarkatiuk, S., and Dig, D. (2014). How do centralized and distributed version control systems impact software changes? In *Proceedings of the 36th International Conference on Software Engineering*, pages 322–333.

Chacon, S. and Straub, B. (2014). *Pro git*. Springer Nature.

Diebold, P., Ostberg, J.-P., Wagner, S., and Zendler, U. (2015). What do practitioners vary in using scrum? In *International Conference on Agile Software Development*, pages 40–51. Springer.

German, D. M., Adams, B., and Hassan, A. E. (2016). Continuously mining distributed version control systems: an empirical study of how linux uses git. *Empirical Software Engineering*, 21(1):260–299.

Gousios, G., Pinzger, M., and Deursen, A. v. (2014). An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, pages 345–355.

Gousios, G., Storey, M.-A., and Bacchelli, A. (2016). Work practices and challenges in pull-based development: the contributor's perspective. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 285–296. IEEE.

Gousios, G. and Zaidman, A. (2014). A dataset for pull-based development research. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 368–371.

Gousios, G., Zaidman, A., Storey, M.-A., and Van Deursen, A. (2015). Work practices and challenges in pull-based development: The integrator's perspective. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 358–368. IEEE.

Igaki, H., Fukuyasu, N., Saiki, S., Matsumoto, S., and Kusumoto, S. (2014). Quantitative assessment with using ticket driven development for teaching scrum framework. In *Companion proceedings of the 36th international conference on software engineering*, pages 372–381.

Jiang, J., Lo, D., Zheng, J., Xia, X., Yang, Y., and Zhang, L. (2019). Who should make decision on this pull request? analyzing time-decaying relationships and file similarities for integrator prediction. *Journal of Systems and Software*, 154:196–210.

Jiang, J., Yang, Y., He, J., Blanc, X., and Zhang, L. (2017). Who should comment on this pull request? analyzing attributes for more accurate commenter recommendation in pull-based development. *Information and Software Technology*, 84:48–62.

Kononenko, O., Rose, T., Baysal, O., Godfrey, M., Theisen, D., and De Water, B. (2018). Studying pull request merges: a case study of shopify's active merchant. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, pages 124–133.

Kubale, M. (2004). *Graph colorings*, volume 352. American Mathematical Soc.

Legay, D., Decan, A., and Mens, T. (2018). On the impact of pull request decisions on future contributions. *arXiv preprint arXiv:1812.06269*.

Li, Z., Yin, G., Yu, Y., Wang, T., and Wang, H. (2017). Detecting duplicate pull-requests in github. In *Proceedings of the 9th Asia-Pacific Symposium on Internetware*, pages 1–6.

Li, Z.-X., Yu, Y., Wang, T., Yin, G., Mao, X.-J., and Wang, H.-M. (2021). Detecting duplicate contributions in pull-based model combining textual and change similarities. *Journal of Computer Science and Technology*, 36(1):191–206.

Ma, P., Xu, D., Zhang, X., and Xuan, J. (2017). Changes are similar: Measuring similarity of pull requests that change the same code in github. In *Software Engineering and Methodology for Emerging Domains*, pages 115–128. Springer.

Mens, T. (2002). A state-of-the-art survey on software merging. *IEEE transactions on software engineering*, 28(5):449–462.

Pham, R., Singer, L., Liskin, O., Figueira Filho, F., and Schneider, K. (2013). Creating a shared understanding of testing culture on a social coding site. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 112–121. IEEE.

Rahman, M. M. and Roy, C. K. (2014). An insight into the pull requests of github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 364–367.

Ren, L., Zhou, S., Kästner, C., and Wasowski, A. (2019). Identifying redundancies in fork-based development. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 230–241. IEEE.

Rodríguez-Bustos, C. and Aponte, J. (2012). How distributed version control systems impact open source software projects. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 36–39. IEEE.

Saini, N. and Britto, R. (2021). Using machine intelligence to prioritise code review requests. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 11–20. IEEE.

Thongtanunam, P., Kula, R. G., Cruz, A. E. C., Yoshida, N., and Iida, H. (2014). Improving code review effectiveness through reviewer recommendations. In *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 119–122.

Tsay, J., Dabbish, L., and Herbsleb, J. (2014a). Influence of social and technical factors for evaluating contribution in github. In *Proceedings of the 36th international conference on Software engineering*, pages 356–366.

Tsay, J., Dabbish, L., and Herbsleb, J. (2014b). Let's talk about it: evaluating contributions through discussion in github. In *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*, pages 144–154.

Van der Veen, E. (2015). Prioritizing pull requests. -.

Van Der Veen, E., Gousios, G., and Zaidman, A. (2015). Automatically prioritizing pull requests. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 357–361. IEEE.

Wang, Q., Xu, B., Xia, X., Wang, T., and Li, S. (2019). Duplicate pull request detection: When time matters. In *Proceedings of the 11th Asia-Pacific Symposium on Internetware*, pages 1–10.

Ying, H., Chen, L., Liang, T., and Wu, J. (2016). Earec: leveraging expertise and authority for pull-request reviewer recommendation in github. In *2016 IEEE/ACM 3rd International Workshop on Crowd-Sourcing in Software Engineering (CSI-SE)*, pages 29–35. IEEE.

Yu, Y., Wang, H., Filkov, V., Devanbu, P., and Vasilescu, B. (2015). Wait for it: Determinants of pull request evaluation latency on github. In *2015 IEEE/ACM 12th working conference on mining software repositories*, pages 367–371. IEEE.

Yu, Y., Wang, H., Yin, G., and Wang, T. (2016a). Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment? *Information and Software Technology*, 74:204–218.

Yu, Y., Yin, G., Wang, T., Yang, C., and Wang, H. (2016b). Determinants of pull-based development in the context of continuous integration. *Science China Information Sciences*, 59(8):1–14.

Zampetti, F., Bavota, G., Canfora, G., and Di Penta, M. (2019). A study on the interplay between pull request review and continuous integration builds. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 38–48. IEEE.

Zhang, X., Chen, Y., Gu, Y., Zou, W., Xie, X., Jia, X., and Xuan, J. (2018). How do multiple pull requests change the same code: A study of competing pull requests in github. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 228–239. IEEE.

Zhang, Y., Yin, G., Yu, Y., and Wang, H. (2014). A exploratory study of@-mention in github's pull-requests. In *2014 21st Asia-Pacific Software Engineering Conference*, volume 1, pages 343–350. IEEE.

Zhao, G., da Costa, D. A., and Zou, Y. (2019). Improving the pull requests review process using learning-to-rank algorithms. *Empirical Software Engineering*, 24(4):2140–2170.