Identifying Participants for Collaborative Merge

Catarina Costa

Fluminense Federal University
Niteroi – RJ, Brazil
catarina@ufac.br

ABSTRACT

Software development is typically a collaborative activity. Development in large projects often involves branches, where changes are performed in parallel and merged periodically. While, there is no consensus on who should perform the merge, team members typically try to find someone with enough knowledge about the changes in the branches. This task can be difficult in cases where many different developers have made significant changes. My research proposes an approach, TIPMerge, to help select the most appropriate developers to participate in a collaborative merge session, such that we maximize the knowledge spread across changes. The goal of this research is to select a specified number of developers with the highest joint coverage. We use an optimization algorithm to find which developers form the best team together to deal with a specific merge case. We have implemented all the steps of our approach and evaluate parts of them.

CCS Concepts

Software and its engineering → Software configuration management and version control systems.

Keywords

Version Control, Branch Merging, Participants Recommendation.

1. INTRODUCTION

The software development process often requires parallel work made by multiple people because of a host of reasons, such as time-to-market and maintenance of old versions. Such parallel work is usually isolated from each other by using branches. However, the task of integrating branches with changes made in parallel can be difficult [4, 8]. It may demand knowledge that a single developer may not have, especially if multiple developers performed changes on the branches and these changes conflict, directly or indirectly. In fact, a recent survey [8] showed that 75% of software developers usually ask for help from other developers when perform merges that include conflicts. They normally try to find someone who has enough knowledge about the changes.

Existing approaches do not address how to select developers who are the best suited to perform a collaborative merge. Some proposals focus on supporting collaborative model merging [6, 14] and collaborative real-time editor [15, 19]. These studies aim to enable all involved participants to work in a collaborative fashion. We also found works that assign people to software activities, where most involve assigning people to issues [1–3, 7, 13, 17, 18, 26] or pull request [12, 16, 24, 25]. However, assigning developers to a collaborative merge brings additional chal-

lenges, as the number of developers, the time interval between syncing of the branches, the number of commits per branch varies, and can be very high, making the process a lot more complex. Dependencies across branches and the changes add further complexity.

We propose TIPMerge[1] [9, 10], an approach to identify developers to merge branches. TIPMerge recommends a ranked list of developers who are the most appropriate to integrate a pair of branches, allowing more than one developer to be chosen for a collaborative merge session. The goal of this research is to use TIPMerge to identify developers with complementary knowledge to perform merge. After presenting the ranking, TIPMerge checks the developer's knowledge coverage about changed files and methods and shows which developers have the highest coverage.

As the number of developers in the ranking can be high, we use an optimization algorithm to find which developers form the best team to deal with a specific merge case. We have implemented all the steps of our approach and have started to evaluate parts of them. Our current implementation is able to analyze Git repositories, independently of their programming language[2]. Our tool is developed in Java, uses Git commands to extract the repository information, and Dominoes [21–23] to identify logical dependencies among files.

2. BACKGROUND AND RELATED WORK

Two common concepts in Version Control Systems are branching and merging. The former is used to promote the development of software in isolation from the main line of development or other branches so that changes don't interfere with each other. The latter is then used to combine the work that was made in isolation. Therefore, teams are free to explore, develop, test, and stabilize the code base without interfering with each other [5]. However, branches may introduce a false sense of safety, as changes in different branches eventually have to be merged, and long periods in isolation may increase the chances of conflicts (when changes are syntactically or semantically inconsistent) [20].

The task of performing the merge can be difficult. There is no consensus about who should perform this activity. Software developers cited the most experienced developer as the best option [8]. But identifying the most experienced developer is non-trivial. Further, there may not be a single developer whose experience covers all the changes.

Developers' assignment is a common research topic in software engineering. Several works address it in the context of identifying developers who have the expertise to work on an issues [1–3, 7, 13, 17, 18, 26] or pull request [12, 16, 24, 25]. Works about issues suggest a small number of developers suitable to resolve a specific report. Pull request works are related to the recommendation of developers for branch merging, but, in general, pull requests contain commits from a single developer and are small [11], akin to merging a workspace commit.

---

[1] TIPMerge is available at https://github.com/gems-uff/tipmerge as an open sourced tool (MIT License).

[2] TIPMerge is language agnostic when running at file-level granularity. At the method-level granularity, it is currently compatible with Java.

In contrast, in the general case of branch merges, the number of developers, the syncing interval, and the number of commits per branch varies and can be high in some situations. For example, a merge in the Rails project (https://goo.gl/7fP3fv) includes commits made by 47 developers in one branch and 52 developers in the other [8, 9], a much higher scale than a pull request merge.

3. TIPMERGE

In this work, we propose TIPMerge [9, 10], an approach to assign developers to merge branches. TIPMerge follows a 5 step process: (1) It extracts the data from the repository until the branch tips, i.e., the two most recent commits of the two branches that will be merged. (2) It mines the files that were frequently changed together in the previous history (before the branch creation) to detect logical coupling among these files. (3) It checks who edited "key" files - files that were changed in both branches and files with logical dependencies across branches. This step collects information about changed files and who changed these files in the branches and in the previous history. (4) It generates a ranking of suitable candidates to perform the merge based on a medal count system. Finally, it runs a coverage analysis that (5) considers the ranking and contributions of each developer to optimize (when given a number of people suitable for a collaborative merge session) which developers together have the maximum coverage of the key files to perform the merge.

TIPMerge considers only developers with expertise on key files. We mainly focus on key files because changes to these files can potentially lead to conflicts and be more difficult to merge. Files changed in both branches can lead to direct conflicts, whereas files changed in one branch, but with dependencies to other file changed in the other branch can lead to indirect conflicts. We use Git commands to identify changed files in both branches and Dominoes [21–23] to identify logical dependencies, i.e., files that were frequently changed together in the history.

TIPMerge uses a medal counting system that checks contributions in key files to rank the most appropriate developers to perform the merge. A gold medal is awarded when a developer changes a key file in a branch. A silver medal is awarded when a developer has changed a key file in the past. A bronze medal is awarded when a developer changes a file that depends on another file. This is analogous to how countries are ranked in the Olympic Games based on medal counts. So, to generate the ranking, our algorithm prioritizes developers with gold medals, then silver, and lastly bronze.

The coverage analysis is also based on changes made in key files. TIPMerge identifies developers who have complementary expertise based on changes made in the key files in the branches and previous history. Although the ranking sorts developers based on their expertise, the top developers in the ranking may have similar knowledge, as they may have changed the same files. In this case, selecting these developers could lead to overlap in knowledge.

As the number of developers in the ranking can be high (we checked projects with more than 600 developers [10]), we implemented an optimization algorithm to find the best combination of developers. Given a number of n developers (provided by the user), TIPMerge finds the n developers that maximize the joint expertise coverage. Figure 1 shows an example of the coverage analysis, where n=2 developers who are suited to perform a merge of branches in the Voldemort project. TIPMerge shows that the maximum coverage (91%) is obtained by *Siddharth* and *Felix*. Other combinations and their coverages are presented. The user can also inform that one or more developers are not currently available to perform the merge.

4. CURRENT RESULTS

TIPMerge is implemented in Java and analyzes projects versioned in Git, independent of the programming language of the project. Our approach can support a finer-grained expertise calculation at the method-level in Java projects. Most features have already been implemented. It is possible to obtain a ranking of suitable developers to perform a merge selecting two commits in two branches.

We evaluated TIPMerge (without coverage) [10] through a quantitative analysis of 28 projects, with a total number of 15,584 merges with at least two developers, and a qualitative analysis of two projects. We found that in 85% of the top-3 recommendations, we included the developer who actually performed the merge. In our qualitative evaluation, we investigated whether the developer who performed the merge (in the project) was the most suited, and evaluated these choices with the TIPMerge recommendations. We found that factors such as the person performing the most recent change, knowledge about specific parts of the code base, and personal preference, had an effect on who eventually performed the merge. The coverage analysis step is implemented, but has not been evaluated yet. To evaluate this step, a comparison of the solutions proposed by the optimization algorithm with the top-3 of TIPMerge ranking will be performed.
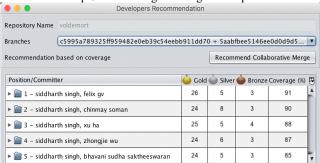

Figure 1. TIPMerge Ranking after the Coverage Analysis

5. CONTRIBUTIONS

In this research we propose TIPMerge, a novel approach that analyzes changes in branches, file dependencies, and the past history to recommend expert developers to merge branches. We implemented our approach in a tool that uses a medal-based ranking system to recommend developers. TIPMerge can help software teams in possible complex merge cases, where changes in key files can lead to direct and indirect conflicts. At the file-level granularity, TIPMerge is able to analyze any project versioned on Git. At the method-level granularity, our currently implementation is able to analyze Java projects. We evaluated TIPMerge recommendations using 28 projects. The results showed that TIPMerge can be useful to recommend developers to merge branches.

In this paper we also present a new step of our approach, the coverage analysis. This step uses an optimization algorithm to help in the decision of choosing the developers with maximum complementary knowledge to participate in a collaborative merge session, introducing the concept of joint expertise coverage. We plan to evaluate the coverage analysis by comparing the initial recommendation ranking of TIPMerge, with the recommendation after the coverage analysis and performing further interviews.

6. ACKNOWLEDGMENTS

7. REFERENCES

[1] Anvik, J., Hiew, L. and Murphy, G.C. 2006. Who Should Fix This Bug? *International Conference on Software Engineering (ICSE)* (New York, NY, USA, 2006), 361–370.

[2] Anvik, J. and Murphy, G.C. 2007. Determining implementation expertise from bug reports. *4th International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)* (2007), 2–2.

[3] Anvik, J. and Murphy, G.C. 2011. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Transactions on Software Engineering and Methodology.* 20, 3 (2011), 1–35.

[4] Bird, C. and Zimmermann, T. 2012. Assessing the Value of Branches with What-if Analysis. *ACM SIGSOFT Int'l Symp. Foundations of Software Eng (FSE)* (New York, NY, USA, 2012), 45:1–45:11.

[5] Bird, C., Zimmermann, T. and Teterev, A. 2011. A theory of branches as goals and virtual teams. *4th International Workshop on Cooperative and Human Aspects of Software Engineering* (New York, NY, USA, 2011), 53–56.

[6] Brosch, P., Egly, U., Gabmeyer, S., Kappel, G., Seidl, M., Tompits, H., Widl, M. and Wimmer, M. 2012. Towards semantics-aware merge support in optimistic model versioning. *Models in Software Engineering* (Berlin, Heidelberg, 2012), 246–256.

[7] Cavalcanti, Y.C., da Mota Silveira Neto, P.A., Machado, I. do C., Vale, T.F., de Almeida, E.S. and Meira, S.R. de L. 2014. Challenges and opportunities for software change request repositories: a systematic mapping study. *Journal of Software: Evolution and Process.* 26, 7 (2014), 620–653.

[8] Costa, C., Figueiredo, J.J.C., Ghiotto, G. and Murta, L. 2014. Characterizing the Problem of Developers' Assignment for Merging Branches. *International Journal of Software Engineering and Knowledge Engineering (IJSEKE).* 24, 10 (2014), 1489–1508.

[9] Costa, C., Figueiredo, J.J.C., Murta, L. and Sarma, A. 2016. TIPMerge: Recommending Developers for Merging Branches. *ACM SIGSOFT Int'l Symp. Foundations of Software Eng. (FSE) Tool Demonstration Paper.* (Seattle, WA, USA, 2016).

[10] Costa, C., Figueiredo, J., Murta, L. and Sarma, A. 2016. TIPMerge: Recommending Experts for Integrating Changes across Branches. *ACM SIGSOFT Int'l Symp. Foundations of Software Eng. (FSE)* (Seattle, WA, USA, 2016).

[11] Gousios, G., Pinzger, M. and van Deursen, A. 2014. An Exploratory Study of the Pull-based Software Development Model. *36th International Conference on Software Engineering (ICSE)* (Hyderabad, India, 2014), 345–355.

[12] Jiang, J., He, J.-H. and Chen, X.-Y. 2015. CoreDevRec: Automatic Core Member Recommendation for Contribution Evaluation. *Journal of Computer Science and Technology.* 30, 5 (Sep. 2015), 998–1016.

[13] Kagdi, H. and Poshyvanyk, D. 2009. Who can help me with this change request? *IEEE 17th International Conference on Program Comprehension, 2009. ICPC '09* (2009), 273–277.

[14] Koegel, M., Naughton, H., Helming, J. and Herrmannsdoerfer, M. 2010. Collaborative model merging. *ACM international conference companion on Object oriented programming systems languages and applications companion* (New York, NY, USA, 2010), 27–34.

[15] Lautamäki, J., Nieminen, A., Koskinen, J., Aho, T., Mikkonen, T. and Englund, M. 2012. CoRED: browser-based Collaborative Real-time Editor for Java web applications. *ACM 2012 conference on Computer Supported Cooperative Work* (New York, NY, USA, 2012), 1307–1316.

[16] De Lima Júnior, M.L., Soares, D.M., Plastino, A. and Murta, L. 2015. Developers Assignment for Analyzing Pull Requests. *30th Annual ACM Symposium on Applied Computing* (New York, NY, USA, 2015), 1567–1572.

[17] Linares-Vásquez, M., Hossen, K., Dang, H., Kagdi, H., Gethers, M. and Poshyvanyk, D. 2012. Triaging incoming change requests: Bug or commit history, or code authorship? *Software Maintenance (ICSM), 2012 28th IEEE International Conference on* (2012), 451–460.

[18] Matter, D., Kuhn, A. and Nierstrasz, O. 2009. Assigning bug reports using a vocabulary-based expertise model of developers. *IEEE International Working Conference on Mining Software Repositories* (Vancouver, BC, 2009), 131–140.

[19] Nieminen, A. 2012. Real-time collaborative resolving of merge conflicts. *2012 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)* (2012), 540–543.

[20] Shihab, E., Bird, C. and Zimmermann, T. 2012. The Effect of Branching Strategies on Software Quality. *ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM)* (New York, NY, USA, 2012), 301–310.

[21] Da Silva, J.R., Clua, E., Murta, L. and Sarma, A. 2015. Multi-Perspective Exploratory Analysis of Software Development Data. *International Journal of Software Engineering and Knowledge Engineering (IJSEKE).* 25, 01 (2015), 51–68.

[22] Da Silva, J.R., Clua, E., Murta, L. and Sarma, A. 2015. Niche vs. breadth: Calculating expertise over time through a fine-grained analysis. *2015 IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER)* (2015), 409–418.

[23] Da Silva Junior, J.R., Clua, E., Murta, L. and Sarma, A. 2014. Exploratory Data Analysis of Software Repositories via GPU Processing. *The International Conference on Software Engineering and Knowledge Engineering (SEKE)* (Vancouver, Canada, 2014), 495–500.

[24] Yu, Y., Wang, H., Yin, G. and Ling, C.X. 2015. Reviewer Recommender of Pull-Requests in GitHub. *IEEE International Conference on Software Maintenance and Evolution* (Victoria, BC, 2015), 609–612.

[25] Yu, Y., Wang, H., Yin, G. and Ling, C.X. 2014. Who Should Review this Pull-Request: Reviewer Recommendation to Expedite Crowd Collaboration. *Software Engineering Conference (APSEC), 2014 21st Asia-Pacific* (Jeju, 2014), 335–342.

[26] Zhang, T. and Lee, B. 2012. How to Recommend Appropriate Developers for Bug Fixing? *IEEE 36th Annual Computer Software and Applications Conference* (Izmir, 2012), 170–175.