Generating Accurate and Compact Edit Scripts using Tree Differencing

Veit Frick, Thomas Grassauer, Martin Pinzger[1], Fabian Beck[2]

Abstract: For analyzing changes in source code, edit scripts are used to describe the differences between two versions of a file. These scripts consist of a list of actions that, applied to the source file, result in the new version of the file. In contrast to line-based source code differencing, tree-based approaches such as GumTree, MTDIFF, or ChangeDistiller extract changes by comparing the abstract syntax trees (AST) of two versions of a source file. One benefit of tree-based approaches is their ability to capture moved (sub)trees in the AST. Our approach, the Iterative Java Matcher (IJM), builds upon GumTree and aims at generating more accurate and compact edit scripts that capture the developer's intent. This is achieved by improving the quality of the generated move and update actions, which are the main source of inaccurate actions generated by previous approaches. To evaluate our approach, we conducted a study with 11 external experts and manually analyzed the accuracy of 2400 randomly selected edit actions. Comparing IJM to GumTree and MTDIFF, the results show that IJM provides better accuracy for move and update actions and is more beneficial to understanding the changes.

Keywords: Change Extraction; Tree Differencing; Abstract Syntax Trees; Software Evolution

Edit scripts describe the differences between two versions of a source code file. Such scripts consist of a list of actions that, when applied to a given source code file, correctly transfers it from one version of that file to another. While edit scripts gnerated by line-based differencing algorithms are quickly generated and provide an overview, they are coarse grained and do not consider syntax information. Approaches such as GumTree [Fa14] and MTDIFF [DP16], compare the ASTs parsed from the source code files instead. This allows the algorithms to refine the granularity down to the level of single AST nodes.

Existing state-of-the-art approaches generate edit scripts that are correct in the sense of transforming one AST into another. However, in a manual investigation, we found that those edit scripts can consist of actions, especially *move* and *update* actions, that can be classified as inaccurate. Consider an edit script where every node of the original AST is deleted and every node of the new AST is inserted. This edit script always correctly transfers the original AST into the new AST, even if both ASTs are exactly the same. The actions of such an edit script would be correct but not accurate. We propose the following definition of an accurate edit action: An accurate action has to fulfill all of the following three criteria: The action has to be comprehensible, helpful, and the most simple solution. We found that over 55% of GumTree's and over 81% of MTDIFF's generated move and update actions are inaccurate

[1] Alpen-Adria-Universität Klagenfurt, SERG, vorname.nachname@aau.at
[2] University of Duisburg-Essen, fabian.beck@paluno.uni-due.de

doi:10.18420/SE2020_24

according to our definition given above. We argue that such a high misclassification rate significantly impacts the understanding of code changes, in particular of moved and updated code. Our approach is based on the GumTree approach and aims at improving the *matching* phase. We use the existing implementation of the algorithm by Chawathe et al. [Ch96] to create the final edit scripts. For improving the matching of AST nodes and subtrees, we add three matching strategies:

Partial Matching: We restrict the scope for the matching to selected parts of the source code, each represented by a subtree in the AST. We assume that most of the changes happen within such a subtree and only few changes happen between them. For instance, the source code of a method is more likely to be changed and moved within the same method. The partial matching approach is therefore to divide the AST into smaller parts that are individually matched. IJM is built as a combination of different matchers.

Merged Name Nodes: Name nodes are children of various other nodes like method or type declarations containing the name of their parent node as value. In order to prevent these nodes from beeing matched with other name nodes, belonging to different parents, IJM merges the name node together with their parent into one atomic node. This reduces mismatched name nodes and shortens the AST.

Name Aware Matching: GumTree, in its bottom-up phase, only uses the node type to determine whether or not two nodes can be matched. IJM addresses this by adding name-awareness to the bottom-up phase of GumTree. This is realized by considering the similarity of the names of the nodes in addition to their node types.

We evaluated and compared the accuracy and helpfulness of IJM to the state of the art approaches GumTree and MTDIFF. A study with 2400 randomly selected edits shows a higher accuracy for move and update actions without increasing the misclassification rate for insert and delete actions. A study with 11 independent external experts showed that they found more helpful for understanding the changes in a revision. Furthermore, an evaluation on 10 Java open source projects shows no increase in edit script size and runtime.

References

[Ch96] Chawathe, Sudarshan S.; Rajaraman, Anand; Garcia-Molina, Hector; Widom, Jennifer: Change Detection in Hierarchically Structured Information. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. SIGMOD '96, ACM, New York, NY, USA, pp. 493–504, 1996.

[DP16] Dotzler, G.; Philippsen, M.: Move-optimized source code tree differencing. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. ASE '16, pp. 660–671, Sept 2016.

[Fa14] Falleri, Jean-Rémy; Morandat, Floréal; Blanc, Xavier; Martinez, Matias; Monperrus, Martin: Fine-grained and Accurate Source Code Differencing. In: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering. ASE '14, ACM, New York, NY, USA, pp. 313–324, 2014.