Tool Support for Recurring Code Change Inspection with Deep Learning

Krishna Teja Ayinala, Graduate Student, Computer Science
Faculty Mentor: Myoungkyu Song, Computer Science

As software evolves, developers usually spend 65% of their time for fixing bugs, refactoring, adding new features to existing code. In these maintenance activities, they often make recurring changes, similar but different changes across multiple locations. Recent work finds that on average 75% of structural changes to mature software are recurring changes. To understand such code changes during peer code reviews, developers inspect diff patches, program differences between original and edited versions. For example, a developer inspect diff patches changed by other team members. Questions the developer asks are often of the following types: "What other code changes are made similar to this change?" and "Is there any location required to be changed similarly but is not modified?" Current code review tools accurately identify individual additions and deletions at a particular granularity by computing line-level differences per file. However, developers using code review tools need to inspect program differences on diff patches file by file despite recurring changes, a group of similar, related edits in multiple locations. They are left to manually examine individual edits, thus, leading to a tedious and error-prone process.

To address the problem, I propose a novel code inspection technique, Recurring Code Changes Inspection with Deep Learning (RIDL). My project focuses on demonstration of this approach by implementing a proof-of-concept prototype, RIDL, and integrating it with Integrated Development Environments (IDEs) as a plug-in. Specifically, first, to train a binary-class classifier, RIDL learns similar code fragments from a clone database. The clone database has been mined from over 25,000 open source programs. Given the clone database, RIDL trains the classifier by using cloned and non-cloned pairs of code fragments. Second, given a specified change, RIDL extracts an edit script by analyzing data and control flow context. Then, it forms change patterns of edit scripts by leveraging the classifier to (i) summarize recurring changes and (ii) detect potential change anomalies in the codebase. Thus, RIDL helps developers easily understand all related changes and focus their attentions on incorrect edits that are likely to pass unnoticed since these anomalies do not create any compilation errors.

Keywords: Code review, Deep Learning, Recurring Changes.