

An Evaluation of the Effectiveness of Program Slicing in Reducing Duplication in Source Code

Rob McGregor and Will Thomasson

PROJECT SUMMARY

A common problem in software engineering is duplicate source code, especially in large programs. The excessive code resulting from duplication makes maintenance and understanding of programs more difficult, and increases the chances of missing bugs while debugging. Finding duplicate code by hand is impractical, and it may be impossible to find all the duplicate code in a large program.

Researchers at the University of Wisconsin have developed an algorithm for identifying and removing duplicate code using program dependence graphs and program slicing [1]. Program slicing is a method of "abstracting from programs [that,] starting from a subset of a program's behavior, ... reduces that program to a minimal form which still produces that behavior. The reduced program, called a slice, is an independent program guaranteed to faithfully represent the original program within the domain of the specified subset of behavior" [2]. Using these techniques, we find duplicate code and place it into a procedure, replacing each instance of the code with a call to the procedure. This reduction of the overall program length makes understanding, maintaining, and debugging code much easier. We also identify clones (sections of duplicate code) using a technique that uses text analysis and compare this method to the University of Wisconsin method. We apply these approaches to several samples of code and find the resulting reduction in source code length.

Intellectual Merit

The University of Wisconsin method is unique in that it utilizes not only program dependence graphs, but also program slicing to detect clones in source code. Other methods rely on textual analysis to find sections of duplicate code, or clones. Program slicing allows the semantics of the code, not just the syntax, to be taken into account when searching for clones.

The University of Wisconsin claims that its method is superior to other methods of finding clones because it can detect intertwined clones, non-contiguous clones, and clones whose variables differ only in name; however, these claims are not supported by experimental results in their paper [1]. While in theory the Wisconsin method detects clones that other methods overlook, this advantage is insignificant if the number of clones found is not significantly greater than the number found by other methods. Because the Wisconsin method is more difficult to implement than more traditional methods of detecting clones, it is useful to know whether or not implementing the Wisconsin method is worth the extra effort.

We apply the Wisconsin method to samples of code using the program slicing tool Unraveler and a program dependence graph tool and evaluate the effectiveness of the Wisconsin method in detecting clones (and therefore in reducing duplicate code). We also use a clone detection tool, Duploc, that uses a text analysis approach and compare its effectiveness to the effectiveness of the Wisconsin approach in detecting clones.

Broader Impact

By evaluating the claims of the University of Wisconsin that their method is superior, we can determine if their tool is a worthwhile investment, both in implementation time and in cost. We can also determine the effectiveness of the Wisconsin method in improving understanding, maintenance, and debugging of code through removal of duplicate code. Finally, we can evaluate the effectiveness of other methods in similar improvements.