**The Problem Addressed**

Similarities are inherent in software and lead to repetitions, so-called code clones. We find clones within and across software systems. Removing clones from programs is often neither possible nor even desirable: Some clones play a useful role in a program (e.g., for performance or reliability reasons); Other clones occur because of the limitations of a programming language; Removal of some clones might conflict with design or business goals that cannot be compromised; Clones often result from beneficial standardization of software design, due to application of company standards, or standardized architectures and pattern-driven development on modern component platforms e.g., (.NET™ and JEE™); Uniformity of design is desirable despite inducing high levels of cloning.

Companies often manage families of similar software systems, formed by multiple variants (releases) of a system deployed to different customers during software evolution, or software Product Lines developed with reuse strategies. Such systems display high levels of cloning.

In current programming paradigms, whether we are in a single- or multiple-system development/maintenance situation, we are bound to deal with programs containing significant cloning. While cloning cannot be classified as inherently good or bad, the knowledge of clones, especially in large software, is useful for program understanding, maintenance, reuse and quality assessment.

**Our Solution: CM/CA Method and Tool**

Clone Miner (CM) and Clone Analyzer (CA) find clones in subject software system(s), help us understand the big picture of the similarity situation, filter clones that are of our interest in a given analysis context, and then zoom into all the details of such clones.

Other clone detectors mostly focus on detecting fragments of duplicated code, so-called *simple clones*. CM/CA, in addition to finding simple clones, also looks at the big picture, where simple clones are possibly part of a bigger design-level replicated program structure. We call these higher-level, large-granularity repeated program structures *structural clones*.

**Clone Miner (CM)** automates clone detection as far as it is possible without user’s intervention [1][2]. First, CM looks for simple clones. Then, CM uses data mining to identify configurations of simple clones that form design-level similarities. CM uses efficient algorithms and heuristics to extract the most useful cloning information in the shortest time.

The follow analysis of clones is done with user involvement. **Clone Analyzer (CA)** applies visualization and abstraction to filter information produced by CM [4]. CA is equipped with clone-query system that provides a user with flexible way to filter cloning information. CA displays multiple views of cloning information as navigation tables, graphs, overview charts, and diff tool.

CA is language independent, whereas CM is easily configurable for different languages, as it works with the lexical tokens rather than parsed structures of the subject software. Currently, our tools can work with Java, C/C++, Perl and VB.net.
The Benefits

- During maintenance, the knowledge of clones aids in controlling the impact of change. It can reduce update anomalies, and control the ripple effect of making changes to the system.
- Knowledge of the design level similarities can lead to better program understanding and possibly identifying areas in which the system can be improved in terms of design and quality of code. These similarities are potential candidates for code refactoring which will enhance the maintainability of the system. Re-engineering of existing legacy code is aided by such design similarities.
- Large granularity, design-level similarity patterns often create opportunities for reuse of design solutions within a given system, or even across similar systems. This form of reuse is natural and enhances current architecture-centric, component-based reuse methods. A meta-programming technique of XVCL (http://xvcl.comp.nus.edu.sg) developed in our lab has been designed to reap benefits offered by similarity patterns. We can represent similarity patterns (and any kinds of counter-productive repetitions) as compact, easy-to-understand and easy-to-work-with XVCL-enabled generic solutions. We got evidences from lab studies and industrial applications that XVCL-enabled reuse can considerably reduce development and maintenance effort [7][9][10][11].

References


Contact: Stan Jarzabek
Department of Computer Science, School of Computing, National University of Singapore
Computing 1, #03-68 Law Link, Singapore 117590
email: stan@comp.nus.edu.sg; http://www.comp.nus.edu.sg/~stan
fax: 65-6779-4580; tel: 65-6874-2863 (office) 65-96255863 (mobile)