ABSTRACT

Software nowadays is usually not developed from scratch. Programmers tend to reuse artifacts from other components or systems in every step of software development, from ideas, requirements, designs, algorithms to concrete code. This practice happens in both open-source and closed-source environments. It is due to the fact that multiple software programs could provide common functionalities or share common specifications, designs and/or algorithms or, developers might use the same libraries/frameworks leading to the same code usages or common programming idioms in their source code. This practice has the benefits of reducing development time and cost, shortening length of code and improving the quality of the products since the reused artifacts are usually well-studied, well-designed, well-tested, and compact. It also introduces *repeated software artifacts* both within a software project and across multiple software projects. When these artifacts evolves, for supporting new features, improving the performance, fixing bugs, adapting changes in the libraries/frameworks or improving the readability, etc., they might also evolve in the similar ways leading to *repeated changes*.

This dissertation focuses on the *repetitiveness* of software artifacts at the source code level. Specifically, this work studies the *repetitiveness of code and code changes* in a *large corpus*. That knowledge is leveraged to recover the code specifications and programming patterns, and code change patterns which can be used to prevent bugs, suggest changes and recommend fixes. The key ideas behind this work are: (1) in a large corpus, code that conforms to specifications and programming patterns appears more frequently than code that does not, and (2) similar code is changed similarly and similar code could have similar bugs that can be fixed similarly.

We have developed different representations for software artifacts at source code level, and the corresponding algorithms for measuring code similarity and mining techniques. We also have built program differencing framework for analyzing changes in software evolution. Our empirical evaluation shows that our techniques can accurately and efficiently detect repeated code, mine useful programming patterns and API specifications, and find actionable insights to ease maintenance tasks.