Successful software constantly evolves. Most programmers work on projects they did not start. Most companies spend more on maintaining old systems than on building new ones. This is good, because it shows that we build software that is worth keeping and maintaining. But it also means that change is the heart of software development.

Sadly most programming tools treat change at a very low level. Programmers make most changes with text editors, and tools—such as version control systems—tend to focus on changes to lines of text. Even if some tools treat change at a somewhat higher level, it is ad hoc and not integrated with other tools. Tools that apply refactorings (i.e., changes that affect internal software structure but not its external behavior) or that add features to programs can make bigger changes automatically, but these changes are not understood by other tools, e.g., version control. Programming is mostly about change, but contemporary tools do not understand changes very well. This makes programming more expensive, time-consuming, and error-prone than it should be.

Program transformations unify all of these change activities. By placing transformations at the center of software development, we can begin to lay the foundations of scientific theories and to build tools that will improve our abilities to automate and simplify software evolution. Our work aims to (a) change the way people program, (b) create a platform for research for the next 10 years, and (c) produce results that become standard software development practice in 15 years. To develop a unified approach based on transformations, we will generalize work in five domains that have used transformations: refactoring tools, software product lines, bug fixing, test evolution, and improving security. We want both average and expert programmers to be able to write, script, modify, and replay their own transformations. We want programmers to think about programs as compositions of transformations, and to automate as many of these transformations as possible. We will develop a change-oriented programming environment (COPE) that integrates our individual tools with version control, to enable an understanding of change at higher-levels of abstraction. In short, we want to start the inevitable process of revolutionizing software engineering from manual development to semi-automated development of software.

**Intellectual Merit:** The future of software development lies in mechanizing tasks that are expensive, error-prone, and time-consuming. Even partially automating tasks or having programmers express their customized tasks as program transformations will be a significant step toward altering the way we think about, teach, create, reuse, and understand programs. Specifically, we consider five activities: (1) analyze what changes programmers typically make and how they perceive, recall, and communicate changes, (2) automate transformations to make it easier to apply and script changes, (3) develop tools that compose and manipulate transformations to make it easier to reuse them, (4) integrate transformations with version control to provide better ways for archiving and understanding changes, and (5) develop tools that infer higher-level transformations from lower-level changes such as code edits to enable our new environment to work even for programmers who use old tools to make changes. COPE will deliver a rich transformation-aware toolset that synergistically integrates all these activities.

The PIs are uniquely qualified to carry out this research. Johnson has done pioneering research on refactorings that has now become standard in software development practice. Batory has created transformation-based theories and tools for software product lines. Dig has developed a prototype of transformation-centric software evolution. Marinov has developed tools for testing refactorings and for the evolution of test code. Bailey has improved the interfaces of software tools. Additionally, the PIs have published jointly for years and have strong interactions with industry and academia (see attached support letters).

**Broader Impacts:** By embracing our proposed view that change (transformations) lies at the heart of software development, US software education and industry will take a critical step toward maintaining its supremacy long-term. This project has the potential to revolutionize how programmers develop software, and to significantly reduce the cost and increase the quality of developed code. The PIs’ previous work on refactorings and design patterns have produced results that are already included in modern integrated development environments used by millions of developers. We will disseminate our results through presentations, books, publications, open-source code, industrial collaborations, and educational activities. In the last five years, the PIs have involved over 40 undergraduate and over 20 minority students in research projects. The PIs have started to revamp the software engineering curriculum at their institutions to emphasize the science of change in large code bases and plan to share the curriculum across academia.

**Key Words:** Software evolution; software maintenance; program transformation; version control; refactoring; IDE