Understanding and Modifying Procedural Versus Object-Oriented Programs: Where Does Domain Knowledge Help More?

Thomas LaToza (Student Member CSS) and Alex Kirlik (Member CSS)
Department of Psychology and the Beckman Institute
University of Illinois at Urbana-Champaign
<latoza, kirlik>@uiuc.edu

Software bugs are an ever-increasing societal problem as computers become more prevalent in our homes and workplaces, and programs grow in size and complexity. In response, many in the software engineering community have advocated a shift in the way programs are written: away from procedural (plan-like) code (e.g., Pascal, Fortran) and toward more declarative (map-like) Object-Oriented (OO) code (e.g., C++, Java). A central claim is that more declarative, map-like code allows developers to better deploy their knowledge of a problem domain (what the code is 'about') than does procedural code. The latter typically consists of a compact plan for solving a problem (such as a recipe or driving directions), rather than a more declarative description of domain entities and their relationships (OO). To test this claim, we conducted an experiment requiring experienced programmers presented with procedural and OO code *isomorphs* of the same algorithm to perform code modification tasks, and crossed this manipulation with whether or not participants were primed on the domain knowledge of the actual problem solved by the algorithm (scoring ten-pin bowling). In contrast to the claims of the OO community, our findings revealed that priming domain knowledge helped those modifying procedural, rather than OO, code: The procedural group whose knowledge of how bowling is scored was primed created significantly fewer bugs than the non-primed procedural group, while priming the OO group led to perhaps even slightly more bugs. This finding suggests that, when trying to modify or "tweak" problem solutions, knowledge of the problem domain is more important when trying to amend existing procedural solutions (e.g. a route given as a list of directions) than when trying to amend more descriptive, declarative solutions (e.g. a map to a destination). Implications for both software engineering and cognitive science will be presented.