Abstract

ET: Enhanced tool
The effectiveness of <proving the correctness of code> in supporting the design of <software implementations> has been demonstrated. An enhanced method is described for the design of <software implementations> based on <first proving that individual procedures satisfy the corresponding abstract functional specification, in order to simplify the succeeding problem of showing that the abstraction serves the program’s intent>. Examples are provided confirming the effectiveness of its support for <proving the correctness of code> in design.

Why ET?
Based on the text and citations, it appears that researchers had already conceived of proving code correctness. This paper constitutes an improved method of proving code’s correctness.

Question - [Method/means of development]
<model-type / solution strategy>
What is the most tractable way of proving code correctness?

<artifact-type>
The artifact being produced is a software implementation. The question is the best way to formally ensure that the implementation serves the program’s intent.

Results - [Procedure / technique]
<model-type / solution strategy>
The author argues that designers should break the problem of proving correctness into two parts. The paper focuses on the first of these two parts: showing that each individual procedure’s implementation (in the form of concrete variables and code) satisfies the functional specification of an abstraction. The author believes that because the abstraction is succinct and clear, that it then will be easy to perform the second step: showing that the abstraction appropriately serves the program’s overall intent.

Validation – [Example]
<model-type / solution strategy>
The author demonstrates how his strategy can be used to show that a set’s procedural implementation matches the set’s abstract functional specification.

It should be noted that the author’s approach has a serious limitation which is not mentioned in his abstract (nor does it have a home in Newman’s template). Specifically, the author closes with a comment that “the restrictions suggested in this paper make it impossible for local variables of a class to be updated except by the body of a procedure local to that very same activation of the class.” In other words, if the software does not perfectly respect encapsulation boundaries, then all bets are off. In the real world, people do occasionally violate abstraction (since not all programmers adhere to best practices). It might be worth highlighting to the paper’s reader that this violation comes with the price that it becomes much more difficult to formally prove the correctness of a program; this might help to reinforce good practices, as well as highlight the limits the proposed method’s applicability. Hence, this is a significant caveat that might be worth mentioning in the abstract, even though it does not fundamentally alter the validity of the author’s work.