
Abstract
XH: Experience and/or Heuristic
Studies reported here of <application repair> supported by <architectural models> generate a number of findings concerning <the feasibility of applying application repair to open source systems>. They indicate that <improved modularity> is (feasibly) met by <architectural repair>.

Why XH?
The authors explicitly state that they tried out architectural repair to see whether it was effective and practical. Along the way, they define a simple model to guide their assessment of the experience, but the model really seems too simple to be the main contribution of the paper. The authors had already discussed forward and reverse architecture repair techniques in a separate paper. The purpose of this one was to add validation by experience on real system.

Question - [Feasibility]
Is it practical to think that we can effectively repair the architecture of an open source application?

Results - [Report]
The authors fixed a variety of implementation-design inconsistencies. Many were fixed by updating the design documentation to match the implementation. There seemed to be a handful of missteps in cutting up the software as they were planning how to achieve the repairs, but they eventually figured out what the software was supposed/trying to be doing.

They offer no list of design heuristics or earth-shattering findings, and it wasn’t exactly what generalizeable lessons they learned. However, their experience did confirm that architectural repair is feasible, if that concept also allows simply updating the design documentation to agree with the implementation (which, incidentally, is consistent with Paras’s paper on “faking” a rational design process). They also seem to think that they did a good job of eliminating unnecessary connections among modules, which they call improved modularity.

Validation – [Experience]
The authors tried out the proposed techniques on two large open source systems (Linux kernel and Vim text editor). They applied forward architecture repair (updating the implementation to match the design) and reverse architecture repair (updating the design to match the implementation). To achieve this, they relied on models of the software: system (essentially modeling the source code), conceptual (essentially modeling the intent), and concrete (essentially modeling procedures and their relationships). These models seem to be linked by a fourth model, called the layered model.