Evolving Legacy System Features into Components

This paper describes a 3-step methodology for evolving a system into components based on the features the system implements. I elaborate on the steps and describe them as follows:

- Group the available test cases by feature
- Run the test cases (by feature) using a code profiler to pinpoint all the code being executed for a particular feature.
- Refactor the identified code into a component
- Replace the identified code with it's component and compare the results of the same test cases used to identify the code - results should be the same
- Compare pre- and post-evolution maintenance costs.

The authors applied their proposed process to the Master System (AMS), a 14 year old product developed by American Financial Systems. After describing the case study and how the methodology was applied to AMS, the authors describe some lessons they learned. Two of those lessons are applicable to the ISVis adaptation project:

- The authors conclude that features are good candidates for evolution into components if they "change often, are concentrated in fewer functions, or depend on or share global variables as a means of communication." While the "display feature" of ISVis might not necessarily change often, it was chosen as the primary candidate for evolution because it is currently nonfunctional on modern machines and this holds it back from being used or updated.
- In measuring success the authors comment that "the true measure of a successful evolution methodology is in reduced future maintenance costs." From that perspective, the adaption of ISVis to have a plugin-style interface for its display code will certainly be a success since that accomplishment will drastically reduce the cost of maintaining that portion of the code.

Two key assumptions made are that (1) the system is coded in a language for which a code-profiling tool is available and (2) that the legacy system has regression test suites. ISVis satisfies (1) but not (2) and thus could not use the methodology described without first developing a reliable set of test cases. This raises another issue: how to reliably test a GUI based application. Further reading on that topic is required to help develop a more reliable process in refactoring ISVis.