Aging of a Data-Intensive Legacy System

The goal of the research described in this paper is to identify the symptoms of an aging legacy system. It is addressed to practitioners because "it advises them what to measure to monitor aging symptoms, what operations are necessary to treat the symptoms and what the expected efficacy of the operations is."

The symptoms described are as follows:

Pollution
Components that are no longer used by the users. Definitions and examples are given to describe duplicate programs, obsolete programs, sourceless programs, useless components, dead data, and dead code.

Embedded Knowledge
This is knowledge about the system that can no longer be derived from existing documentation. Definitions and examples are given to describe incomprehensible data and modules and missing capacities.

Poor Lexicon
"This is present when the names of variables and components have little lexical meaning or are in any case inconsistent with the meaning of the components they identify."

Coupling
"The programs and their components are linked by an extensive network of data or control flows." Determining this involves inspecting the code for pathological files, control data, and module complexity.

Layered Architectures
This occurs when there are several different solutions spread out across a system's architecture. This symptom can be identified by looking for useless, obsolete, temporary, permanent, or anomalous files. Also look for semantic redundant data and superimposed data structures. Specific to database applications, we should look for computational redundant data and structure data.

After defining the symptoms to watch for the paper describes the lessons learned from reengineering a legacy system.

- *Lesson 1* - Before renewing an old software system, it is wise to clean it thoroughly of the pollution that has accumulated over the years
- *Lesson 2* - The effort spent on restoration may result in a trade-off between the quality targets desired for the renewed programs and the resources available for the restoration process.
- *Lesson 3* - Extracting the knowledge incorporated in the programs is a long and costly activity.
- *Lesson 4* - It is best to check the lexical quality of the programs frequently
- *Lesson 5* - Coupling is one of the most harmful and costly symptoms of aging of a legacy system
- *Lesson 6* - Analysis of the metrics indicating the symptom of layered architecture is useful as a means of understanding why the initial design of the legacy system was not adequate to deal with evolutions in the application domain.

Lessons 1, 3, 4, and 6 directly apply to our eventual refactoring of ISVis and should be kept in mind as we move forward with the reengineering process. ISVis exhibits all of the aging symptoms described in this paper to varying degrees, suffering most severly from embedded knowledge. Steps should be taken to improve the general understandability of the code.