Developing a new Data Archive in a Time of Maturing Standards

by Vernon Leighton[*]

Abstract
Today, there are many customs, standards and standard applications which groups just beginning a data archive can choose to adopt or not adopt. This presentation discusses the strategies that the Cultural Policy and the Arts National Data Archive (CPANDA) team used in the development of the archive, including the criteria by which established standards and practices were evaluated. The choices made by the team as they worked to develop the archive will also be discussed.

Introduction
This past year, the CPANDA team undertook to design and implement a new data archive. In developing a new archive, one important consideration is to do so in a cost effective way. Naturally, certain features are desired, but one does not want to waste effort through unnecessary duplication. Like everyone else, we want to get where we want to go, without doing it the hard way.

The primary guiding principle, then, in our data archive development was to design the archive with the features we desired while limiting the amount of duplication of effort required on our part. This is not a novel idea, but it is better to have a sound idea than a novel one. Because we began our archive development long after many other individuals and groups, including many of you in this room, had already done so, our first strategy was to evaluate earlier efforts to see what we should adopt or build upon, and what we should create ourselves. Our question was: Which practices should be adopted, and which, like a Mark Twain classic, are better talked about than enjoyed? Our second strategy was to follow, as much as possible, the DRY Principle as articulated by Andrew Hunt and David Thomas (Hunt and Thomas, 2000, p. 26-28). DRY stands for "Don't Repeat Yourself." This principle states that you should have only one canonical version of information about both the data and the software, and that all other forms of the information should be regenerated from the canonical form and should be disposable. These are the strategies we used for developing a data archive in a time when many other efforts had already come before us.

The first strategy for controlling duplication of effort was, not surprisingly, to try to build upon earlier efforts. For this reason, evaluation was an important component during the early stages of development. The criteria we used in these evaluations were robustness, performance, and institutional support. Robustness means how much of the realm of all likely cases and circumstances the item could handle. For software, this concept includes its portability to a variety of platforms and programming languages. Performance means: first, has the item been implemented at all? and secondly, how well, how simply, and how quickly does the item work? Institutional support is a measure of how committed organizations are to the support and future development of the item in question.

Our second strategy for controlling duplication of effort was the DRY principle, which is, as stated by Hunt and Thomas, "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system" (p. 27). That single representation is then used to generate data products and even software used in the processing and publishing of the data sets. The DRY principle has helped reduce the amount of recoding and reformatting at CPANDA, however imperfectly applied.

Evaluations
I and other CPANDA members spent a great deal of time evaluating standards, practices and software for possible adoption into our archive. They include: metadata (codebook) format standards, standards for controlled vocabularies to describe data sets, software for analysis and archival management, full text indexing software, and software used to create codebooks in the DDI format.

Case I: Codebook Formats
One of the first decisions made by our group was what format to use to store the metadata for the data sets. The choice was not difficult, and we immediately selected the DDI format for the codebook presented to the users. Its advantages are many and obvious. DDI uses XML, which has many positive features. As far as robustness, eXtensible Markup Language (XML) is well understood, uniform and extensible. Regarding performance, there are many applications designed to parse and process XML. Institutional support for XML is strong, with many groups working on using and improving the standard. The DDI

itself is robust in that it handles many different aspects of data sets. It has institutional support through many IASSIST members, including ICPSR, NESSTAR, etc. These organizations are committed to supporting the DDI and extending it to cover even more types of data sets.

Performance is one area where the DDI has problems. Because of the richness of its cross-references and structures, applications that use the DDI are easier to design if they treat it as a static object. This object orientation favors the use of tree representations of the document, such as the Document Object Model (DOM) over event driven representations, such as the Simple API for XML (SAX). Because of the large size of these XML documents, the use of built-in DOM methods for data retrieval can be quite slow and can use a great deal of memory. In our own operation, the naïve use of built-in DOM data retrieval calls was unacceptably slow, and the use of superior data management resulted in a 300-fold increase in performance.

For example, we have a program that pulls from each variable in a codebook the text of the question in the survey that relates to that variable. The question text is then loaded into a relational database for search and retrieval functions. If the function repeatedly obtained all variables through the DOM method $codebook → findnode("/codebook/dataDscr/var"), the algorithm runs very slowly. We solved the problem by creating a codebook object that loaded most variable-level information sequentially during initialization into a hash data structure. The repeated access to the hash was up to 300 times faster than the DOM method as implemented in libxml2.

Other formats that we could have chosen for variable level metadata include the Triple-S format in XML (Hughes, Jenkins and Wright, 2001), and a variety of proprietary formats developed for individual software applications, such as SPSS and SDA. All these formats suffer from a lack of elements for bibliographic information about the whole data set, lack a richness of structure for describing many possible types of data set layout, lack tools to parse them in many languages on many platforms and lack institutional support to address those shortcomings and extend them in the future outside of the groups that created them. On the positive side, these formats tend to be based on plain text, and can usually be parsed in a straightforward manner. Because of their simplicity, some of them offer greater processing speed than some implementations of the DDI.

Skeleton in Closet

When I stated that we chose the DDI for our format to present to users, I confess that I was misleading you. When we began the project, we chose the Survey Documentation and Analysis (SDA) software to manage the online analysis of our data sets. At first, we had difficulty in getting XML parsing tools with acceptable features loaded onto our server. In order to get moving, we chose to represent the variable level data initially in SDA's DDL format.

The data set processing tools were designed so that the codebook was an object accessed through a standard set of methods. The codebook object was originally implemented on top of SDA's DDL format, and the XML DDI format was generated by one of the programs as a transformation of the DDL. Therefore, the authoritative XML version of the codebook was not the canonical version used by the archive software; instead, the XML version was a temporary product subject to being discarded and rebuilt. This situation had to change, because so long as the XML version was a disposable product, no changes could be made to the canonical version using its rich set of fields for data description.

Case II: Controlled vocabularies

CPANDA provides bibliographic descriptions of each of the data sets that we manage, which means that subjects and descriptive terms must be used to assist with the bibliographic control of the data sets. Because we need a controlled vocabulary of descriptive terms, CPANDA staff evaluated a variety of possible vocabularies. After discussions with ICPSR, it was decided to work with them to add to their thesaurus new terms related to Cultural Policy and the Arts. By working with them, we would be able to tap into a large vocabulary of terms related to surveys and social science research, and they would increase the base of organizations that use their terminology. It was also decided to use Library of Congress Subject Headings in addition to the ICPSR terms, to allow bibliographic access by those familiar with that more commonly used vocabulary.

The advantages of the collaboration with ICPSR are 1.) we do not have to create our own thesaurus of terms, 2.) we can use a vocabulary already rich in terms related to social science data sets, and 3.) we can add to it, enabling us to customize it somewhat to our own needs. It is also robust and has institutional support.

Case III: Software for Analysis and Archival Management

Several systems have been developed to perform analysis of data sets and manage data archives. Because analysis is quite exacting and difficult to program well, it would require a great deal of effort to duplicate. There are also many logistical issues related to the management of a web site, and adopting software that would manage the site would again save effort.

We examined the Nesstar system (Ryssevik and Musgrave, 2001) for both its analysis and archive management functions. Most important, it passes the first hurdle of performance: it has been implemented in a production mode, not just experimentally in beta mode. It is a

well built system which could readily accomplish our goals. It uses the XML DDI codebook. It allows full text searching for variable discovery. It deals with many archival management functions in a robust fashion and is institutionally supported by the EC. We chose not it adopt it primarily because it is on the Microsoft NT platform, and we have committed ourselves to the Unix platform. So, we did not select it, but not because it came up short in our evaluation.

We examined the reports on Harvard's Virtual Data Center (Altman, et. al, 2001), and were quite impressed with their plans. If they succeed in making the system robust and stable in the way they have envisioned it, it will be quite an attractive option on a Unix-like platform. Being open source, it will be robust in the sense that if it lacks a feature and the local archive has the ability to create that feature, the feature can be added. Many useful archival management features and a flexible, distributed architecture are in development. The software will use the XML DDI. It has the institutional support of Harvard, MIT, and the NSF.

In the performance realm, however, we feel that right now it is at too preliminary a stage of development to be adopted. It seems to only have been implemented in an experimental stage, without a track record of stability and portability. It does not seem to be robust yet. At last report, it could only run on some versions of Linux. These details may have changed by now, but as of the time of the evaluation, we could not commit to its use. We remain interested and hopeful that we might adopt it in the future. However, it seems to be not quite there yet.

We did not investigate Data Ferret from the U.S. Federal Government as closely as perhaps we could have. The interface presented to the users did not appeal to us, and we decided that we would not consider it further.

The software that we in fact chose to license for the analysis of the data sets online is the SDA software from the Computer-assisted Survey Methods Program at Berkeley (http://sda.Berkeley.edu:7502/). Although it has limitations, such as not having source code available, it does have virtues, like the fact that it already works well in production mode. One can operate it within a data archive as a black box analytical engine and design one's own interface to initiate online analysis. The results of the analysis can be captured and customized. It does not have archival management functions, but there is nothing to prevent those functions from being added to the interface that accesses SDA. Depending on how VDC is implemented, it is not inconceivable that SDA will be complementary to it.

Case IV: Full Text Indexing

On our site, as was already explained, we have developed a feature that allows users to search the full text of questions, variable labels and other variable notes for terms of interest in the variable discovery process. In order to accomplish this, some software must index the appropriate text and provide operators for users to query the index. It is possible to create ones own full text indexing application, but doing so would require a fair amount of effort, while others have already created such tools. Here again, we evaluated a variety of products before settling on one option. We may change our decision in the future.

Many database management systems have full text indexing capabilities. We decided against using a major commercial product like Oracle because of price. The price of Oracle is not just the licensing fees, but, because it is quite challenging to administer, one has to add the cost of hiring experienced personnel who are able to manage it. That personnel cost is perhaps higher than the licensing. Otherwise, Oracle's functionality, robustness and institutional support would make it acceptable.

We settled upon MySQL for our database management system. It is open source and free software. It is popular and has a large following in the open source community. It is fast and scales well, even if it does not implement all SQL standard features. It has a built-in full text indexing function. Unfortunately, the built-in indexing has very limited features and operators available.

Because the software that we chose has limitations, we evaluated products that might supplement the product used. Several major Internet Search Engines offer indexing software that can interface with database management systems. We discovered that these companies will license their products for modest sums if they index web pages, but they immediately begin charging much more if one uses the interface to index databases. Perhaps they know that such interfaces can be quite commercially lucrative and charge accordingly. In any case, many of these search engines are not available on many platforms and with APIs to many programming languages.

As a result of these evaluations, we have settled for the moment on the built-in features on MySQL and we hold out hope for improved functionality in the 4.0 release.

NOTE POST-CONFERENCE: In July of 2002, we found that the Swish-E search engine was compatible with our web programming environment. We are now using the Swish-E search engine for full text indexing of our MySQL database, by extracting the records with a PERL program and wrapping them in XML before handing them to Swish-E. The coordination of the Swish-e index and the MySQL database is handled using the Swish-E PHP class developed by Olivier Meunier. Again, we are able to leverage the sophisticated indexing technology of Swish-E and the data retrieval power of MySQL.

Case V: Codebook Editors
Here I would just like to report that we evaluated and attempted to use the MADDIE software developed at the University of Minnesota to create DDI conformant XML codebooks. The result of this experience is that we do not plan to use MADDIE in the near future for codebook editing. The primary problem as of six months ago with MADDIE is that it is not very robust. Many features that one comes to expect from text editors in general are either not present or do not perform acceptably.

I believe that the problem with the MADDIE project is that the project has tried to reinvent the wheel. They have built their own text editor to manage XML DTDs. They have had to reimplement many of the features that come standard on a myriad of other text editors. Because of the enormous amount of reimplementation involved, necessarily developed on a limited budget, the project has been overwhelmed and the product suffers from a lack of robust features. Future efforts should perhaps try to build upon a text editor that already has a rich set of functions but which is open source and programmable. An editor like EMACS could be given custom extensions to offer XML specific features based on a DTD or Schema. One would then leverage the other editing features already developed by the open source community.

That having been said, I would like to thank Wendy Thomas, Bob Wozniak, and Hicham Berrada for letting us use their code. They have put a great deal of work into its development, and they should be appreciated for their efforts, despite the results not being what one might hope for.

Evaluating Evaluation
In all of our evaluations, CPANDA sought to find workable solutions to data archive needs by adopting or building on the work of others. Those efforts need to be ongoing. We explain our criteria and findings in order to foster discussion and reevaluation and welcome criticisms.

The DRY Principle
The DRY Principle is quite common throughout computer science. Programs have functions and included libraries to reuse code. Object brokers such as CORBA, COM and SOAP are designed to reuse executables. Configuration files allow compiler options to be specified in one unambiguous location. The principle can be pushed to advanced levels in cases such as compiler compilers and automatic code generators, where the source code itself is a temporary duplication of the base information stored in one unambiguous location.

The DRY Principle was successfully implemented at CPANDA with regard to codebook metadata management and Web page creation, but it was not so successfully implemented in terms of dynamic, automatic construction of the software itself. In the data realm, we have established a canonical version of the codebook and build all other codebook products dynamically from it. See Chart 1 for a diagram of that process. On the Web site, we use server-side database access to dynamically generate pages customized to the user's context. Automated source code generation has not been systematically pursued, but should be placed on the schedule.

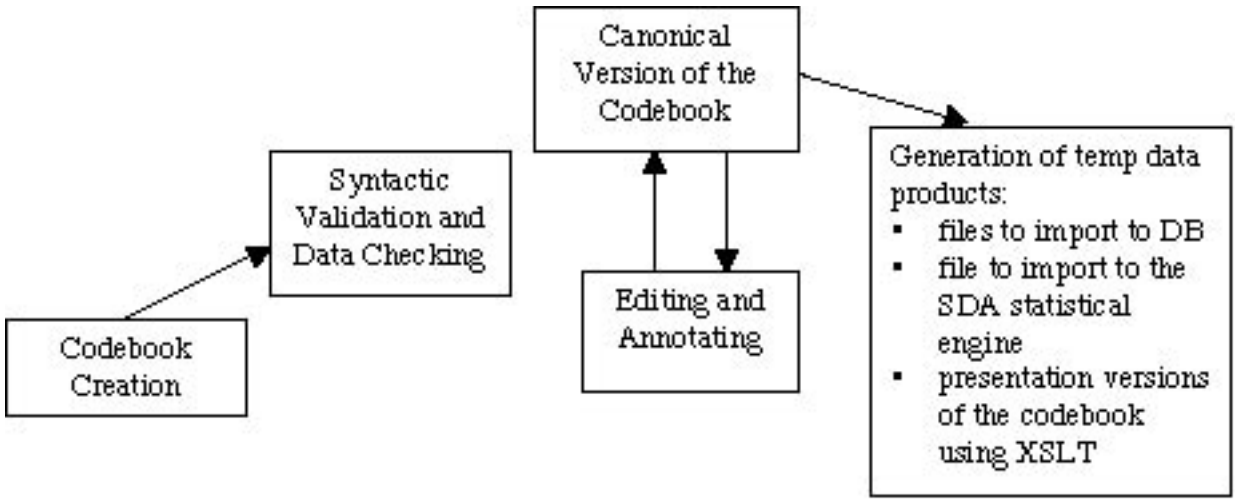

Chart 1

Conclusion
The concept of limiting duplication is one that is worth making a priority in the development of any software project. The two strategies of adopting previous efforts and minimizing duplication have allowed us to develop our data archive in an effective and efficient manner.

NOTES:

Altman, M., et. al. 2001. "A Digital Library for the Dissemination and Replication of quantitative Social Science Research," Social Science Computer Review, (v. 19, no. 4, Winter, 2001), p. 458-470.

Hughes, K., Jenkins, S., Wright, G. 2000. "Triple-S XML: A standard within a standard." Social Science Computer Review, v. 18, n. 4, Winter 2000, pp. 421-433.

Hunt, A., Thomas, D. 2000. The Pragmatic Programmer: from Journeyman to Master (Reading, MA: Addison-Wesley, 2000), p. 26-28.

Ryssevik , J., Musgrave, S. 2001. "The Social Science Dream Machine," Social Science Computer Review, (v. 19, no. 2, Summer 2001), p. 163-174.

* Paper presented at the IASSIST Conference, Amsterdam, May 2001. H. Vernon Leighton, Cultural Policy and the Arts National Data Archive, Princeton University, vleighto@Princeton.edu.