Dissertations and databases:

the historian as software engineer

Gervase Phillips

Manchester Metropolitan University

This article argues that historians have always been closer to programmers than has perhaps been recognized, and that historical software projects undertaken within the framework of the traditional third-year dissertation are useful training not just for the potential historian, but also for the potential software engineer.

The historian of tomorrow will be a programmer or he will be nothing.
(Emmanuel Le Roy, *Le Territoire de l'historien* (Paris, 1973))

The final weeks of the summer term see the normal frantic rush of second-year students looking for a suitable topic for a dissertation. Traditionally, the aim is to produce a piece of work from ten to fifteen thousand words, comprising a significant amount of original research, drawn from primary material, and demonstrating the ability to formulate and sustain a cogent argument, evaluate evidence and communicate original ideas. All this should be within the conventional, scholarly framework, logically structured and scrupulously referenced.

With increased student numbers and strained traditional resources, one possible solution to the problem of finding a topic is a software project. This choice utilizes IT skills in a manner which the student can immediately appreciate, and integrates IT into the mainstream of the history student's experience. There is no real reason why a software project cannot be considered by the same criteria as a normal dissertation, and open to similar judgements: essentially historical judgements. But the tight methodological rules which have governed the writing of academic history can be of more than passing relevance to the software engineering industry itself. As the use of IT becomes more widespread in the teaching of history, and, as Le Roy predicted, more historians become programmers, it is important to avoid some of the pitfalls into which the burgeoning software industry fell. Interestingly, those rigorous methodological rules insisted upon in the production of history dissertations are not only good practice for potential historians, but also for potential software engineers.

The production of software was, and to some extent still is, bedevilled by the idiosyncrasies of individual programmers. Since there was very rarely a single right way of writing a program, a single correct algorithm to solve a given problem, much code has been produced with features reflecting only the individual proclivities of the programmer. There is a parallel here with the 'production' of history. Starting with the same problem and the same primary sources, different historians have produced widely differing interpretations of the same events. Thus American historians, influenced by political beliefs, sectional loyalty, or the social mores of the times in which they wrote, have given a plethora of differing accounts of the outbreak of the American Civil War. The difference, *vis-à-vis* the production of software, however, is that there is an established methodological framework within which history is written. The conventions of historical writing, such as footnoting and referencing, facilitate comprehension. The origins of arguments, interpretations, and particular lines of thought can be traced back to their sources. New interpretations, revisions, and previously unknown original sources can all be incorporated into the existing corpus of historical knowledge within the conventional methodological framework.

Software engineers, on the other hand, had no such framework or conventions. If a program needed to be revised, up-dated or debugged, the programmer was faced with a mass of code, possibly in a low-level language, where it could be very difficult to identify what any particular line was doing at any stage, making the amendment of the code immensely problematic. Given a lapse of time, even the original authors of code could be unable to unravel their own work', and even greater problems ensued when different programmers had to struggle with other peoples' unannotated and undocumented code. Knowledge of a historian's sources makes it easier to 'debug' his or her work. What is required is precisely the kind of conscientious referencing that is (or should be) second nature to the historian. Code should be elucidated by textual comment in the manner of good history. Object orientation will make this more plausible and more obvious: objects have a history as well as formal parentage.

All this has interesting implications in teaching at undergraduate level. History students are likely to have a better grounding in documenting their work than their more technically gifted peers in the sciences. It is important that this aspect is not neglected in teaching the use of IT, and if the use of computers is taught as an intrinsic part of the historian's method of production, there is no reason why it should be. If IT is taught as a bolt-on unit, with little reference to the subject a student is reading, it may not be recognized that some of the skills the student is taught as a historian remain relevant, even essential, when using computers. It may prove, ironically, that students who produce substantial and well-referenced text for history assessments come to regard the use of computers as so distinct from their normal work, that they, like many software engineers, fail to produce adequate documentation in that part of their studies. However, if the need to produce sound documentation is stressed, then the training that a degree in history gives students should potentially make them promising software engineers.

Consider, for example, a database project. Databases are one of the most exciting and useful IT applications for the historian, a mainstay of undergraduate history and computing courses. The nature of historical sources makes the creation of historical databases a challenging exercise for historian and programmer alike. All the standard historical problems arise and the standard historical criteria apply: the material included in a database must be subject to the same care and attention as any primary source material. An obvious example is that the historian must ensure that any statistical evidence he or she is working with is reliable, comparable, representative and fruitfully categorized. The programmer/historian must address the motives of those who compiled the figures and constructed the categories and must be aware that the definitions of categorizations and classifications are themselves matters of history. Furthermore, the programmer/historian must think carefully about the design of the database. Historical databases, resources for analysis rather than functionally analytical, may seem both objective and empirical, yet the data stored in them is selective. The design of the database, the names and data definition of fields, and the structure of tables in a relational database, are essentially historical choices, judgements open to the criticism of other historians. Attempts were made to quantify the morale of the British Army in 1917 by counting the occurrence of words indicative of good or bad morale in their letters home during censoring (Cab. 24/26 G.T.2052 and Cab. 24/36 G.T.3044). Were the views expressed in letters that men knew would be censored, to relatives they did not want to distress, really likely to be representative of their thoughts? To follow this judgement in database fields of 'good' and 'bad' would be to fail in historical analysis, just as would be the suppression of the use of this contrast.

The programmer/historian must justify a design (argue a case), select and interpret data (evaluate evidence), and present the data clearly in a format in which others may use it (communicate original ideas). In practice, this means that the construction of a working database imposes further problems on the trainee historian, providing at least as great an intellectual challenge as a more conventional piece of research. All this must be done within a rigorous framework including annotated code, referenced sources and textual and technical explanations of design strategy. This is an exercise which addresses fundamental methodological questions concerning the academic discipline of history. The application of computers to the study of history that has accompanied the shift in emphasis from the Great Men of the past to the masses, is well illustrated in a project based around census returns. However, further issues are raised, particularly by the use of statistics. Some American historians, the 'cliometricians' (most notably Robert W. Fogel and Stanley L. Engerman), have argued that there are areas of human behaviour best understood as a system in which both variables and their relationships can be quantified (Fogel and Engerman, 1974; Tosh, 1984, pp. 198–202). What could be a better basis for a discussion of their ideas than students using computers to formulate their own historical models and theories? And in the realms of more conventional history, a database project gives awareness of methodological debates. The selection and interpretation of historical data raises questions about metahistory itself; if the building up of a source of empirical evidence is demonstrably subjective, what of the wider implications for the concept of historical objectivity and the existence of the historical fact? The value of this kind of exercise in history is readily apparent. The added boon is that such a project equips the students with transferable skills of a valuable nature, both in programming and in writing documentation. For such purposes, the documentation produced should be of comparable length to a standard dissertation.

As an example of what I have in mind, the following notes are a brief suggested outline for a third-year dissertation project. Databases are probably the most suitable exercise, but there are alternatives. More experienced programmers might like to attempt simulation programs to test some of the great What-ifs? of history, or interactive educational programs that quiz users (perhaps secondary-school pupils) on their historical knowledge and mark them appropriately. The possibilities are very wide.

An example outline for a third-year software project based around the design and implementation of a historical database

Programming

Implemented Paradox 4 database with data normalized to third Normal Form.

Documentation

See Sommerville (1989).

Requirements specifications

Functional system requirements

These are the system services which are expected by the user of the system. Since the user (in this case a historian) is uninterested in the technical detail, technical implementation concepts should be avoided in this document. In short, this document should be a straight, plain-English statement of what the system is required to do.

Non-functional requirements

These set out the constraints under which the system must operate, for example all input being expressible using the ASCII character set. These documents should be complete and consistent: that is, they should contain *all* the requirements, and no contradictions.

Formal specification

This is a technical analysis of requirements, expressing which Paradox functions will be utilized to meet the above specifications. This document may be finalized after a process of prototyping and requirements validation (see below).

System design

This is a detailed explanation of the design of the database, the tables implemented, relationships and key fields. The design strategy adopted for a database may be prototyping, where the designer experiments with different implementations to find the most suitable technical approach, and the approach which best fulfils the specified requirements. Put simply, this allows the programmer to implement a database, discuss any shortcomings with potential users, then adapt or re-design the database until it is satisfactory. This is, perhaps, the most appropriate methodology for fledgeling programmers. After all, historical databases are not safety-critical. If at first students do not succeed, they can try again, with no harm done. So long as the process is adequately documented, improvisations, experiments, and thinking on the feet is to be encouraged.

Testing, validation and verification

This is a full account of the strategies adopted in the testing of the finished database and the results of that testing. Particular attention should be paid to verification (does this system meet the requirements of the user?) and validation (is the technical implementation of the system correct?).

Operation and maintenance manual

This documentation should instruct the user in how to operate the system and, if necessary, make alterations and updates. Again, since the system is to be used by historians rather than computer scientists, it should be non-technical, as far as possible.

Report

The final document should be a full evaluative report on the implementation of the database. The student might like to take the following areas into consideration:

- How successful was the implementation?
- Are there alternative design/implementation strategies which might produce a better system for the historian?
- How useful are such databases to the historian?
- What are the methodological problems associated with implementing databases of historical data?

References

Fogel R.W. and Engerman S.L. (1974), *Time On The Cross: The Economics of Negro Slavery*, London, Wildwood House.

Cab. 24/26 G.T.2052, 'Notes on the morale of British Troops in France as disclosed by the censorship, 13th September, 1917'.

Cab. 24/36 G.T.3044, 'The British armies in France as gathered from censorship, 12th December, 1917'.

Phillips, G. (1992), *A Quantitative Analysis Package for Historians*, M.Sc. Thesis, University College of Wales, Aberystwyth.

Sommerville, I. (1989), *Software Engineering*, London, Addison-Wesley.

Tosh, J. (1984), *The Pursuit of History*, London, Longman.