27

The development of an integrated archaeological software system

J. Huggett*

27.1 Introduction

The term 'integration' is increasingly used when referring to computer software, but the concept of integration in a package has been interpreted in a variety of ways. It not only describes the performance of the software but also has implications for the way in which the software is used, and the methods by which data are stored and processed within the system. This paper is concerned with the implications for archaeologists of integrated software within a micro-computer environment. An underlying theme is the application of commercially available software in order to reduce development time and costs, and to ensure a reliable upgrade path.

27.2 Integrated software

'Integration' is often applied to software which has a particularly comprehensive and user-friendly front-end—the new 'Control Center' provided with dBase IV is one example of this. Here, integration is used to describe the attempt to create a seamless interface between the different elements of the same package: data definition, data entry, report generation, creation of applications, and so on. The old dBase III 'Assistant' which it replaces was in effect a menu-driven command generator, limited in its facilities, and restrictive for experienced users. In contrast, the dBase IV 'Control Center' is no longer command-based, but, through a series of menus and different screens, enables the user to concentrate on the tasks to be done—creating files, painting screens and forms, establishing relationships between data files, and setting up queries. All that is necessary is a basic knowledge of database design. Thus, the process from conception to completion is simplified without requiring any awareness of the mechanics by which it is achieved.

'Integration' can also be used to describe the extent to which different packages conform to a particular standard, whether file formats or common interfaces. Standardised interfaces are particularly common within packages produced by the same manufacturer. For instance, Lotus software has a particular style of interface which is shared by all the software in the Lotus range, making learning to use the different Lotus packages and its clones an easier process. Similarly, there is the GEM family of software which imitates the Macintosh style of mouse-driven user interface. Most software is able to handle files generated by other packages, but the ability to convert a file into some internal format is quite different to the ability to use that file without modification—any compromise in the conversion process unavoidably means loss of information. Compatibility between software packages at the level of file interchange has increasingly become a marketable asset, but the large number of 'standard' file formats for drawings, spreadsheets, text files and databases mean that inevitably not all will be supported.

* Deansway Archaeology Project
Archaeology Section
Hereford and Worcester County Council
Worcester

'Integrated' software is also a class of software in its own right. The increasing popularity of packages such as Framework, Ability and Microsoft Works illustrates a demand for a single package which will do the work of several. Word processor, spreadsheet, database, graphics and communications are all included within a single package, sharing a common interface and common file structures which enable the interchange of information between the different sections. Different software manufacturers interpret the idea in different ways, but the underlying principle is the creation of a straightforward link between tasks and the ability to communicate information between the separate elements within the single program.

The fully integrated package has a common user interface which offers functionality without extensive training and can transfer files internally without difficulty. It is also generally cheaper to buy the integrated package than to purchase the individual parts. On the other hand, some parts of the integrated package may be weaker than others, some may integrate better than others, and arguably none will be as powerful or as flexible as the stand-alone package. An integrated package may be 'content free' in the sense that it is adaptable to different tasks, but expansion may prove problematic and it is often positively advantageous to tailor software for a specific purpose, rather than suffer a generalised and consequently a possibly inefficient system.

There are therefore different levels of integration which may be defined, but in general terms these may be divided into two categories: at a low level, separate stand-alone packages which have the ability to exchange files and perhaps share a similar user interface, and at a higher level, the single package which performs multiple functions normally associated with individual pieces of software. Clearly, there are advantages and disadvantages associated with each, but the principle remains the same—the creation of a less complex, more intuitive machine-based working environment.

27.3 Integrated software in archaeology

The concept of integrated software is not a new one to archaeology. A variety of suites of more or less integrated software have been available for some time, including Wilcock's PLUTARCH system (Wilcock 1974), Cribb's SITEPAK spatial analysis system (Cribb 1986, Cribb 1987), Kintigh's statistical toolkit (Kintigh 1987), the Bonn Seriation and Clustering Package (Herzog & Scollar 1988), the Institute of Archaeology's IASTATS and IAGRAVES cemetery analysis routines (Hodson & Tyers 1988), and a variety of excavation recording systems. These and most other archaeological systems are user-developed systems, few are widely used, and none have achieved any level of general acceptance as a standard by which others may be measured. Why is this?

None of these archaeological systems are fully integrated in the strictest sense. Cribb, for example, sees his SITEPAK mapping and spatial analysis system as being aimed at the middle ground between widely available commercial software and dedicated user-written applications (Cribb 1986, p. 11), and provides a software environment for a particular type of archaeological analysis. In contrast, Kintigh's statistical routines are supplied as a series of individual stand-alone routines (Kintigh 1987). Each system is applied to a particular field and performs a variety of tasks which have been identified within that field, but it is not capable of expansion by the user or of operating outside that specific application area. None have a built-in, fully-featured word processor, for instance! All the software is purpose-built, using one or other of the many programming languages, and consequently development costs are high and, arguably, there is considerable duplication of effort, at least at the nuts-and-bolts level.

None have reached the level of integration which might be expected in a total 'excavation analysis' package. Instead, the division between excavation and post-excavation is maintained by the software employed at each stage. Indeed, there are no specifications laid down for such a system and it could be argued that it is unlikely that everyone could agree on what should be included except at a very general level. In addition, it is questionable whether such a uniform, standardised system would be of any great benefit to the practice of field archaeology (Huggett 1987).

Clearly therefore, just as there are different levels of integration in commercially available software, so also are there various levels of potential integration in archaeological software, only some of which may be both achievable and desirable. The most suitable level of integration is likely to be at a relatively low level—one that retains the benefits of an integrated system whilst maintaining adaptability and flexibility. One way of achieving this is through a careful blend of commercially available, off-the-shelf software, in which the links between elements in the system are at file level, with some form of overall administrative software controlling movement between the packages.

27.4 Constructing an integrated system from commercial software

Using existing software packages to create a system has a number of advantages. The developer can largely ignore the low-level aspects of the software (the mechanics of drawing lines, or searching records, for instance) and concentrate on the way in which the software is to be used. Provision for the transfer and conversion of files is also made within the packages so that the developer can focus on what is actually put in the files rather than how to move the files from one package to another. Consequently development time is reduced and the resulting system is potentially more robust and capable of expansion.

There are two main features which are necessary in a software package considered for inclusion within such a system: compatibility and adaptability. Of these, adaptability is perhaps the most important, since this feature can be used to overcome any problems of compatibility.

Software compatibility makes it possible to transfer information between the different packages and between different users—there is little point in buying desk-top publishing software if it cannot import onto a page a drawing from the CAD package also being used. Compatibility may also mean that additional software such as expensive fonts can be used by several packages within the system. Software compatibility is not the same as having compatible recording systems, however, but standardising on file types rather than the contents of files is perhaps a more realistic and preferable route.

The ability to modify a package may range from simply setting up different style sheets within a desk-top publishing package in order to simplify the production of reports, to a facility which allows the complete redefinition of the appearance, commands and operation of the package. In this way, software which is 'content free' can be tailored to perform a specific task whilst retaining the flexibility of the original package. Ideally, a package should come with a built-in command language which can be used to create macros or programs designed to perform precise functions. The most powerful packages enable the developer to redesign the generalised user interface in order to make it easier to use and to guide a user more accurately through the necessary stages. The end result may be a package which bears very little resemblance to the original software, but underneath the surface all the built-in, standard functions are used to process the information. In fact, there are relatively few packages in the world of PC software which allow this degree of modification: examples include top of the range database systems such as dBase III/IV or Paradox,

spreadsheets such as Lotus 123 or Quattro, the Sprint word processor, and AutoCAD with the AutoLisp extension. One of the few fully integrated commercial packages which also enables a high degree of user adaptation is Framework with its FRED language.

The high-level user interface can also take a variety of forms, but the basic principle is that it should provide a simple 'point and shoot' facility for users to start up applications and copy and delete files. This type of software acts as an administrative program sitting on top of the other applications within the system and from which all applications are run. MS-DOS, CP/M or one of the PC variants of Unix are not considered to be an option here, although the simplest form of interface could take the form of a batch file menu system. The two most obvious interface packages are Microsoft Windows and GEM Desktop, and these have an assortment of imitators. Neither are particularly configurable, but both provide some form of graphical interface which is intuitive and easy to use. However, the space that such administrative software occupies can cause problems when attempting to run a memory-hungry application although the memory requirements of an increasing number of software packages can be configured to suit.

27.5 An example system—the Deansway Archaeology Project

Rather than continue to refer to integrated software systems in the abstract, an existing system may be introduced as an example. The software used on the Deansway Archaeology Project in Worcester is not put forward as a completely integrated system: it is a semi-integrated system which serves as a focus for discussion of the benefits and problems associated with this approach.

The Deansway Project uses a variety of commercial software running under the icon-based interface provided by GEM Desktop. Packages used consist of AutoCAD Release 9, GEM Graph, GEM Draw, Ventura Desktop Publisher, WordStar and dBase IV (see Huggett 1989). The level of integration between the packages varies and tends to focus on Ventura. For example, images produced using the GEM software can be placed on a Ventura page, as can text generated from WordStar. Software bugs in Ventura 1.1 prevent the use of AutoCAD slides, but plot files can be used, and AutoCAD is capable of producing a variety of standard output files. On the other hand, WordStar is not able to output documents using images generated by other packages, for instance.

Of these packages, only dBase IV and AutoCAD are truly modifiable in that they are equipped with programming languages which can be used to alter the way in which they work. WordStar and Ventura can have document layouts created which simplify the process of creating standardised output, but otherwise the packages used are very much in the category of 'what you see is what you get' in the sense of performance and user interface rather than output. As a result, the primary programming effort has concentrated on dBase and AutoCAD in order to configure them to perform as required. Where needed, DOS batch files or C programs have been written as small stand-alone utilities to create procedures for backing up files or downloading soft fonts for instance.

The programming and modification of the packages had a variety of aims. Whilst both dBase IV and AutoCAD work straight from the box, neither are configured to work efficiently within their application area. Consequently, it proved necessary to tailor them to the application in order to make them simpler and faster to operate, reduce the learning curve, and cut down on the overheads of facilities which were either not required or were only needed in a greatly modified form. In addition, part of the overall system shared a common style of interface—mouse-driven pull-down or pop-up menus. dBase is particularly lacking in this area, so the application was developed

entirely using pop-up menus, although a mouse interface does not currently exist. One aspect of the system which has to be borne in mind is that it has been regularly called upon for demonstration purposes to other specialists, officials, and members of the public. Thus the appearance of the system and the extent to which it was integrated has proved to be almost as important as its content—a program which performs impressively yet has an under-developed user interface will mean nothing to an non-expert onlooker.

A major benefit has been the speed of development of the overall system. This is an important factor since all the development work has been carried out since the Deansway Project started and while data entry was being carried out. The software packages provided the necessary basic facilities straight from the box, although it was apparent that the level of investment in terms of training was very high, particularly with packages such as Ventura and AutoCAD.

For example, the addition of specialised commands to perform specific repetitive tasks has resulted in a dramatic improvement in the amount of time taken to digitise a context plan. Putting a single level point on a plan required the drawing of a level triangle to be imported, scaled and placed in the correct position, and the text for the level reading to be positioned, scaled and entered. This has to be carried out for every single level point on the plan. A simple function automates this procedure, enabling the operator to simply pick the position for the level triangle and enter the level reading without any of the intermediate commands being necessary. Similarly, to draw a grid point would require the accurate drawing of four lines, a central dot, and the positioning, scaling and entering of the text for the grid co-ordinates. Instead, a single command allows a point to be selected and the grid point is automatically scaled and drawn, and the co-ordinates calculated and written beside it. Both these and other commands operate in a continuous loop which the user ends by simply pressing the Return key or a mouse button. This level of automation means that, on average, the digitisation of a single context plan takes around 10 minutes to complete.

The tailoring of the software to the defined tasks reduced the level of training that a person needs in order to operate the system efficiently. The most obvious example of this is the dBase system which is completely self-contained and requires no knowledge of either databases or of dBase itself. The extensive use of status help lines, pop-up bar menus and context-sensitive help at the press of a key make it simple to use and quick to learn. AutoCAD is similarly made much simpler to use by the addition of the specialised commands and modification of the pull-down menu system. For instance, teaching a novice computer-user to use the standard AutoCAD facilities to digitise a context plan took in excess of one day to complete a single plan. Using the modified system, trainees were digitising plans within one hour.

The reduction in training time is in effect achieved by protecting users from the elements of the package which they do not need to use or to have any knowledge about. This might sound like an argument for the creation of a computing elite anxious to protect their knowledge from the archaeological masses. However, there is a wide gulf between computer literacy and computer expertise (see Richards 1985, for instance) and it is an inefficient use of resources to expect every archaeological computer user to become a competent programmer or, to use the jargon, a 'power user'.

27.6 Conclusions

Integrating the separate packages by a combination of easing the transfer of data between them and creating a series of standardised user interfaces reduces the learning curve which is necessary. Transferring files between machines, let alone

packages on the same machine, often causes considerable problems for users, and this is an area where considerable improvements can be made. Similarly, using a uniform user interface over the top of the various packages and attempting to customise the software to conform to a common style of interface (mouse-driven pull-down menus, for example) all help to assist the user to become familiar and comfortable with the system.

The level of integration discussed here is relatively low, yet the benefits are clearly apparent. Greater integration is possible and indeed desirable, but the reasons are those of flexibility, not necessarily those of compatibility. There is a danger that integration of software could be interpreted as standardising not only on the user interface and the interchange of data between systems, but also as the definition of record structures themselves and the way that information is recorded and stored. Integrated systems per se offer considerable benefits and advantages—it is their application which is likely to give rise to the greatest problems.

Proper integration of software makes the resulting system easier to use and far more flexible than the individual stand-alone components. The ability to transfer information smoothly between the different elements can only improve the functionality of the end product for the user.

27.7 The Future

Integrated software packages which are capable of being modified and customised according to requirements are already with us, with Ashton Tates's Framework III perhaps the most obvious example. It operates on the principle of single data objects, frames, which can be manipulated within the various different environments. The whole package is bound together by a common interface and has a built-in programming language which can be used for the development of applications which can then become part of the package using the applications menu.

Creating a flexible integrated package from scratch is also becoming a more realistic alternative. The development of Computer Aided Software Engineering tools means that systems can be created using standardised routines for user interfaces and file handling, and the system developer can concentrate on the design and functionality of the final system. PC-based CASE software is not yet so advanced that it can be easily used by a non-programmer, however, despite the advertising claims of manufacturers.

To some extent, the future is not in archaeological hands—the new Systems Application Architecture from IBM offers the prospect of applications programming using a common core of routines: the Common User Interface, the Common Programming Interface, Common Communications Support, and Common Applications. Such a level of standardisation would completely change the way in which applications are developed (for example, Langa 1989). With each package using the resources available under the Systems Application Architecture there will be no need for software to have built-in screen handling, user interfaces and so on. The duplication of effort involved in the creation of these for every single application will be removed, allowing developers to concentrate on the functionality of the software. The main barrier to this at present is the cost of the hardware needed to operate such systems.

However, whether the IBM standard is adopted or not, integration of powerful software at this level will happen in the near future. Doubters need only consider the current state of computer applications and recall with nostalgia the CP/M Z80 standards of 1980 to appreciate that the rapid improvements in hardware have nearly been matched by developments in software, and this is a trend which seems likely to continue. The development of such highly integrated software provides great

opportunities for archaeological users—it is the way that archaeologists choose to use it that will determine its eventual utility.

Bibliography

CRIBB, R. 1986. "A Dedicated Spatial Analysis Package for Archaeology", *Archaeological Computing Newsletter*, 9: 5–12.

CRIBB, R. 1987. "Some Spatial Analysis Applications Programs", *Archaeological Computing Newsletter*, 13: 6–14.

HERZOG, I. & I SCOLLAR 1988. "A Mathematical Basis for the Simulation of Seriatable Data". In Rahtz 1988, pp. 53–62.

HODSON, F. R. & P. A. TYERS 1988. "Data Analysis for Archaeologists: The Institute of Archaeology Packages". In Rahtz 1988, pp. 31–41.

HUGGETT, J. 1987. "Computer Usage in British Archaeology—a Review", *Archaeological Computing Newsletter*, 10: 12–17.

HUGGETT, J. 1989. "Computing and the Deansway Archaeology Project", *Archaeological Computing Newsletter*, 18: 1–7.

KINTIGH, K. W. 1987. "The Archaeologist's Analytical Toolkit", *Archaeological Computing Newsletter*, 10: 4–6.

LANGA, F. 1989. "The End of Application Software?", *Byte*, 14 (2): 6.

RAHTZ, S. P. Q., (ed.) 1988. *Computer and Quantitative Methods in Archaeology 1988*, International Series 446, Oxford. British Archaeological Reports.

RICHARDS, J. 1985. "Training Archaeologists to Use Computers", *Archaeological Computing Newsletter*, 2: 2–5.

WILCOCK, J. D. 1974. "The facilities of the PLUTARCH System", *Science and Archaeology*, 11: 16–24.