Standard software versus high-level languages

R.C. HUTTY
GEC-Elliott, Borehamwood, England

(Presented by P. Heywood)

1. Introduction

The purpose of this paper is to assess the merits of using Standard Software as against a High-Level Language for application programming. Although some of the points made in this paper are applicable generally to all fields of computing, they are intended to refer mainly to the computing field of process and industrial control.

2. Definitions

As the terms 'standard software' and 'high-level language' are often misused, their meanings, as used in this paper, will be outlined.

Standard software

Standard software is that software which is designed, coded and tested once, but used, with little or no alteration, in more than one computer system. Standard software for a particular computer is normally developed by the computer manufacturer. Standard software falls into three categories:

1. Basic standard software.
2. Application standard software.
3. Problem-orientated languages.

1. Basic standard software refers to standard software which is designed for a particular computer range and is general-purpose enough to be used in any computer system which utilises one of the computers in the range. Basic standard software, e.g. executives and operating systems, is used in almost every computer system. Other basic standard software, e.g. mathematical routines, although general-purpose, is not necessarily used in all systems.

2. Application standard software, normally used with the support of basic standard software, is designed to satisfy the software requirements for a particular application area. Application standard software is developed only once but can be used in any computer system required to perform the same kind of function, e.g. a DDC system. Normally the only application programming work required is the initialisation of parameter values relevant to a particular system and a requirement specification of the parts (usually optional) of the software required for the system.

3. Problem-orientated languages are similar to high-level languages except that they are defined for a specific field of application, e.g. sequence control, and are machine-dependent. The software for a system is written using the statements of the language. However, the set of statements available is restricted and designed only for the narrow field of application. It is therefore not likely to be useful for any other field of application.

High level languages

High-level languages are used for one-off application programming. High-level languages are now replacing assembly languages for one-off application programming. The advantages and disadvantages of using high-level languages instead of assembly languages are not discussed here.

High-level languages, e.g. FORTRAN and ALGOL, and their statements are designed for general use. Real-time languages such as RTL are included in this category.

High-level languages are not wholly general but are designed for very much wider fields of application than the narrow field for which problem-orientated languages are suitable. For example, real-time languages are designed for all industrial control and real-time applications, whereas a sequence control language would only be suitable for relevant industrial sequence control applications.

3. Specific comparisons

For a comparison between the use of standard software and a high-level language, every aspect of a computer manufacturer's business, from selling to commissioning, must be considered.

Marketing

The marketing of systems is enhanced by the availability of extensive standard software, especially application standard software and problem-orientated languages. It is possible to provide meaningful publicity describing the particular item of standard software and its area of application. Potential customers are able to identify their company's requirements with software designed to solve their own particular problem. Standard software packages lend themselves to more illuminating advertising languages. High-level languages can be advertised, but their impact on potential customers is not as great.

Selling

Standard software makes the task of selling systems easier. Sales teams can aim at definite market areas rather than a general area. Salesmen are able to sell fixed price packages for some types of application standard software and are able to communicate with the potential customers via the standard software because it is designed for the customer's application and is readily familiar to and understood by the customer.

Tendering

Tendering of software systems currently involves a higher degree of risk than is normal in an engineering environment and therefore anything which helps to reduce this risk is obviously advantageous. Estimating the software content of a contract is more accurate when standard software is available than with high-level languages. The problem of estimating storage for a system is reduced if standard software is used because a larger amount of store is pre-defined by standard software than languages.

Not only storage estimates, but also the manpower required (which is often proportional to the storage requirements), are easier to assess because a standard software package inherently defines what is required to be done to make it suitable for a particular system. Even if it is not possible for the whole of the software system to be standard, at least that part which is standard is likely to be estimated accurately. Standard software enables the function of a software system to be specified precisely. The customer therefore knows exactly what he will receive and the programmer who will implement the system will know what to provide.

Programming

Standard software reduces the programming skill required. This makes the software less expensive, more controllable and reduces the risk. Standard software should provide an over-all software cost saving over the use of high-level languages. A programmer's job is reduced when using standard software because it has been predefined and pre-structured. Programming may be just a matter of supplying data to application standard software.

It is likely that non-computer people will be able to 'program' a system using application standard software or problem-orientated languages. Thus, engineers who specialise in the application field for which the standard software is designed can 'program' the software for his own application, thereby providing a system which does exactly what he wants it to do. Therefore, the problem created by the difference between what an engineer actually wants and what a programmer actually provides is removed.

Commissioning

Commissioning problems are reduced when standard software is used because there are fewer bugs in well tried and proven standard software than a first-time one-off program.

Documentation

Standard software reduces the amount of documentation to be produced for each contract. Standard documentation is a natural by-product of standard software. However, high-level languages provide a good degree of self-documentation.

4. General comparisons

There are many points other than those associated with each phase of a contract, which are general and worth noting.

Staff changes

Staff changes during a contract pose fewer problems when standard software is used, because the software system and its requirements are capable of being well defined initially and do not

usually change.

Machine dependence

Standard software is usually machine-dependent and, as such, is designed to be as efficient as possible, with respect to both storage and execution time. However, high level languages are usually machine-independent, so when one of these languages is required to be used on a particular computer the object code produced by the compiler may be unsuitably inefficient – this depends a great deal on the particular language-computer combination.

Suitability

It is very difficult to define a high-level language which is suitable for all applications within the extensive areas of application for which they are designed. For example, the inefficiencies produced by high-level languages are likely to be too large to be used as an alternative to basic standard software. However, efficiency is not always of prime importance.

Unfortunately, it is usually the case that standard software, especially application standard software, does not provide exactly what is required for a particular application, even though the application is considered to be in the area intended to be covered by the software. This usually happens when a customer's requirements extend outside the norm for which the software has been designed. Where the requirements are not essential then it must be pointed out to the customer the extra costs that are incurred by either 'bending' the standard software or programming the system in a high-level language, as a one-off system, as against using the standard software intact. Where the requirements are essential then there is no option but to 'bend' the standard software or to use a high-level language.

Quality

The suitability of standard software depends upon the quality of its design. A knowledge of the functional requirements of standard software is an important necessity before its design can begin; this will ensure that the software is likely to satisfy more applications than it would otherwise. The design of standard software should ensure that it can be easily made to fulfil the requirements of a particular application without redundancy. This is often currently achieved by using a modular approach in the design of the software.

Tools for a good system construction facility are necessary for modularly constructed software. It is better to 'bend' standard software than to use only a high-level language since at least some of the software will remain intact and as such be well proven. The bending should be performed, if possible, by the team who implemented the software, in order to reduce the risk of errors.

Development cost

Standard software development costs must be compared with the cost of producing a compiler for a high-level language. If any appreciable amount of standard software is developed then its cost will be greater than the cost of developing a compiler. However, it is usual to have available both standard software and a high-level language in order to completely cover all possible applications, and therefore development costs are required for both.

5. Conclusion

It would seem that there will always be a requirement for high-level languages and standard software because of the shortcomings of standard software. Although standard software, where it can be used, will usually be less expensive and involve less risk than a high-level language, it is only in simple applications that a system could be made up completely of standard software. Hence, in most systems, standard software requires to be either supplemented, or replaced, by a high-level language. It is important that a high-level language be compatible with the standard software.

As a general rule, wherever possible, standard software should be used in place of high-level languages.

Discussion

Q. In what level of programming language is your standard software written and specified?

A. It is written in assembly language, and specified in terms of the facilities it provides (and the implications thereof) in assembly language terms.

Q. How stable do you find your standards?

A. Packages are always modified, but only slightly.

Q. Are not modifications to standard software difficult to test?

A. Possibly so for an individual system, but not essentially worse than testing the original package.

Q. Suppose a piece of standard software (let us call it S) is slightly bent to form something we shall call S'. Then henceforth, is S or S' the new standard software?

A. The software S' may be eligible for standardisation, but is certainly not new standard software automatically.