Code Management Systems

V.Innocente*, C.Onions
CERN, Geneva, Switzerland
and
T.P.Shah
Rutherford Appleton Laboratory, Chilton Didcot, U.K.

Abstract

We discuss various aspects of Code Management Systems: their necessity and the attributes they should possess. The existing systems in use are briefly reviewed and compared.

1 Introduction

High Energy Physics experiments involve complex detector systems and generate a large amount of data which have to be usually processed in a large variety of machines in different countries. The software to handle this processing is developed at various sites and then merged together at a central distribution centre. The life of this software is of the order of ten years and during this time it would normally evolve to take into account new algorithms and upgraded detectors etc. A computer based system which facilitates the task of code development, maintenance and distribution is of vital importance in such an environment. A code management system is needed to meet some of the following specific requirements:

- For inserting common sequences shared by several routines in the appropriate places.
- For organising the source code into decks or modules and defining a set of modules which would make up a production program.
- For code distribution to collaborating institutions and maintaining parallel versions at different sites.
- For generating different program versions (e.g. to handle the inevitable machine dependent code).
- For maintaining a history of code changes and for providing the means of recovering an earlier version.

*permanent address: INFN, Sezione di Napoli, Napoli, Italy

33
In the following sections we list the desirable attributes of a modern code management system and then review some of the existing systems in use in the HEP community. Finally, in the concluding section we give a summary table of what each system provides and we speculate on the factors that would ultimately govern one's choice of a code management system in the LHC era.

2 Desirable Features

We list below the requirements a modern code management system should satisfy in order to provide the functionality needed in the context of a modern large HEP experiment where the software development and processing is distributed over many institutions and computing systems.

1. Because modern HEP experiments tend to be large collaborations, and the software is developed at different sites, the code manager has to be able to maintain parallel versions on different computer systems. Hence the code management system has to be portable.

2. It should be easy to use with a user-friendly interface. The code to be modified should be quickly accessible and any editing should be possible using the local editor without leaving the environment.

3. The code manager should provide the same interface to the user on different platforms (i.e. he/she should be able to use the same commands).

4. The code manager must support the inclusion of macros such that common code can be expanded when required (e.g. to produce object libraries).

5. The code manager must be able to handle conditional code in a flexible way (e.g. machine dependent code, generating different program versions).

6. It should provide a history of the code development with a facility for recreating an older version if needed.

7. It should be possible to obtain the differences between two versions and, optionally, it should be possible to generate a correction deck (a delta file) to update an earlier version.

8. Comments on why a new version was produced should be required (e.g. who made the change and why).

9. The organization of the source code or macros should be in decks or modules and it should be possible to define a group of modules which logically belong together.

10. The code manager should provide some control over simultaneous changes to a module by means of a warning or a lock.

11. It should be possible to merge variant versions of a module (although this cannot be done in a fully automatic way and care would always need to be exercised in any merging operation).
12. The code management system should have facilities for the distribution of code to outside institutions, i.e. a transportable format should be provided. However, in order to make the transfer quick over wide area networks, updating with delta files should be possible. The delta file should only be able to modify the unique file to which it applies.

13. Finally, the code manager should allow multiple source code libraries with no arbitrary limit on the number of lines of code.

3 Review of some existing Code Managers

In this section we review some existing code managers, particularly those which have been adopted by sections of the HEP community. We start with PATCHY which is the oldest and has probably been used by more HEP people than any other code manager. Then we look briefly at other products (mostly commercial): SCCS, VAX CMS, HISTORIAN, CMZ and Codebase.

3.1 PATCHY

PATCHY [1] was developed at CERN around 1965 at a time when programs were fed to the computer via a card reader. By maintaining the source code in a PATCHY Master file (PAM) on tape and using a small cradle of update cards the user was freed from the burden of reading in a trayful of cards! It reads in and acts upon several sequential files by means of cards. PATCHY, because of its origins, is not an interactive program at all. Because it accesses the data sequentially PATCHY is necessarily slow or cumbersome in those facilities best provided by random access.

PATCHY handles common sequences well and conditional code can also be managed but in a slightly more complicated manner. History of code development can only be maintained if all changes are made via correction decks at the start of the PAM file. In particular, if source code is modified directly using the local editor then no history is available (unless done manually by way of a comments deck at the start); there is no automatic way of going back to an earlier version in this case. The updating process cannot be controlled or monitored in any way. PATCHY is very good for distributing code to other sites on tape and correction decks can be circulated for the PAM files in other centres but no guarantee that they would apply to the correct version(s) of the PAM file. PATCHY, as already mentioned, has been the most widely used code manager in the HEP community.

3.2 Unix SCCS

This system is not used in HEP but we mention it as a reminder that the problems and solutions relating to code management are not unique to our community. The Source Code Control System, SCCS [2] maintains a record of versions of a file containing software, a program or documentation. It is supposed to keep track of each set of changes, who made them, why and when. Changes can be merged and control is exercised to ensure that two people do
not edit the same file at the same time. It also allows old versions to be recovered. There does not, however, seem to be provision for the transfer of files to other sites. Also the notation is a little too succint!

3.3 VAX CMS

This is a commercial product which is derived from or inspired by Unix SCCS. VAX CMS [3] provides sophisticated facilities for maintaining a history of code development. Code can be modified using the local full-screen editor. The system controls simultaneous updating of code and provides a way of merging variant versions arising from a common ancestor. However there are no facilities for handling conditional code and common sequences can only be handled using the Fortran INCLUDE. Code distribution is also not foreseen. These limitations arise because this system is for VAX use only. It has been used by the CDF experiment and they have written a pre-processor to solve the problems of conditional code and common sequences handling. Users of VAX CMS are very pleased with it, particularly its speed and robustness.

3.4 HISTORIAN

This is another commercial product and was originally intended as a replacement for PATCHY at CERN. It is marketed by OPCODE Inc. based in Texas. The HISTORIAN PLUS system [1] consists of a batch sub-system and an interactive sub-system. In the batch mode, updates are applied to code stored in the library and either a new library, the "source" code, "compile" code or a so-called "compatible" library for export can be produced. The interactive mode uses HISTEXTRACT to extract the code to be modified, which can be edited using any local editor and a correction set generated, or the library updated, interactively using HISTGEN. Typically, users generate correction sets which are sent to a library manager for inclusion in a new release. The new library which is produced contains a history of updates and thus permits the recovery of earlier versions. HISTORIAN does not allow variant versions but old versions are recoverable. Note that the version number is a function of the library as a whole and not of the individual decks. Control of updates is minimal. HISTORIAN handles conditional code and common sequences very well, in particular allowing nested conditions and automatic propagation of common deck changes. Code distribution via "compatible library" is possible but usually these libraries tend to be rather large. A useful feature, which is not found in other code managers, is the unique identifier given to each record in the file. If a record is deleted, then attempts to modify the record in a later run are flagged with a warning. In other code managers, a different record would be modified without any warning at all. At the time of writing, CERN has negotiated a new contract with OPCODE such that the company will only maintain the current version; no further development or installation on new machines is foreseen. HISTORIAN has been used by several experiments (e.g. ALEPH,CHARM2,NA31,UA2...) at CERN and generally the users have been quite satisfied with it.
3.5 CMZ

CMZ (Code Management system using Zebra) [5] was developed by CodeMe S.A.R.L. This is also a commercial product which is about to be officially supported by CERN and it runs on any computer on which the CERN library does. CMZ sets out to provide a code manager which is fully PATCHY compatible and provides for interactive code modification and development using the local system editor. From within the CMZ environment it is possible to compile and link Fortran and C programs, import or export C, Fortran, "text" or PAM files and generate correction sets by comparing two PAM cycles. It also provides several tools to check and tidy Fortran code, such as statement relabelling and indenting, checking for undefined variables etc. It handles common sequences and conditional code as PATCHY does but allows the inclusion of conditional code also in sequence definitions. In CMZ, when a deck is modified and then saved, a new RZ cycle is created; multiple copies enable history to be maintained. A mechanism to handle versions is provided, but multiple lines of developments are not allowed. CMZ does not provide any enforced control of changes but a history of all changes applied to the deck is kept in comment lines at the top of the deck itself. The organization of the source is such that PATCHY PATCH files become RZ directories and the decks then are within this directory. The code distribution is done using CARD images of the CMZ files (which are PATCHY compatible) or correction decks.

The appeal of CMZ is clearly for those who wish to make a smooth transition from PATCHY to a more modern product. Its functionality is however somehow constrained by its design aim of being completely compatible with PATCHY. It has been used at CERN by the L3 collaboration who use it in the code development phase (using Apollos) but code stability for production is maintained by PATCHY mainly for historical reasons. Code distribution to outside laboratories is also done using PATCHY; correction sets are distributed regularly to keep production libraries in collaborating institutions up to date. The L3 collaboration has experienced some problems with the integrity of the CMZ library files (they get corrupted from time to time). These problems are supposed to be solved by a new version of the Zebra RZ package. CMZ has also been chosen by the H1 HERA collaboration at DESY.

3.6 Codebase

This is a commercially available code manager developed originally to the specifications drawn up by the Zeus experiment at HERA [6,7]. It is marketed by BASE GmbH. Its design aim was to provide a portable code manager which would have the functionality of VAX CMS but cater also for common sequences and conditional code handling. Written entirely in C it is supposed to provide a common interface on various platforms (VAX, IBM/VM, IBM/MVS, GOULD etc.). It stores the decks as modules with a prefix (equivalent to a directory). These modules can be extracted for modification, edited using the local editor, expanded for compilation and testing and saved as new versions in the Codebase library. This new version, or any earlier version of
this module (note that version number applies to the module), can be extracted from the library. The Codebase system allows common code handling and powerful conditional code handling using Boolean expressions allowing logical operators AND, OR, and NOT. Every time a module is saved in the library a comparison is made between the original and edited versions and corrections are appropriately inserted. Comments on changes are also recorded and a full history of all changes to a module can be obtained by requesting a special annotated listing. Codebase also controls the updates that can be performed on a module simultaneously by means of "locks".

Codebase is a very new system which is now beginning to be used seriously by the Zeus experiment on the HERA machine at DESY. Potentially it is rather powerful but, at the time of writing, not all the advertised features work on all platforms. It has so far been most extensively tested on the VAX and IBM/MVS systems. Its response time for some commands is still considered rather slow and it is not yet felt to be robust enough.

4 Summary and Conclusions

In the table below we give a list of desirable attributes (Features) and what the different systems have to offer (the more stars the better it is). Common Sequences are supported by all code managers except VAX CMS; only Codebase and CMZ allow these sequences to reside in any library (note that PATCHLY can also emulate this but rather painfully !). Similarly Conditional Code is handled by all except VAX CMS and SCCS for reasons discussed above. The Multiple Versions feature is either through a single line of development or via variant versions. The latter are supported only by VAX CMS and Codebase. Control of Changes refers to the possibility of several people modifying the same module simultaneously; again VAX CMS, Codebase and SCCS issue some warnings and have "locking out" features. Documentation of changes means that at least it provides a way to document who modified the code. Organization refers to how the code is structured. Distribution refers to the availability of a transport format for distributing code and/or update decks. Multiple language asks what sort of languages are supported by the code manager. Interactivity refers to the power of the interactive version and to its speed. Utility stands for tools other than the one used for the code management itself, like code checking and tidying tools. Size refers to the size of the library for the same code. Portability refers to the present availability of the code managing program itself on various computers. We conclude by giving a rough guide of relative cost to the user: CERN products tend to be "free" (ignoring manpower costs) whereas commercial products not distributed through some general licensing agreement through CERN would cost "real" money.

The final choice of system depends on the weight that one gives to the different desirable features. Of course the best code management system is one which costs nothing and offers all the features we want and works so well that its existence is hardly noticed ! In practice this utopian situation cannot be attained and the choice must depend on what one can afford and what
is likely to be most acceptable to the user community in the experiment. If
PATCHY compatibility is a must then CMZ would seem to be a good choice;
if, on the other hand, you want a powerful system which can handle merging
of variant versions then a Codebase-like system could be the answer; VAX
CMS would be a good solution if the code was likely to be developed and
maintained on the VAX machines exclusively. It could also be the case that
in the future, with Unix systems being the norm everywhere, there will be a
sufficiently portable and powerful but user-friendly Unix based product!

A good code manager should facilitate both the development and main-
tenance of code. Portability, robustness and speed are very important features.
Code managers with these qualities, which also address the particular needs
of distributed computing, will clearly be needed in the LHC era.

5 Acknowledgements

We would like to thank the following people for providing us with useful input
for this paper: J.C. Hart, H. Kowalski, P. Palazzi, S. Fisher.

References

Equipment Corporation, Maynard, Massachusetts.
1990.
[7] Requirements for a Code Management and Distribution Facility, ZEUS-
Table 1: Features of various Code Management Systems

<table>
<thead>
<tr>
<th>Feature</th>
<th>PATCHY</th>
<th>SCCS</th>
<th>VAX CMS</th>
<th>HISTORIAN</th>
<th>CMZ</th>
<th>Codebase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common Seq</td>
<td>***</td>
<td>?</td>
<td>?</td>
<td>***</td>
<td>***</td>
<td>***</td>
</tr>
<tr>
<td>One Lib.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt; 1 Lib.</td>
<td>*</td>
<td>?</td>
<td>?</td>
<td>***</td>
<td>***</td>
<td>***</td>
</tr>
<tr>
<td>Cond. code</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td>***</td>
<td></td>
</tr>
<tr>
<td>Multi Vers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unique line of dev.</td>
<td>?</td>
<td>?</td>
<td>***</td>
<td>***</td>
<td>***</td>
<td>***</td>
</tr>
<tr>
<td>Variant versions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control of changes</td>
<td>***</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>***</td>
</tr>
<tr>
<td>Docum. of changes</td>
<td>***</td>
<td></td>
<td>***</td>
<td></td>
<td></td>
<td>***</td>
</tr>
<tr>
<td>Prop. of changes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>***</td>
</tr>
<tr>
<td>Organization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>***</td>
</tr>
<tr>
<td>Deck/Module</td>
<td>***</td>
<td>?</td>
<td>***</td>
<td>***</td>
<td>***</td>
<td>***</td>
</tr>
<tr>
<td>Patch/Group</td>
<td>***</td>
<td>?</td>
<td>***</td>
<td>***</td>
<td>***</td>
<td>***</td>
</tr>
<tr>
<td>Set/“Class”</td>
<td>?</td>
<td>***</td>
<td></td>
<td>***</td>
<td>***</td>
<td></td>
</tr>
<tr>
<td>Distribution</td>
<td>***</td>
<td>?</td>
<td>?</td>
<td>***</td>
<td>***</td>
<td>***</td>
</tr>
<tr>
<td>Transport Format</td>
<td>***</td>
<td>?</td>
<td>?</td>
<td>***</td>
<td>***</td>
<td>***</td>
</tr>
<tr>
<td>Multiple Languages</td>
<td>*</td>
<td>?</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interactivity</td>
<td>**</td>
<td></td>
<td>***</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Utilities</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>***</td>
</tr>
<tr>
<td>Speed</td>
<td>+</td>
<td>?</td>
<td>**</td>
<td></td>
<td></td>
<td>***</td>
</tr>
<tr>
<td>Size</td>
<td>***</td>
<td>?</td>
<td>**</td>
<td></td>
<td></td>
<td>***</td>
</tr>
<tr>
<td>Portability</td>
<td>***</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cost</td>
<td>***</td>
<td>**</td>
<td>**</td>
<td></td>
<td></td>
<td>***</td>
</tr>
</tbody>
</table>

The star rating is our assessment of how well a feature is catered for: * means feature is available, *** means it is well implemented or is rather complete, a blank means it is not supported and a '?' means that we are unsure of this feature (or rating) for this product.