Problems with COTS Software: A Case Study

Jamshid Beheshti and John Dupuis

McGill University

Abstract

This article reports on the use of Commercial-Off-The-Shelf (COTS) software for developing a dynamic environment for an online public access catalogue (OPAC). COTS products are widely used throughout the industry. While there are many potential benefits, use of COTS components is also fraught with pitfalls. The research on creating a dynamic environment for OPACs is based on the previous work in this area, Public Access Catalogue Extension (PACE), which was developed with custom-based software programs. Although in the previous research project all the programs were successfully developed in C and C++, the present project relied very little on original and custom programming. Instead, a number of COTS products were used to construct the dynamic environment: *Macromedia Director*, *3D Dreams*, *Extreme 3D*, *Crossroads*, and *Easybase*. These COTS products were chosen for their ability to produce the desired results, their availability at reasonable costs, and their capability to integrate with one another. A small experimental database with one hundred MARC records was constructed in Easybase. Models were built in *Extreme 3D* and converted to *3D Studio* using *Crossroads*. These models were used in *3D Dreams* to create three-dimensional environments for use in *Macromedia Director*.

1. Introduction

Commercial off-the-shelf (COTS) software has been gaining popularity in the last few years. It is considered an economically viable method of integrating various software components to produce a new product. Many commercial and governmental organizations are now relying on COTS software rather than developing and maintaining their own programs. While in 1997, about one quarter of the software used in a corporation was COTS; it is estimated that by

2004 this number will increase to more than 40%.[1] COTS products seem to be less expensive than proprietary software to integrate into the infrastructure of an organization, whilst decreasing the development time for new products. Considering that large and complex systems require a massive amount of coding (for example about 300,000 lines of codes for cellular telephones) and that many programmers can only compose about ten lines of code per day,[2] COTS products provide feasible alternatives to custom development efforts.

COTS software products have been defined as " commercial items that have been sold, leased, or licensed in a quantity of at least 10 copies in the commercial marketplace, at an advertised price. COTS software products include a description or definition of the functions the software performs, documented to good commercial standards, and a definition of the resources needed to run the software." [3] Four issues need to be addressed when considering COTS software: " functionality and performance, interoperability, product evolution, and vendor behavior." [4] Functionality of COTS software determines the outcome of the system, while its performance depends on various factors including vendor's specifications. Many variables affect COTS programs interoperability with each other and with proprietary software, including the source of COTS software, presence or absences of standards, and the extent of use of open architecture. Vendors' commitment to upgrade COTS products and the software evolution cycle is usually independent of users' system requirements. Finally, each vendor's ability, willingness, resources, and attitude determine their behavior and conduct towards their customers and the level of support that they provide.

Developing a new system based on COTS products involves acquiring different components, customizing various software packages to suit local requirements, and gluing the pieces together.[5] The advantages include predictable license costs, shorter development time, broad and immediate availability, comprehensive functionality and potentially frequent upgrades. The reality, however, may be quite different. The disadvantages of using COTS products may consist of initial cost of purchasing, limited functionality, almost complete

dependency on the vendor, difficult integration procedures, potential unreliability, incompatibilities among vendors, and limited license agreements.[6]

This paper outlines two projects to demonstrate qualitatively the advantages and disadvantages of using COTS software in the construction of the systems.[7] All the programming for the first project, Public Access Catalogue Extension (PACE), was in-house custom made using standard languages. The second project, Virtual Public Access Catalogue (VPACE), relied entirely on COTS software for the creation of the final product.

2. Custom software application: PACE

In a government-funded research, Public Access Catalogue Extension (PACE) was designed as a graphical interface for browsing library catalogues. PACE was created as an option for novice users, to alleviate their problems in coping with retrieval systems. The underlying assumption is that browsing is a natural and effective approach to many types of information-seeking problems, which requires a smaller cognitive load than analytical search strategies do. PACE was also designed as a visual interface because browsing is primarily visual and is dependent on patterns and shapes presented on the screen.[8]

PACE presents MARC records in a simulated environment, mimicking users' mental images of books and libraries, in an attempt to reduce their cognitive burdens. PACE is designed to augment or extend the capabilities of existing online systems, not to replace them. Users should be able to choose between a traditional OPAC interface and PACE or a combination of the two. Simulation data are derived from the physical description of books, including number of pages and height embedded in MARC records. This information is used to generate an image of a book, the dimensions of which are proportional to the actual size. The spine of each book appears on the screen as a three-dimensional image with call number and full or truncated title. Each book is assigned a colour using a hashing algorithm based on the publisher and title in order to distinguish it from adjoining books. About ten books are displayed on one screen on a simulated shelf. Depending on their width (number of pages),

fewer or more books may appear on the screen. Navigation is accomplished by allowing the user to click on the spine of a book so that a simulated "title" page is displayed. This page contains the full title, author, call number, publisher, publication date and LC subject headings derived from MARC record.

PACE was tested in a college library in an operational environment against a second-generation online catalogue (BestSeller). The results showed that for novice users, a simple browsable search mechanism worked as well as a sophisticated online system. More than 80% of the college students, however, preferred the simulated images of books and bookshelves in PACE to the traditional text-based catalogue. PACE proved successful because users readily and easily recognized the bookshelves and the books based on their experience in the libraries. They did not need to use an unfamiliar interface based on unknown mental models.[9]

The original proposal for creating PACE was written in 1991 and funding became available in 1992. At the time, since graphical user interfaces were not widely accessible and were somewhat unstable, it was decided to design and develop a bilingual version of PACE under the DOS environment. The interface, however, had to be graphical and not command driven. All the codes were written from scratch in C for the major portion of the programs, and C++ for some of the GUI segments of the project. COTS software was not widely available at the time and those in the market place were limited in their functionality and interoperability.

PACE was divided into two major sections: the back-end consisting of importing, massaging and managing the MARC records, and the front-end, which included the main interface. Two qualified research assistants were employed on a part-time basis over a three-year period, each responsible for one section. Approximately 3500 hours were spent by the assistants to code and test the programs. This time included the number of hours spent in meetings to discuss the project, specifications, and development plans; in consultation with one another to coordinate the program algorithms and codes; and testing and

reiterations. The total amount of funds spent on the educational version software, namely programming languages, was about $500.

3. COTS software application: VPACE

Virtual PACE (VPACE) extends PACE to include not only the library holdings, but also pertinent digital information available on the Web. Users can utilize VPACE to search and browse through a simulated (virtual) library, choose a book, open the title page, choose a subject heading, and see related digital information appear in a book form. This information may be a written text, a picture, a movie, a sound recording or a combination of all these media. VPACE attempts to combine the physical and digital libraries through the simulated books. As users walk virtually through the simulated library, they can browse the bookshelves. They can choose a book and open the title page. If the MARC record contains information in the 856 Tag, a marker on the title page will alert the user that the full-text or other information within the book is available and the system can automatically access this information. Users can also choose one of the Subject Headings to explore further either the local database or the Web. The results will be collected and displayed as physical items, so that users can still utilize the same familiar mental models.

Several factors prompted a decision to use COTS products to develop VPACE including limited time frame for development, standard Internet connectivity requirements, limited funds, and limited pool of expert programmers to draw from.

During summer of 1999, one full-time research assistant was employed to develop VPACE using a total of seven COTS software packages: Macromedia Director, Shells Interactive Film Arts 3D Dreams, FoxPro, EasyBase, Extreme 3D, Crossroads, and 3D StudioMax. A framework architecture was used as opposed to the traditional API approach. While in the API approach the COTS components are embedded in the custom codes, in the framework model the situation is reversed and custom codes may be embedded in the COTS products.[10]

At the end of the summer, all the developmental work on VPACE was halted. Although the front-end interface for a prototype VPACE had been developed, the product was not advanced enough for any type of testing. It was also decided that allocating more resources to improve the interface or initiating the programming for the back-end using the existing COTS software was not feasible and would not have resulted in a fully operational product.

The assistant spent about 500 hours on the project and approximately $2000 was spent on educational version of COTS products. Following is a description of each product, the procedures for their use, and the problems encountered in developing the VPACE interface.

Macromedia Director: http://www.macromedia.com/software/director/

The cornerstone application used in the project was Macromedia's Director Version 7. Director is a multimedia presentation development tool, which according to the company:

> … is the market-leading multimedia solution for creating engaging, rich media corporate presentations, e-merchandising applications, and entertainment. Director combines graphics, sound, animation, text, and video to create streaming, multi-user, interactive Web content that is easy to deploy for CD-, DVD-ROM, and the Web.[11]

Director is a popular application for creating complex, interactive presentations and courseware, simple animations, and Internet content. The Internet component is currently very popular, with Director's Shockwave and Flash used to create games and other interactive content. (see for example: www.hotwheels.com).

As can be surmised from the name, Director uses a metaphor, which is similar to that of being a director of a feature film. The various components of Director include a cast, script elements, a stage, a score and a soundtrack. The stage is the screen that includes the attributes of size and colour. The cast consists of various elements that are placed on the stage, such as graphics, sound, animations and text. The score is used to control and organize the cast members

and their actions on the stage. Scripts are programs, written in Director's programming language Lingo, which control the precise actions of the cast members and the movie's interaction with the user. There is a movie script for macro-level control. One can also associate a script with any cast member or part of the score to control more precise actions.

Lingo, Director's programming language, has evolved over the past few years. The last version before 7 had a very wordy, almost COBOL-like syntax, under the assumption that non-programmers could easily learn it. Realizing that creating sophisticated applications require a more complex programming structure, with version 7, Lingo evolved towards the object oriented paradigm, specifically towards a Visual Basic-like syntax. This transformation resulted in a compromised product. The syntax in version 7 is certainly more compact but also not quite object oriented yet. For example, the methods (functions) of one object form cannot be accessed by another at the same level. Only the controlling movie script can access the methods in objects. Another problem is in string manipulation. For the purposes of the project, functions had to be defined from scratch for changing the case of a character string and for stripping the blanks off the beginning and ending of a string.

One important feature of Director is Xtras, which are third-party programs that add functionality to Director, either in the form of new cast member or new Lingo commands, or both. There is a rich community of third party developers adding functionality for everything from string processing to networking.

It is worth noting that while Director is a very powerful tool for developing multimedia products, we did not take full advantage of its capabilities. Our movie only had a handful of active frames in the score and thirty or so cast members on the stage. Lingo was used extensively, however, to manipulate the environment and Xtras were utilized to extend Director’ s functionality for the purposes of the project.

Shells Interactive Film Arts 3D Dreams:
http://www.doitin3d.com/3ddreams/index.htm

Shells Interactive Film Arts Xtra 3D Dreams was the key ingredient to our application. It adds both cast members and commands to Director. The cast member is a three-dimensional (3D) viewport, which can be placed on the Director stage. Within that viewport (a screen within a screen), 3D rendered objects can be added in either 3D Studio Max or Microsoft's ActiveX format. 3D Dreams enables the programmer to create dynamic and interactive animation sequences rather than static and unidirectional screens. Elements can be added on the fly, as long as they are predefined as cast members, and can be manipulated and moved about the viewport using the commands that the Xtra adds to Director, such as "move3d" and "jump3d." Other components such as lights and cameras may also be included in the viewport environment. Lights control the illumination and shadows while cameras allow the programmer to control the viewport perspectives. Cameras may be moved around to give the impression that the user is actually moving through the environment and multiple cameras may be used in the viewport to change the user's perspective.

Our initial experiments involved taking one of the demo programs that Shells provided and changing it to suit our needs. The demo, a bumper car game, was an environment with user controlled movement in a ‘ room’ consisting of a floor, a fence, and a ceiling. Removing the other cars, discarding the fence and smoothing out the car's movement provided a starting point for the project.

The next step involved placing a book in the environment with an author/title image on the front cover. The book consisted of a 3D primitive, which can be created in 3D Dreams in simple geometric shapes -- in this case a box. The box was located in the center of the room and 3D letters, which were imported (see below), were placed in front of one side of the box so that they were aligned along the top part from left to right. Although a book image was now available in the environment with a readable author and title, the response time was unacceptable. it took about a minute for each book to be configured and inserted on the screen, which would not be practical in an OPAC.

To solve this problem, customer service staff at Shells was contacted and after several emails, they suggested that 3D primitives should be used with overlaying graphic textures that represented the text on the front of a book. This involved putting text in a text cast member then assigning the graphical image of the text in a field to a graphics cast member followed by overlaying that graphics image onto a 3D primitive as a texture. It was a complex and time-consuming task. Every possible permutation and combination of these procedures were tried, using various coding schemes in different scripts in an attempt to obtain the desired results.

The final solution consisted of placing the code in the script section of the template book object, from which all the displayed books are instantiated. This solution was difficult to implement and did not work properly, if the code was executed as the objects were created. The final procedure involved creating all the books from the parent object then creating and painting the textures on them one at a time.

At this time we had to upgrade the 3D Dreams software to version 2. The upgrade was mostly effortless except that the texture painting did not work anymore. All the books were stamped with the attributes of the very first bibliographic record. Once again a number of solutions were attempted in vain before the Shell technical support had to be contacted. After some consultation, the solution laid in renaming the graphics cast member each time the text image was loaded, or otherwise the first image in the cache would be used.

The next step involved making the text on the books appear consistent and much more readable. Some books, depending on the number of characters in the title, were quite clear while others were difficult to read. Many variables and combination of different attributes were used to create a consistent environment: the size of letters, number of letters on a line, margins around the text on the book image, the number of lines on the cover, etc. Unfortunately, a satisfactory solution was not found.

OCLC Cataloguing CD, Foxpro and the Marc Records

To create the prototype interface, a small database of MARC records was needed, which was not readily available in the existing testbeds. An older version of OCLC bibliographic CD-ROM was used to download approximately one hundred records, containing the string 'mcgill', in a text file. Microsoft FoxPro was used to remove the extra blank lines and to create header records in a separate database for each book. The header records had summary level information including an invented book identification number, the author and title. The two databases, header and details, could now be exported to a tab-delimited file for easy importing into another database engine.

As far as displaying formatted MARC records on the stage, it involved a fair amount of Lingo coding. A text cast member was used with an updated text property. Ideally, various fields such as publisher or author should have been underlined or italicized. Although the characteristics of the entire text could be manipulated, the characteristics of individual fields or sections of the text could not be handled independently.

EasyBase: http://www.kobald.com/EasyBase/

EasyBase is an Xtra for Director created by Austrian programmer Klaus Kobald. Since Director has no native database capabilities, EasyBase is used for data storage and retrieval. EasyBase adds new Lingo commands to Director to build and maintain a flat file database system. The datafiles are text files, which the programmer must define and manipulate completely within Director. EasyBase is relatively simple to use with a fairly sophisticated command language. Basic operations such as reading a group of records with the same key are relatively straightforward, as is appending new records to the file. Unfortunately, the documentation is rather sparse with very minimal definitions of commands. While creating, editing and appending records are simple operations, the exact sequence of commands and command parameters to do the task have not been explained properly, making each operation very time-consuming.

The two databases created in the previous step in FoxPro, were imported into EasyBase. The book images were stored in two files: a header file, and a

bibliographic file with co-ordinates of the book in the 3D environment, call number, number of pages, and dimensions.

Extreme 3D & Crossroads: http://home.europa.com/~keithr/crossroads/

Extreme 3D is Macromedia' s now discontinued graphics modeling program. The four-year old version, supplied by Macromedia, lack many of the features and functionalities of more current software such as 3D Studio Max. Extreme 3D, however, could be used to create three-dimensional versions of the letters of alphabet. Although creating the letters and basic punctuation was fairly simple, saving or exporting the files into any format that 3D Dreams could interpret, namely 3D Studio or ActiveX, was difficult. Extreme 3D could export in Autocad format, which is a common file format. Hence, a new software, Crossroads, was used to convert the graphics files.

Letters and punctuation were created in Extreme 3D, exported to Autocad format, exported to Crossroads, and then saved in 3D Studio format. These were then imported into the 3D Dreams environment with the correct size and orientation.

3D Studio Max: http://www2.discreet.com/products/products.html?prod=3dsmax

3D Studio Max is used to create three-dimensional models and simple animations. It is mainly utilized by architects and designers to create complex, interactive models of buildings and other structures. It is also used for animation applications.

Since Extreme 3D was not directly compatible with 3D Dreams, it was decided to use 3D Studio Max to build the various components of the environment such as floor, walls, ceiling, bookshelves, and other furniture. Once these elements were created, they had to be imported into the 3D Dreams environment. While reconfiguring the size and orientations was a relatively effortless task, the textures of the components created some problems. Eventually the 3D Studio Max texture files had to be renamed to accommodate 3D Dreams. Duplicating the components, such as the bookshelves, also proved time-consuming and difficult. First the new object's X, Y and Z parameters had to be adjusted to place

it in the appropriate location of the room. Second, the pitch, roll and yaw parameters had to be recalculated to correct the orientation of the object. The location of the objects, particularly the shelves, had a direct impact on the book images. The programming algorithm was designed to place books on the shelf in relation to the center of the shelf itself. In some cases, this general algorithm did not work properly and had to be modified to place the books in the appropriate locations.

4. Discussion and Conclusions

Two different methodologies were employed to produce two products. In the first instance, using custom architecture, PACE was successfully developed and tested. The second methodology, which involved COTS software, however, did not produce the desired results.

Table 1: Producing PACE vs. VPACE

Project	**PACE**	**VPACE**
Architecture	Custom development	COTS products
Time spent coding and testing	3500 hours	500
Funds spent on educational software	$500	$2000
Final product	very successful	not successful
Development time	slow	fast
Reliability	high	low
Enhancements	fairly easy	fairly difficult
Dependence on software vendor	none	completely
Needed expertise	very high	moderate
Response time / throughput	very high	low

Table 1 demonstrates the advantages and disadvantages of utilizing COTS products in developing VPACE as compared to custom programming for PACE. VPACE was developed in less than five hundred hours as opposed to 3500 for PACE, whilst the cost of software for the former was about one quarter of the

latter. It should be noted however, that the VPACE project was terminated due to unsuccessful development and problems in effective integration and manipulation of CODS software. While development time for PACE was significantly more than VPACE, the end product was considerably more reliable. Any modification and enhancements to the system were relatively easier to implement in PACE than VPACE, which required much consultation with the vendors. On the other hand, PACE required a high level of expertise and domain knowledge in coding as opposed to using CODS software in VPACE, which needed familiarity with script languages and connectivity among the products. Although VPACE was not tested in a real environment, primarily results indicated a very slow response time and throughput compared to PACE.

While much has been published regarding the cost of CODS software,[12] [13] including cost analysis models such as Constructive Cost Model (COCOMO),[14] [15] few projects have reported the actual experience of using COTS versus custom development.[16] In this project, we have compared the two methodologies and have concluded that for specific applications such as VPACE, custom architecture might be a more feasible approach than complete reliance on CODS products. Our experience and those of other developers[17] indicates that while CODS products might be suitable for applications in "known environments" custom architecture should be considered for new and unexplored projects.

Acknowledgements

This project has been supported by a grant from the Social Sciences and Humanities Research Council of Canada.

References

[1]Voas, Jeffrey. (1999). Can generic software be assured? *Computer Software and Applications Conference, 1999. COMPSAC '99*. Proceedings. The Twenty-Third Annual International. 94 – 95.

[2]Voas, Jeffrey & Payne, Jeffery. (1998) COTS Software failures: Can anything be done? *IEEE Workshop on Application-Specific Software Engineering Technology, 1998. ASSET-98*. Proceedings. 140 – 144.

[3]National Research Council. (1997). *Ada and beyond: Software policies for the department of defense*. Washington, D.C.: National Academy Press.

[4]Boehm, Barry & Abts, Chris. (1999) COTS Integration: *Plug and pray?* Computer, 32(1): p. 35.

[5]National Research Council, Canada. (1999). Using COTS software in systems development. http://wwwsel.iit.nrc.ca/projects/cots/COTSpg.html. Visited April 2000.

[6]National Research Council. (1997).

[7]National Research Council, Canada. (1999).

[8]Beheshti, J. (1992) Browsing through public access catalogs. *Information Technology and Libraries* 11(3):220-228.

[9]Beheshti, J.; Large, V.; & Bialek, M. (1996) Public Access Catalogue Extension (PACE): A browsable graphical interface. *Information Technology and Libraries* 15 (4): 231-240.

[10]Vigder, Mark R.; Gentleman, W. Morven; & Dean, John. (1996). *COTS software integration: State of the art*. Ottawa: Institute for Information Technology, National Research Council Canada.

[11]Macromedia. Retrieved March, 2000 from: www.macromedia.com/software/director/productinfo/

[12]Schneidewind, N.F. (1999). Cost framework for COTS evaluation. *The Twenty-Third Annual International Computer Software and Applications Conference, 1999. COMPSAC '99*. Proceedings: 100 -101.

[13]Voas, J. (1998). COTS software: The economical choice? *IEEE Software* 15 (2): 16 -19.

[14]Boehm, B.; Clark, B.; Horowitz, E.; and Westland, C. (1995). Cost Models for future software lifecycle processes: COCOMO 2.0. *Annals of Software Engineering*, 1:57-94.

[15]The COCOMO Suite. (1999) University of Southern California: Center for Software Engineering. Retrieved April 15, 2000 from: http://sunset.usc.edu/research/cocomosuite/index.html.

[16]Seacord, Robert C.; Wallnau, Kurt; Robert, John; Dorda, Santiago Comella; and Hissam, Scott A. (December 6, 1999) Custom vs. Off-The-Shelf Architecture. The Software Engineering Institute (SEI), Carnegie Mellon University. Retrieved April 15, 2000 from: http://www.sei.cmu.edu/publications/documents/99.reports/99tn006/99tn006abstract.html

[17]Voas, Jeffrey & Payne, Jeffery. (1998).