Lessons Learned by Use of (C)OTS

'DASIA 98'

- Data Systems in Aerospace -

Athens, Greece

May 25 - 28, 1998

Rainer Gerlich

BSSE System and Software Engineering

Auf dem Ruhbuehl 181
D-88090 Immenstaad

Phone: +49/7545/91.12.58
Mobile: +49/171/80.20.659
Fax: +49/7545/91.12.40
e-mail: gerlich@t-online.de
www: http://home.t-online.de/home/gerlich/

BSSE System and Software Engineering

Lessons Learned by Use of (C)OTS[1]

Rainer Gerlich
BSSE System and Software Engineering

Auf dem Ruhbuehl 181
D-88090 Immenstaad, Germany

Phone +49/7545/91.12.58 Mobile: +49/171/80.20.659 Fax +49/7545/91.12.40
e-mail: gerlich@t-online.de www: http://home.t-online.de/home/gerlich/

Abstract: Software reuse and Commercial Off-the-Shelf Software (COTS) is considered as a means to reduce software development costs and time. However, this does not come for free. Each such product may have its own lifecycle during which its properties may be changed. Also, a full view on the code may not be possible. In case of problems a user may need the support of the vendor - before the project finishes. Such drawbacks become even more important when developing highly reliable, fault-tolerant systems. This paper discusses pro's and con's and lists a number of problems and gives suggestions how to protect against such impacts. It also compares COTS with non-commercial Off-the-Shelf Software (called OTS here). For the reference project Linux as OTS turned out as rather stable and suitable for a commercial, fault-tolerant system while some COTS products caused some crashes. By introduction of clear interfaces and use of standards the project could survive.

Keywords: Commercial Off-the-Shelf Software (COTS), reliability, cost aspects, software development lifecycle, Linux

1. INTRODUCTION

During DASIA'97 a panel discussion about "Cost aspects of the use of commercial off-the-shelf software" (COTS) was held during which pro's and con's were discussed in view of reliable system applications. One position is that only software which has been built according to well-known (high quality) standards is acceptable for aerospace and similar applications. The other position is that (C)OTS software may be acceptable if procurement, test and acceptance procedures are properly defined and appropriate components can be found on the market.

This paper discusses more problems and solutions related to use of COTS and (non-commercial) OTS based on recent experience in a project. Especially, it identifies a number of organisational problems which have to be considered beside the principal problem of reliability and quality. Differences between COTS and OTS on one side and between (C)OTS and development from scratch on the other side are outlined from this organisational point of view.

It may be expected that by use of (C)OTS development costs and time can be saved. However, this saving has to be compared with increased procurement, test and acceptance effort and possibly with additional development effort needed to select, adapt or to integrate the (C)OTS software to specific user needs.

In any case, we have to look for sufficiently reliable (C)OTS packages either according to already available knowledge or by an evaluation exercise. We also have to take into account the mid-term and long-term availability of (C)OTS packages and we need to make our architecture robust against potential changes by suppliers.

So it has to be analysed carefully whether use of (C)OTS will be really helpful in practice and what has to be done to succeed.

2. ANALYSIS OF PRO'S AND CON'S

COTS means "Commercial Off-The-Shelf Software" and covers commonly used software packages like operating systems (OS), databases (DB), graphical user interfaces (GUI) etc. For such areas non-commercial software (OTS) is also available, for which Linux is an example. In the following, "OTS" means "freely available" software of which sources are available. For such public domain software of which sources are not provided similar conclusions are valid as for COTS.

2.1 An Example Project

During a recent commercial project COTS and OTS packages such as OS, DB and GUI were used. The project required fault-tolerance in order to ensure continuous operation all over the day because the system collects data which are needed for charging of services. Hence, loss of data means loss of money, and this is the basic motivation to achieve the needed degree of fault tolerance. So compared to aerospace or air traffic

[1] The brackets around the "C" shall indicate that also freely available, not only commercial software will be addressed.

control systems the mission-critical or safety-critical aspects are not relevant.

The short-term development schedule (seven months planned) and the restricted budget lead to the decision to use (C)OTS packages for the project. In fact, there was no other choice. And at the end, the project could be completed successfully and we had learned a lot which will lower the risk for future exercises with (C)OTS.

Due to the required fault tolerance the system architecture is nearly identical with fault-tolerant aerospace systems: two hot-redundant sets of components (processor and communication lines) with cross-coupled data channels and channels for status exchange, the capability to reconfigure after loss of a chain and to take the repaired chain into operation without any need to stop the complete system. However, the most significant difference to aerospace software was the need to select commercial hardware and as much (C)OTS as possible. The following components have been taken into account: a PC platform, TCP/IP, ISDN, RS232 and SysV-IPC as communication media, and the COTS/OTS packages Solaris/x86 [1], Linux [2], Oracle [3], Adabas [4], Borland C++ [5], GNU tools (including C compiler) [6], Teles ISDN drivers for Intelx86 [7], ObjectGEODE [8].

2.2 (C)OTS vs. Project Specific Software Development

One argument against use of (C)OTS is that it may not be sufficiently tested and hence will imply a risk for later operation. However, this conclusion may not be completely right. Software development from scratch may still include bugs due to insufficient number of tests, even if development standards and test and acceptance procedures exist. The full insight view on the software may not prevent existence of bugs when the software has passed the acceptance procedure.

On the other side, (C)OTS which is commonly used may have already been subject of a variety of tests applied by different users. Therefore, the number of remaining bugs in such software may be lower than the corresponding number of software developed in accordance with high quality standards, but tested only by one user or project.

In case of COTS the supplier defines properties of his product on which a user may rely on, at least there exists a contractual relationship which gives a user rights to get a product which is compliant with the product description. This is different for OTS.

The question may be raised now whether an "OTS" package like Linux for which nobody feels directly responsible may be appropriate for a project requiring a certain level of reliability. The specific answer in this case is: Linux as OTS turned out to be more stable (from an overall perspective) than the equivalent COTS software package Solaris/x86 or MS-Windows. And, it allowed a feasible solution (probably the only one) at the end in a puzzle of COTS components: the high variety of support given by the large Linux community and the openness of Linux lead to a higher number of potential solutions.

More generally: if sufficiently reliable less organisational problems will occur for an OTS package compared with COTS. From this view point an OTS package is similar to own software: like for own software sources are available and a user does not depend on a supplier and his policy. This becomes even more important when products from different suppliers have to be integrated.

In case of Solaris/x86 and Oracle both vendors followed own company specific policies: SUN is pushing Java [14] and gives weak support for (MS) Windows, while Oracle is pushing (MS) Windows and does (currently) not support an integrated link between Java and a database.

To summarise: apart from reliability and stability considerations an OTS package may be the better choice when going to replace own software development by (C)OTS packages. COTS packages may impose serious constraints on a project, and hence may not be helpful for the project, at all.

Finally, if (C)OTS is used more organisational problems may have to be solved than in case of project specific development from scratch. However, by adequate means such problems can be mastered if a project intending to integrate several (C)OTS packages needs another development approach and management procedures than a pure project specific development from scratch.

The pro's and con's are discussed in more detail in the next section.

2.3 COTS vs. OTS

Major cornerstones on the way to a final decision towards COTS or OTS are: responsibility for the package, openness of the package, flexibility of the supplier, saving of investments, future evolution, validity of product information.

The discussion below will show that from an organisational and management point of view more positive arguments for OTS are found. What is considered as a positive argument for COTS, may turn out as a problem when a COTS package is integrated into a more complex environment.

1. Responsibility

   In case of COTS there is a commercial partner which takes over (at least some) responsibility for a product's properties and maintenance. This is (usually) not true for OTS, e.g. in case of Linux no official responsible exists who will respond if a problem occurs. There is a community which may provide an answer, but there is no contractual relationship to somebody.

   However, the potential advantage of having clear responsibilities may become a serious disadvantage. E.g. when a problem occurs at the interface between two COTS packages it may happen that vendor 1 says it is a problem of package 2, and vendor 2 refuses to take care of the problem because he thinks it is a problem of package 1. This will cause a deadlock for the user. In case one of the packages is of type OTS a user can (may) remove the deadlock by changing the sources (preferably in an upward compatible manner).

Such experience we made for Oracle/Borland C++ and Oracle/Solaris/x86 (and also for the HP hardware platform and Solaris/x86). Oracle dropped support for Borland C++ by a later version because problems at the interface could not be solved. Solaris/x86 did not provide the support we needed for the Oracle GUI.

Missing responsibility in case of OTS may cause that COTS vendors do not to support OTS packages. So there may be a problem to integrate an OTS package with a COTS package in order to complement the functionalities. This occurred for Oracle/Linux.

2. Openness of a software package

Openness of a software package may be needed to adapt existing software to user specific needs or to identify and/or to remove a bug. Openness of a COTS package brings it closer to OTS and own software and hence lowers the risks for the project.

If a supplier is flexible enough (see point 3 below) and feels responsible like a user expects, openness may not be required because then the needed modifications will be made by the supplier. However, then more money may be needed to pay for the modification.

In such a case OTS has a major advantage because everybody can modify / extend the existing software (preferably by well-defined interfaces in order to remain upward compatible) and can enhance the product this way for all users.

The same conclusions hold in case of weak or lack of documentation: no "wait cycles" are needed until the hot-line calls back, the source code is immediately available.

In our case we had a problem with Oracle for which a problem occurred at the interface between Oracle tools. We could not solve it due to lack of information, and the vendor did not react within a period acceptable for the project. We could solve the problem when replacing an Oracle component (database) by an equivalent component (Adabas) due to standard interfaces.

Also, we detected a deadlock problem at high system load related to SysV-IPC performance which we could remove because we could change the macro thanks to the flexibility of the ObjectGeode code generator. This was rather easy to do and helped us a lot.

So openness of a COTS package will help to succeed as it minimises the risks of its usage in case a user wants to apply the package in "non-standard" manner, but still moving within the supplier's envelope. Such "non-standard" usage may already occur when a COTS package is integrated with another one.

3. Flexibility of the supplier

In case of a "non-standard" application a problem with a (C)OTS package may be solved if the supplier is willing to provide a solution within a reasonable time. When a responsible rejects to react the supposed advantage of having clear responsibilities becomes a major disadvantage.

In case of OTS either a user can help himself or he may get support from a large community.

The lesson we learned relates to a hardware-COTS interface. For the HP-PC platform [9] we got a problem with the graphics chip set. Although the S3 chip set appeared in the hardware compatibility list of Solaris/x86, it did not work for the HP-PC. The response from SUN was: yes the S3 is supported, but this HP model does not appear in the list, and we do not guarantee that our drivers will work correctly and therefore we do not give further support. The position of HP was: we delivered this model with Windows95 and there it works. We don't care about Solaris/x86.

The problem was solved at short-hand by buying an additional graphics adapter and later on by moving to Linux.

4. Saving of investments

It may happen that a certain feature disappears with the next update. This may cause serious problems for a user, if he applied the dropped feature and an equivalent feature is not provided. In case of OTS he still can access the relevant sources and may add them to the updated version.

In our case Oracle was asked for in advance whether the Borland C++ compiler is supported. The answer was yes. However, with the next update this support disappeared because both vendors (Oracle and Borland) could not solve a problem at the interface between both COTS tools. Hence, all the software written for the database application became obsolete.

Our conclusion for risk minimisation is: do not rely on specific properties of a product (here the interface provided by Oracle for the Borland compiler), you will loose a lot if the tool vendor stops support. When two vendors are involved the risk is twice as high, at least. Take a standard solution instead. If not available look for such solutions for which a standard exists. In our case we moved over to a SQL interface by which our C application communicates now with the Oracle database. This interface we can even keep when we are replacing the Oracle database by Adabas which frequently happens in our development environment.

5. Future evolution

   Some applications may require support for advanced features or features not commonly used on a certain platform. Also in this case, the potential advantage of clear responsibility in case of COTS may turn out as a disadvantage if the vendor is not willing (or ready) to provide the needed capability, even if he should be paid.

   In our case we had a problem with a COTS package for ISDN services. Only a specific Teles card was supported for Solaris/x86, and not by SunSoft itself, but by Teles drivers. It turned out that the goals of the provided ISDN software were quite different from our application and that we had to operate it in a no-standard manner. When operated this way, the package turned out as instable and we frequently got a system crash.

   We did not get support to solve the problem. The price for another specific development was too high and we also would have had a schedule problem. On the other side, we could not solve the problem ourselves due to lack of information about the COTS packages. The problem was immediately solved when we moved to Linux later on because there we had full view and could establish the needed drivers to establish the feature we needed.

6. Validity of product information

   We learned that wrong understanding of incomplete information may cause serious problems. In case of COTS the right understanding may even come after delivery of the product.

   Several times we recognised that the given information about properties of a COTS product was not like we understood it due to incomplete information. So we could only get a clear view on a product's properties when we received the package or after we had installed it. Fortunately, this happened early in the lifecycle and hence we could react on the problems without loosing too much. Provision of trial installation packages was also helpful to detect such incompliances early enough.

   In one case (Oracle) the reason was that the information about the product was not valid for all platforms, but this was not explicitly said. So the given information was "for all OS supporting Motif", but the PC/x86 platform was excluded in fact. This was recognised after software delivery when we missed the expected package.

   In another case (WABI delivered with Solaris/x86) the user was only informed during installation about the missing support for TCP/IP.

   Both cases caused serious problems for the project and the project could only survive due to Linux, an OTS package, for which the needed feature was either available or could be provided by our means.

3. A COTS PUZZLE: PRO'S FOR OTS

To further illustrate the problems which may occur when COTS package are used instead of developing an application from scratch we list a number of problems we encountered in our project because the mixture of COTS packages did not work together for a given platform.

In the given case we got following top-level requirements for the product:

R1.	PC platform
R1.a	one computer
R2.	multi-user capabilities
R3.	capability for technical evolution, open interfaces
R4.	capability for performance upgrade
	e.g. to move from PC to workstation
R5.	data base support
R6.	graphical user interface
R7.	fault-tolerance, capability for continuous operation

To meet the constraints of the budget we needed to consider (C)OTS packages.

Requirement R1.a seems to be clear a priori. However, due to the empty intersection between the considered COTS packages we seriously thought about a two-computer-platform before we found the final solution.

Due to requirements R1 and R4 we defined the OS as UNIX-based and selected Linux in a first iteration. This closed R2 and R3 as well. R7 was closed by selecting EaSySim II [10] with ObjectGEODE which provide a template to build fault-tolerant systems and support UNIX OS. To cover R5 we looked on the databases available for Linux and decided for Adabas as a commercial product (COTS) which was just provided for Linux. We did not consider the available OTS databases for Linux.

Now, to satisfy R6 we got a serious problem. It turned out that the support given by all databases on a UNIX-based platform is text-based, not graphics-based. Full graphical support is only given for (MS) Windows platforms. However, for good reasons (which were confirmed by our experience in the project, see the following chapter) we wanted to remain with a UNIX-based OS. The solution to our problem seemed to be the WABI tool which allows to run a (MS) Windows (3.1x) application on top of a UNIX OS. This was true but we needed some more iterations to succeed.

Unfortunately (as seen from now) we moved from Linux to Solaris/x86 in a second iteration for the following reason: It turned out that the Adabas GUI was not sufficient for our application. So we moved to Oracle which provides a powerful GUI called PowerObjects. Now, the next problem came up: Oracle is only available for Solaris/x86 and SCO Unix, but not (at least officially) for Linux. So we decided to move from Linux to Solaris/x86. And we ran into our next problem: The Oracle GUI runs on Windows 3.1x/WABI, but the Oracle database is not available for Windows 3.1x. Consequently, we needed communication by TCP/IP between the Oracle database running on Solaris/x86 and the GUI running on WABI/Windows. Now, we ended up with a deadlock: WABI for Solaris/x86 does not provide such TCP/IP services, hence no communication is possible between the database and the GUI.

We needed another iteration which brought us back to Linux for good reasons: for Linux support was available for a database (Adabas) and for TCP/IP communication between the UNIX and WABI/Windows 3.1x.

Finally, we got the following solution: we took Linux and Adabas, introduced WABI for Linux from Caldera [11] which supports TCP/IP and remained with Oracle/PowerObjects as GUI. This turned out as feasible and stable solution.

By this solution we also got rid of the two other problems which occurred at the interface between the Oracle components "database" (DB), the "Data Base Designer" (DBD) and "PowerObjects" GUI: both Oracle applications (DBD and PowerObjects) worked properly with Adabas, while they did not interface correctly with their own data base tool. It seems that this is (possibly was) a problem of the Oracle ODBC drivers.

4. STABILITY OF (C)OTS PRODUCTS

Due to its openness and stable multi-user capabilities we preferred a UNIX-based system right from the beginning and we did not change our mind when we got the problem with the GUI's which are only available for (MS) Windows platforms.

During the project we temporarily used a Windows95 platform because another project required Windows95. When we got crashes by our (C) database software which was under development, serious damage of the disk occurred due to lack of proper Windows95 protection mechanisms. The crashing program corrupted the disk seriously and we needed to re-install the OS and to repair the disk several times.

Then we decided to leave the Windows95 platform. We moved to Windows NT 4.0, and the number of crashes was reduced and no further corruption of the disk was observed. However, we still needed to reboot the system several times for reasons we still do not know.

This shows that COTS (in this case MSWindows) is not necessarily more reliable than OTS (in this case Linux). For Linux we did not observe any crash.

In case of GNU native Sparc C compiler we recognised some problems with finding syntactical bugs (we do not know if an improvement has been made since last summer). When moving from native Sparc as development platform to PC x86 platform a number of bugs was identified by GNU C x86 at compile and run-time which were not detected by the native Sparc C compiler.

In any case, careful evaluation is needed before a decision is made for a certain (C)OTS package in order to prevent against risks as much as possible. And this remains the responsibility of a user.

5. HOW RISKS MAY BE MASTERED

Above discussions showed that in the cases discussed above OTS is superior to COTS. However, the reader should not get the impression that this is a strict recommendation for OTS. What we learned is that OTS gives higher flexibility to react on problems when the application is a non-standard one. Then OTS comes close to own software development. The only open point is the quality.

This possible disadvantage may be compensated by the fact that (in most cases) a rather large community is using a software package which allows to achieve a rather good stability and reliability. So far, Linux as OTS did not crash, while COTS packages crashed: in a few cases Solaris/x86 for the ISDN add-on, WindowsNT for whatever reasons, and Windows95 frequently and seriously due to lack of proper protection mechanisms. Moreover, the openness of Linux helped us to solve a number of specific problems introduced by COTS packages. On the other side we were somewhat disappointed by the GNU native Sparc C compiler.

According to above experience we recommend COTS to cover standard features which do not need specific modifications or extensions, or integration with other COTS products. For non-standard applications OTS, sufficiently open COTS or own software may be more appropriate. When using COTS packages standardised interfaces should be introduced (e.g. SQL in our case) to achieve sufficient stability and independence from suppliers. This surely will be helpful for OTS packages as well.

What helped us to survive in this confusing puzzle of COTS and OTS packages and their guaranteed or virtual properties was

1. an architecture which introduced clear interfaces between the different pieces coming from different sources,

   and

2. early risk reduction by execution of prototypes already including the (C)OTS packages.

The SQL interface allowed us to combine own software and components from different vendors ending up with a more stable solution compared with the case taking all the (C)OTS components from one vendor only and using internally non-standard interfaces: we replaced the specific Borland C++ / Oracle interface by a SQL interface and we got rid of the constraints imposed by the two tool vendors. Such standards will surely keep the system stable in future (at least this is our hope), even if a vendor will change properties of his product.

Concerning (2) we could only identify the risks imposed by (C)OTS products and solve them with minimum costs because we were able to execute our own software rather early with the (C)OTS packages, and could do stress testing already in an early development phase. So we could identify the SysV-IPC performance problem already at the beginning and could solve it.

This experience leads to the specific conclusion that use of (C)OTS packages require a life cycle which takes care of the specific conditions of (C)OTS integration. Surely, the "waterfall model" [12] cannot be applied, because this would not allow early risk identification. The approach proposed by EaSyVaDe [13] for early system validation turned out to be very helpful because integration and stress tests could be done sufficiently early.

Finally, our application is rather stable now and is not compromised by the used (C)OTS packages.

6. CONCLUSIONS

A new aspect arises when a number of (C)OTS components are used: problems occur when more than one (C)OTS package shall be integrated. This requires specific care and early identification of risks. In case of own software development such problems do not occur because the interfaces can be harmonised, while in case of external software serious constraints may be imposed.

Consequently, when needing more than one (C)OTS product the real challenge of (C)OTS begins because the intersection of all selected (C)OTS components may turn out as empty at the end. Obviously, this is a new aspect which we have to take care of to succeed.

Although a lot of problems occurred in our project when using COTS and OTS, it is our (strong) feeling that we only could finish within the accepted schedule and given budget because we used (C)OTS. Especially the OTS helped us to solve a number of serious problems raised by COTS packages. We also think that even the problems described above would not justify a complete development from scratch: it is more efficient to use (C)OTS and to learn how to master the related problems than to develop already existing software from scratch again.

The iterations we needed to do due to use of (C)OTS packages added an overhead on our schedule. This has to be considered for future exercises, especially the impact on the schedule. Probably, selection and integration of (C)OTS packages will drive the critical path of a project, at least for a short-term schedule. Due to early risk identification we were ready to deliver after nine months compared with seven months planned. The delay of two months occured because we needed time to test and to identify a number of potential solutions.

We think that with increasing experience how to integrate (C)OTS and to minimise the risks, use of (C)OTS becomes more and more efficient. This will help us to make more progress in system and software development: to deliver within the schedule at acceptable costs. So we will again try it with (C)OTS packages.

7. REFERENCES

[1] Solaris/x86 2.5.1
SunSoft Inc. 2550 Garcia Avenue, Mountain View, CA 94043, USA

[2] Oracle Database Software V7.3.2,
Oracle Corporation, 500 Oracle Parkway, Redwood Shores, CA 94065, USA

[3] For download of Linux software see e.g. http://www.linux.org or http://sunsite.unc.edu/pub/Linux/kernel/linux

[4] Adabas,
Software AG, Uhlandstraße 12, D-64297 Darmstadt

[5] Borland C++

[6] For download of GNU tools see e.g. host ftp.quintus.com, directory /pub/GNU

[7] Teles S0/16 Adapter + Solaris/x86-Driver
Teles AG, Dovestraße 2-4, D-10587 Berlin

[8] ObjectGEODE SDL-Tool, Verilog, 150 rue Vauquelin, F-31081 Toulouse Cedex, France

[9] HP Vectra VL 5
Hewlett-Packard France, Coomercial Desktop Computing Division, F-38053 Grenoble Cedex 9, France

[10] EaSySim II environment, Rainer Gerlich BSSE, Auf dem Ruhbuehl 181, D-88090 Immenstaad, Germany

[11] User's Guide Wabi V 2.2 for OpenLinux, Caldera Part No: 200-WBMN-001 8/96
Caldera, 633 South 550 East, Provo, Utah 84606, USA

[12] Barry Boehm, "Software engineering" IEEE Transactions on Computers C-25, 12 (December 1976) 1226-1241.

[13a] R.Gerlich, V.Debus, Ch.Schaffer, Y.Tanurhan: EaSyVaDe: Early Validation of System Design by Behavioural Simulation, ESTEC 3rd Workshop on "Simulators for European Space Programmes" Noordwijk, November 15-17, 1994

[13b] OMBSIM (On-Board Mangement System Behavioural Simulation), ESTEC contract no. 10430/93/NL/FM(SC), Final Report Nov. 1995, Noordwijk, The Netherlands

[14] For more information see http://java.sun.com