Reasons for Migrating Legacy Systems

Mihaela Carmen TRUFASU, Project Manager – WYLOG ROMANIA
Gabriel Claudiu CONSTANTINESCU, Project Manager – KEPLER ROMINFO

The tendency for most organizations is to hold on to their legacy systems as long as possible. Even though there are new technologies that improve efficiencies, many factors contribute to inertia. Usually there are critical components at the operations level that are not addressed by off-the-shelf software. There are also historical data elements that must be retained. Then there is the cyclic nature of the processes that cannot be interrupted if the organization is to function effectively. These issues must be dealt with in realistic terms in order for legacy systems migration projects to be successful.
***Keywords**: legacy applications, migration, migration decision, application development platforms.*

The continuously evolving programming languages and the always improving programming technologies along with the development of capabilities on the behalf of the hardware components provide a wide range of deployment solutions for IT systems.

The innovation on hardware and software technology have started to compete tremendously each other in the late ’80s when computers became “personals”, by having smaller sizes and becoming more and more user friendly because their new operating systems.

The companies that have had a glimpse on the huge advantage of using machines to do the most laborious, recurrent and complicated tasks or to store data for later usage have formalized requirements and have started to implement systems tailored on their needs and operational demands.

Customized systems have been built using programming languages and hardware architectures that represented the top technology at that time.

The most powerful programming tools of that down of a new informational era were – in no specific order - COBOL, Fortran, C++, Pascal, FoxPro, PERL, dBase.

Even though the programming tasks were not easy to be fulfilled because there was no integrated development environment, no debugger or integrated linker and compiler these solutions have been tested, improved and done their duties. Time passing by, they became stable, they became efficient and they store more and more useful information.

The customized features and of lack of interoperability standards have transformed these solution into closed systems.

Only to give an example we can mention that if need be to extract data from COBOL data files the only effective way to do that was to write COBOL routines. If need be to store data to these files the answer was also COBOL.

COBOL data files do not contain information about their own organization and record structure. That is, if you have just a COBOL data file but not the Cobol program capable of writing or reading this file, you cannot correctly interpret the data contained in this file.

For every COBOL data file that you want to be able to read and interpret, you need the following:

- SELECT statement for the file. It comes from the Input-Output Section of Environment Division of the program that wrote the file. This statement tells the system how the file is organized: it contains file organization, file access mode, etc.
- FD statement for the file. It comes from the File Section of Data Division of the program that wrote the file. This statement tells the system how the file record is organized: it contains record field lengths, offsets, usages, pictures, etc.

- The COBOL data file itself.

Even if you have all 3 components as described above, reading and interpreting the data is still difficult:

- You need to extract SELECT and FD statements for the file from the Cobol program. Therefore you need a specialized Cobol parser that does it.
- You need to parse the extracted SELECT and FD statements and get file and record information from these statements. Again you need an FD/SELECT statement parser and record layout builder.
- Finally, once you have all file parameters and record layout, you need to read the actual Cobol data file and convert the record that you read to required non-Cobol format. Since some of Cobol data formats are not used by any other language or database, interpreting Cobol data is far from trivial.

Some of these languages have a same ancestor and have evolved into new programming languages. This might be the example of PERL that has evolved from C into PHP.

PHP succeeds an older product, named PHP/FI. PHP/FI was created by Rasmus Lerdorf in 1995, initially as a simple set of PERL scripts for tracking accesses to his online resume. He named this set of scripts 'Personal Home Page Tools'. As more functionality was required, Rasmus wrote a much larger C implementation, which was able to communicate with databases, and enabled users to develop simple dynamic Web applications. Rasmus chose to release the source code for PHP/FI for everybody to see, so that anybody can use it, as well as fix bugs in it and improve the code.

PHP/FI, which stood for Personal Home Page / Forms Interpreter, included some of the basic functionality of PHP as we know it today. It had Perl-like variables, automatic interpretation of form variables and HTML embedded syntax. The syntax itself was similar to that of Perl, albeit much more limited, simple, and somewhat inconsistent.

By 1997, PHP/FI 2.0, the second write-up of the C implementation, had a cult of several thousand users around the world (estimated), with approximately 50,000 domains reporting as having it installed, accounting for about 1% of the domains on the Internet. While there were several people contributing bits of code to this project, it was still at large a one-man project.

The server side script interpreting technology has improved its capabilities and performances, for example with this evolution.

All these applications that store business intelligence and the enterprises business process patterns have evolved with the operational structures and within the business structures.

If we place ourselves in the ‘90s when the Silicon Valley made history on the American technology market we see that the software development tools had to register a boom in their enhancements. We don’t know exactly if the software evolution has started the hardware revolution or the lack of performances in the software design has pushed the innovators to build better machines but we start seeing old application that cannot take benefits from the capabilities of the new hardware PC components.

Because of their design, because the development tools which have well served the last 5 years became old-fashioned and very slow when comparing to the new ones.

We seen, in the late 90’s or even sooner the need to reengineer the code, to transfer the data to new management systems to improve the code and optimize the speed. This is the meaning of MIGRATION - migration from legacy systems to new ones, from old software and software architectures to new ones that fully benefit from the hardware and software capabilities of the new systems.

When solution architects were asked to design the new system that reproduce exactly all the functions of the old one, they have say that these systems must be REWRITEN.

The lifecycle of on an application become shorter and shorter and is directly linked to the software and hardware performances. This pace is also imposed by the competition in each business, he who got the information at the best quality and at the perfect time got the power.

When designing the new application the trend of evolution in the business requirements is carefully studied, forecasts and development scenarios are ruled to simulate the response of the future system.

The first reason for migrating applications – with or without architectural redesign, with or without adding other functionalities – is TO IMPROVE PERFORMANCES.

1. To improve performances.

The migration and the redesign that aims improving performances regard: data processing speed; data saving and retrieving speed; user interoperability features.

Speed performances depend mostly on the best usage of hardware capabilities of the machines. This is assured by the usage of the best technologies and drivers for these components.

Improving performances depend on the operating system on which the application is installed; software interfaces of the hardware components; software technologies that access the interfaces of hardware components.

Before studying the ways to improve the application performances designers analyze the following: data storage; data processing; data retrieval; the way that applications access data; the way that the user interact with the system; the slowest/longest processes in the system; the balance of the response time in the average processing time.

2. To make it portable

Most of the IT systems that have been designed ten years ago were platform dependent because there were very few choices at that time. These systems were exploiting the capabilities of that platform

The hardware evolution and the movements on the IT market have led to a branch like (hierarchical) evolution of operating systems. This evolution has been really explosive when LINUX has proved well its abilities and hit the market with his open-source distribution kits.

The reliability of this open and free distributed operating system was the best thing that happened for the companies in the SOHO market segment. Computer usage became cheaper. This gave also a bump to the software industry and software vendors have to reorient their production on this direction.

The most important vendors for software development tools have oriented their efforts to provide tools for building multiplatform application.

Another business feature that requires portability is the hardware platforms variety. Within a company the processing needs linked to the IT systems differs from one department to the another, the graphical design department requires graphical stations with more capabilities on the graphical board and RAM, the financial department requires small graphical resources but more storage and processing resources, for instance.

Modules of the IT systems have to work well in both above mentioned departments and on the laptops of the employees with high mobility.

To fully get benefit from the IT system this has to be PORTABLE on various exploitation platforms – hardware and software – and to provide a high level of INTEGRABILITY. This is the second most frequently occurring reason for migrating legacy applications.

3. To add processing options impossible to implement within the current solution

There are two reasons that led to this decision:

1. the legacy system need to improve its speed, its effectiveness, its interactions etc;
2. new features are required by the business evolution, features that cannot be implemented within the current application due to the effort estimation or to the capabilities of the solution design.

An in-depth analysis of the main operations is required to decide the new system architecture that take advantage of the legacy system and uses all the legacy data as the old system has done. The decision to reorganize legacy data is to consider not only the whole transformation effort but also the effort of developing appropriate tools to translate data from the legacy structure to the new one.

In most of the situations decision makers choose a two step migration based on the multi-tier architecture to insure that the migration of the informational system will not

affect the whole business development and to spread in time the deployment costs.
The legacy data storage system is left untouched as the business layer of the application is rewritten. A data access layer is developed to retrieve and store legacy data. The business layer interacts only with the data layer at this stage of the application lifecycle.
When the new system performs well the decision to migrate towards a new data organization can be postponed according to the budgets. Only the data layer shall be rewritten if need be to change data storage system or data organization.
This kind of migration is a borderline as it is very close to reengineering which is a complex step in the application lifecycle.

4. To widen/restrain the access to the application

In the late '90s the top in designing applications was the design of client/server application. Usually there was a database server and an application installed on the user's machine. All data storage and retrieval tasks were transferred to a dedicated machine that required special features as a bigger hard disk and more RAM. The end-user, in this topology, was accessing data on the server by using a fat-client application installed on their workstations.
This was the most frequently deployment pattern for the client/server applications. The users share the same data but each of them has to run on their workstations "client" applications that implement the entire business specific logic.
Adding new features or modifying old ones involved the re-deployment of the client application on all the workstations.
The users having a high level of mobility had to install on their laptops or mobile devices applications that were called "fat" not only because all the business logic was stored in these client applications but also because of the resources that they required.
We should also think that usual laptops have offered not so much hardware capabilities in late '90s.
Moreover, the connection to the database servers for the mobiles users required the remote access at the company network, connection that lead to security gap or increasing the security costs.
These were only a few reasons for creating new system models to cope with the issues of the client/server architecture. This model is called multi-tier and most of the applications that have been migrated from client/server architecture were migrated to multi-tier web applications. This was called "webisation".

5. To allow globalization

The 1.0 release in may 1991 of Visual Basic represented a new start in the history of rapid applications development (RAD) tools. The application development acquired a new dimension because of the intuitive way of building applications in this "visual" development environment. This speed in application development and deployment has importantly decreased the cost of using such informational systems and made them more accessible to companies.
Migrating applications from Visual Basic 1.0 to Visual Basic 6.0 was an easy decision to take because every new version of the application development tool brought improvements and new design and operating features.
When globalization became a wide spread phenomenon and companies that were using these applications have wanted to open new branches in China and East Asian countries that have emerged as the latest important players in the global economy they had the huge problem with their legacy applications: these applications even though they were developed to support Multilanguage they were not able to cope with DBCS. English and European software typically use about 100 different characters to represent words and numbers. So a single character can bit store in a byte—being 8 bits.
***D**ouble-**B**yte Character Set*, is a character set that uses two-byte characters rather than one-byte characters. Some languages, such as Chinese, Japanese and Korean, have writing schemes with many different characters that cannot be represented with single-byte codes such as ASCII and EBCDIC. DBCS characters must be used with hardware and software that support the double-byte format

The only solution for this problem was the migration of these applications towards languages that have DBCS support that Visual Basic 6.0 had not.
The natural solution of this problem seemed to be the migration to the new Visual Basic version – the 7.0, version that has all the features that allow managing, displaying and processing string resources using DBCS.

6. To make it look better

Even it is hard to imagine, systems that have been built in the late '80s, in COBOL for instance, are performing well nowadays too. These systems were designed for workstations having text display and a very poor range of colors.
The decision to migrate these systems has been postponed in the time because the system was doing really well. The reason for it cannot be avoided anymore is the fact that they are old-fashioned and not so easy to use for the users who are used to have mousses and windows on their screens.
The nowadays users look for so called "user friendly" application interfaces that exploit and develop their operational habits.
Not only the COBOL written applications need improvements on the user interfaces but also those systems that have a user interface that has generated problems or misunderstandings in use.
The solution of these user interface problems may go from redesigning and rewriting the interfaces to the migration of the new system to a new development environment or to a new operating system.
The most conservative solution that we have met and built was the development of a user friendly web interface that drove the legacy system. The idea was quite simple: the user sees the web application and doesn't have a clue that behind this there is the old COBOL application running that expects exactly the same information from him and has exactly the same old and good behavior.

7. To meet a marketing goal

Consumer's rules apply around the world to the information technology as well as to all the industries. There are trends and fashions that come and go that point the users taste towards systems that are preferred because they are well known and they are sold by huge corporations that have invested big money in their marketing politics. There are also less known development platforms that perform really well comparing to the previous ones but are not used because their low profile on the market.
Most of the time the consumer preferences are linked to important performances features that are well represented into the media: reviews, websites and forums.
The independent software vendors that provide less or more tailored solutions must improve those solutions according to the software and hardware industry evolution trends and have to listen to the consumer's preferences for platforms and development tools.
Usually the migration is to be done to a new development platform or database management system that improves also the average performances of the system.

Conclusions

Most of the time when a company decides to migrate an application there is a mix of all the above reasons that led to this decision. In this above lines we inventoried only the main reasons that we have met in our ten years of experience in developing and migrating customized software applications for various independent software vendors from Europe and USA. We have also tried to extract each atomic reason from the mixture.

Bibliography

1. Dr. Vladimir Bacvanski – "An Object-Oriented, Component-based Approach to Migrating Legacy Systems", July 2004;
2. O'Reilly & Associates, Inc. – "History of Programming Languages", May 19, 2004;
3. STAR Ireland – "Introduction to Asian Translation.", August 17, 2004; http://www.star-ts.com/Asia.pdf
4. http://www.webopedia.com; http://msdn.microsoft.com/library/