8-16-1996

# AN EXPERIENCE IN THE ESTABLISHMENT OF A SOFTWARE REUSE CULTURE IN A REAL ENVIRONMENT

F. Montecinos
*Simon Bolivar University Department of Systems of Processes*, 90-22179@usb.ve

# AN EXPERIENCE IN THE ESTABLISHMENT OF A SOFTWARE REUSE CULTURE IN A REAL ENVIRONMENT

F. Montecinos
e-mail: 90-22179 @usb.ve
Simón Bolívar University
Dept. of Systems and Processes
Caracas - Venezuela

## 1. Introduction

The notion of reuse has always been present in the reasoning process of human beings. Every time that there is an attempt to solve a problem using the "experience" of previous situations, in trying to understand what is unknown by means of represen-tations or adaptations of what is known, or whenever only the elements that can help to solve our disadvantages are modified, previously acquired knowledge is being put to good use and reuse is being applied.

The concept of reuse within software engineering was defined as such by Dought McIlroy in his paper entitled *Mass Produced Software Components*, which was presented before the NATO Software Engineering Conference of 1968. From its inception, software reusability has been viewed as one of the possible ways of solving the software crisis which was declared by the defense department of the United States, based on the growing need for large-scale, reliable, cost-effective systems. In his paper, McIlroy proposed the use of a library of reusable, automated-technique components to adapt said components to varying degrees of precision and robustness.

During these almost thirty years that have passed, software reuse has not become well-established as a development standard. Although it is true that, at an individual level, all programmers take advantage of codes that have already been created, and that large-scale reusability projects in the United States, Europe and Japan have yielded good results, there is still no accepted, generalized or standardized methodology or strategy to formalize and solve all the disadvantages within a real development environment.

However, *Strategic Information Network C.A.* (SINET), a part of the H-LIGHT project, was proposed for studying the feasibility of establishing a methodology or strategy based on software reuse, that would take full advantage of the experience and knowledge acquired in new developments, in order to improve response times within the market and therefore increase their competitive levels. In order to undertake this endeavour, the system Honorarius 300 was selected as the base system and the task was oriented towards the design and development of software components and schemes that would truly comply with the idea of being reusable.

## 1. Background of honorarius 300

Honorarius 300 is a multiuse system developed under the *client-server* paradigm and designed for service companies, that automates the follow-up of hours invested and the handling of business accounts, working under a multitariff and multi-currency plan.

## 1. The development strategy

Prior to implementing any development strategy within the software reusability culture, one must be aware that this software reusability involves not only a technological change but also changes at the organization level. In order to put software reusability into practice, training is needed as well as the setting of standards and, more importantly, the necessary **commitment of the management**.

In the case of this project, the organization knew the advantages of code reuse and the management of Honorarius 300 was interested in taking advantage of all the time that had been invested in developing the system, transmitting the knowledge that was acquired through implementation thereof and in obtaining software components that could be combined in various ways to produce custom-made applications for the users in an efficient, fast and reliable manner.herefore, the first real action that was taken was to define the objectives of the project and to establish the requirements at the conceptual level for the components to be developed, as set forth below.

## 1. Requirements based on honorarius 300

These requirements are established to guarantee easy connections in the future with Honorarius 300 as well as to make full use of the knowledge acquired during the development of this system.

1. The components should maintain the coding standard already established in the project.
2. The components should take full advantage of the knowledge acquired.
3. All the new components should be 100% compatible with the Honorarius 300 system.
4. The performance of the application must not be unnecessarily sacrificed to meet the conditions of a development paradigm.
5. The interface coding technique should be changed to adapt to the new reusability requirements.
6. Requirements inherent to software reuse
7. One of the primary points for software reuse is that the components generated can be treated as blocks or blackboxes.
8. Each component developed must be free from errors.
9. The documentation concerning the components must comply with the second software reusability rule, as proposed by Krueger [1992].

*For the software reusability technique to be effective, it should be easier to reuse the devices than to create them from scratch.*

1. All the parts of the components must be reusable and those that are not must be easy to change.
2. Development tools

The next phase after having determined the conceptual framework for the development of the components was the selection of the adequate development tool. The following arguments were chosen as evaluation criteria:

1. Compatibility.
2. The level of profitable use of the tool's potentiality.
3. Experience in the handling of the tool.

It is important to highlight that the objective was clearly aimed at reusing pre-existent systems as well as the codification of new components to be conceived under the reusability plan, without sacrificing the knowledge acquired, the solutions generated nor proven and established development policies.

Taking into account all the points that have been stated, the development tool that was chosen was Microsoft® Visual Basic™ 3.0 (VB).

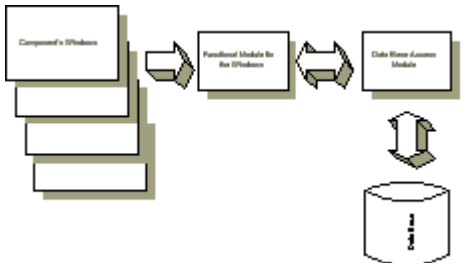
## 1. Problems to be solved

The third phase in the strategy was to design a plan for the development of the components, which would support all the requirements that had been defined and that would also be capable of being implemented with the selected development tool.

Before proposing a plan for the components, certain points had to be clarified or defined, as follows:

1. Which paradigm would be used for the development.
2. What technique, based on code reuse, would be chosen for the development of the interface.
3. How would the high isolation levels among the components be handled, in using a relational database handling device, without sacrificing the performance of the application.

In the case of the development paradigm, even though Object-Oriented Programming (OOP) intrinsically provides a good degree of object encapsulation and isolation, it was proved that this is not a necessary condition for implementing software reuse, inasmuch as these qualities can be achieved with a good design and a clear reutilization goal, so that it may developed under any programming paradigm. In consequence, an approach based on programming by modules was selected, the design of which must always take into account the software reuse requirements.

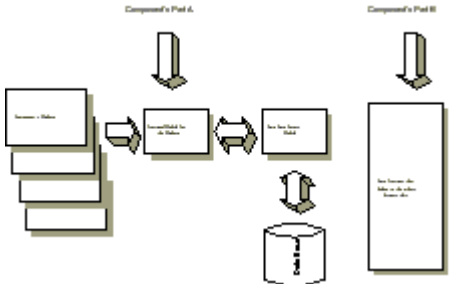


In order to solve the second point, it was decided to test a plan wherein the interface would comprise its graphic representation or forms and its functionalities, to be implemented within files for VB functions and modules, whereby the graphic representation would only require the services from its associated functional module, which in turn would be capable of communicating with the modules acting as interfaces with the database.

Regarding this latter point that remained unsolved, it was concluded that a balance had to be found between the main objectives that were being handled. An equilibrium must exist between the needs for isolation and the requirements for efficiency. Therefore, it is indispensable to have a real understanding of the universe that surrounds a component, in order to design it as part of a whole and make good use of the benefits in the work design and platform, without forgetting the reuse expectations, thus avoiding communication overhead among the components, taking full advantage of the resources available and generating components designed with a global vision, the frontiers of which would enable their reuse.

## 1. Structure of the components to be developed

The result of the entire search and evaluation process was the following structure for the development of the components:



The higher-level components in this structure model the business units and would include two fundamental parts:

1. Part A, constituted by all the screens belonging to the business unit modelling the component, its associated functional module representing the behaviour of all the screens of the component and the access module in charge of the interface between the component and the entire database.
2. Part B, comprising non-basic functionalities that should belong to other business units and which are required by part A of the component.

Apart from this, every component should have the necessary documentation for its use: a database design, internal structures, public functions, behaviour, basic and non-basic requirements regarding other components, the level of knowledge needed by the database and the more important aspects for handling thereof.

## 1. Contributions arising from this experience

During the development of this project, a set of ideas was generated regarding the application of a software reuse-based methodology.

## 1. Classification of components

The following component classification schemes were determined, as a result of the development structure thereof and the need to represent the interrelations and requirements of a component regarding the rest of the universe.

One initial classification scheme was to catalogue the components of one same project according to levels, using the information requirement dependence levels in a component as distribution parameters . Components needing information collected or processed by another component would be at a higher level than components handling their own information only.

The next scheme, based on **component reusability** levels, could be a corollary of the first scheme and is the result of analizing the feasibility of reusing components at different levels within a development environment. Components with a higher level of information requirements represent highly specialized business units and the limits of their reuse expectations are therefore well defined within similar information systems or among those that tackle one business area with similar characteristics, whereas components at a lower level of information requirements are less specific and can thus overcome the frontiers between projects of a different nature.

## 1. Isolation levels within a component

Taking into account this necessary knowledge of the universe, a scale can be established within the functions of the components, depending on the **isolation level** thereof.

The isolation levels with possible functions are the following:

1. Independent: This level represents the maximum isolation level and includes functions that have no contact with the database (DB) nor with the functions or methods of other components. Due to this self-contained characteristic, the only limiting factor for reusing this component is that it must carry out some activity that is suitable or acceptable within another system.
2. Interconnection: These are functions that do not have contact with the DB either; however, they use the functions or methods of other components. They could be affected by large-scale modifications within the components to which they are related, but under less extreme conditions the only requirement for their use is that the connection interface be maintained.
3. DB-dependent: These functions only use the portion of the DB that corresponds to their components. Therefore they could be affected by modifications that alter the DB design of their component, whether due to new requirements or to changes in the platform.
4. Dependent on the DB of other components: These functions have a knowledge as to the internal structures of other components in order to make use of the resources available to them. Their level of reuse depends on the stability of the knowledge they need within the universe.
5. Requirements and impacts of reuse within the organization

Prior to putting into effect any plan for the implementation of a reuse methodology, the organization must have a clear idea as to **what** reusability is, **what** are its benefits and limitations and its goals or objectives as to the development strategy and the reusability levels which it aspires to achieve must have been defined.

After defining the objectives, the organization must analyze its reality, the knowledge it possesses and the resources it has available, in order to determine if its goals are easily achievable. Likewise, it is necessary for the organization to be aware that its world of development creators - the designers and programmers - shall be divided in two. The development creators of reusable items have to be the best professionals within the organization. Designing a component is not a simple task and there is no clearcut recipe that can be followed.

At the end of what can be called the first stage of implement-ation, the organization has already obtained some benefits from its strategy. In fact, the question as to what is to be reused has been answered, inasmuch as reusable items have been created. The problem now is how to achieve the reuse of these within the organization. Maybe this second, less technical, more management- and human resources-oriented stage is where most of the projects are faced with their greatest challenges. In fact, for the project to survive, it is not enough to be capable of producing truly reusable devices. The manufacturing of reusable components becomes a necessary, though insufficient condition for the implementation of a software reutilization culture. It thus becomes necessary to market the components and the advantages thereof.

## 1. Conclusions

This paper has proposed a possible execution strategy, as well as a set of situations reflecting the existing conflicts between theory and implementation. However, these proposals are not put forward to discourage efforts aimed at putting into practice software reusability-based theories. On the contrary, they are only for purposes of analysis and reflection. Moreover, after this experience, we can affirm that, although not easy to reap, the results of reusability are tangible and generate a vast amount of benefits.

References available upon request from FMontecinos