Reuse Libraries for Real-Time Multimedia over the Network

Luigi Benedicenti  
Faculty of Engineering  
University of Regina  
3737 Wascana Parkway  
Regina SK Canada S4S 0A2  
+1-306-585-4701  
Luigi.Benedicenti@uregina.ca

Giancarlo Succi  
Department of Electrical &  
Computer Engineering  
The University of Alberta  
Edmonton AB Canada T6G 2G7  
+1-780-492-7228  
Giancarlo.Succi@ee.ualberta.ca

Carl Uhrik

Tullio Vernazza  
DIST – Università di Genova  
Via Opera Pia 13  
16145 Genova Italy

+39-10-3532793  
Tullio.Vernazza@dist.unige.it

George Kovacs

Andrea Valerio  
Coclea S.r.l.  
Via Magazol 32  
38068 Rovereto  
Italy

+39-0464-490201  
andrea.valerio@coclea.it

ABSTRACT
Throughout the software industry there is an increasingly critical need to reduce the costs of producing software, while at the same time providing higher quality and coping with an increasing demand for sophisticated, ultra-complex systems. Software development with and for reuse promises to address this situation.

This paper considers a case study involving a company in Northeastern Italy which undertook the implementation of a reuse-oriented, multimedia, network-distributed software entities library. It was soon discovered that unfortunately institutionalizing reuse is not a straightforward process. Despite completing the implementation and refinement of the tool, the firm encountered resistance in getting software engineers and managers to use it.

The main role in reuse was played by management taking decisions in setting up an appropriate corporate reuse policy that rendered the reuse application tool effective. This paper surveys the associated problems and suggesting potential solutions, making references to the particular case study.

1.0 INTRODUCTION
By way of an illustrative example of the introduction of reuse into a commercial software concern, this paper will outline a case study focusing on a company in Northeastern Italy which undertook the implementation of a reuse-oriented, multimedia, network-distributed software entities library. However, before going further, first some background about the principal players needs to be given.

The Software House¹ which is the subject of the case study is a very young company of moderate size (approximately 200 employees). It was created in 1993 to develop telecommunications software and is located in Northeastern Italy, the region of Italy close to the Austrian border. The company's mission is to rapidly develop high quality software for telecommunication applications, applying a general policy of reuse, object oriented (C++) programming and various state-of-the-art tools. More so than in other IT sectors, radical changes force telecom software developers to constantly update network software to cope with rapid technological evolution, global market forces, competition and growing user demand for higher quality and new services. Thus, a reuse based approach

¹ The company at present wishes to remain anonymous and thus will be referred to throughout this paper as simply "The Software House."
appeared to the management to be the magical philosophers' stone capable of transmuting costs and constraints into high quality and a minimal time to market.

Since reuse is a topic too murky to dive into without extensive outside consultation, a number of industry experts on reuse - both in the United States and in Europe - were engaged. However, remote consultants can only periodically address general needs. To facilitate a more continuous reaction, the Laboratory for Computer Engineering at the University of Trento was involved in contributing to the reuse implementation efforts. The University Laboratory was created in 1991 and since 1993 has been located in a technological park in the neighboring city of Rovereto, a small town 20km south of Trento standing between the Lake Garda and the Dolomite Mountains. Its mission is to perform computer applications research into technical & human related problems in industrial contexts.

The cooperation was established to help the Software House in its reuse centered software development environment. In particular, the University was assigned the development of a reuse library management tool which would be telematic in nature and work across a network to access distributed software items that would come from every conceivable point in the software life cycle.

Although there was a period of somewhat storming cross-indoctrination between industry and university, the cooperation was productive in the end. This was actually more of a surprise than one might think. In Italy, due to accentuated deficiencies of both academic and industrial cultures, the traditional conflicts between academic and industrial counterparts reach the highest extremes: industry people see university people as theoreticians incapable of making things work, while the university people see industry people as incapable of addressing long term goals, dedicating attention only to what pays off in the short term. Initially, this set of attitudes had the sad effect that engineers from the Software House were extremely suspicious of whatever was proposed by the University, and people at the University were showing absolutely no interest in themes extremely relevant to industry (such as execution performance, quality, formally structured requirements, standards, synchronization of themes with the soon-to-be-active projects in the Software House).

The following account will describe how it was eventually discovered that unfortunately implementing reuse is not a straightforward process but it requires a more general revision of the whole software production process. It is extremely interesting to note that this empirical discovery matches the results of an American study conducted by Frakes and Fox and published on Communication of the ACM[8]. This paper is organized in the following way: in section 2.0, an account of the tool which was designed and implemented will be given. Then, section 3.0 will describe how, despite completing the implementation and refinement of the tool, the firm encountered resistance in getting software engineers and managers to use it. Thus, some attention is dedicated to the presentation of the problems that were arising, and in section 4.0 some analysis and proposed potential solutions (already partial implemented) will be posed. In section 5.0, a brief summary of the overall picture of the reuse environment in the firm will be given. section 6.0 will describe some of problems met during the project that have not yet been properly addressed. section 7.0 some remarks about the future and going beyond the present state of affairs will be offered.

2.0 THE TELEMATICS APPLICATION

At first, software reuse seemed quite simple. The most obvious need concerned the availability of an enabling tool, such as a distributed reusable software library. The role of such a library is to provide an efficient and effective means to identify and retrieve any of the desired reusable software entities which may be located anywhere across a network. For the sake of generality and maximal benefit, software entities were not restricted to traditional software components such as code, but were conceived to be anything with a self standing identity existing in the universe of software development and maintenance (i.e., anything produced throughout the whole software life cycle). Thus, examples of software entities are:

- requirements documents
- designs and architectures
- “chunks” of source code and object libraries
- classes and class libraries
- both end user oriented and programmer oriented documentation
- pictures and animation clips
- sounds and audio tracks
- databases
test plans and test data

A closer inspection of these classes makes it apparent that there are instances of any of them that can be reused: reusing portions of code is quite common as well as cutting and pasting parts of documents and pictures; but also attaching a newly discovered sound to a pre-existing application or sharing a document template are everyday practices. So the proposed reusable software entity library needed to handle any kind of software entity, and consequently, it was required that it be multimedia.

Furthermore people at the Software House were increasingly operating in a distributed fashion: portable PCs connected through a cellular phone to data banks or the Internet were widespread, telecommuting was observed to be very popular (especially with workers seeking more flexible hours but also for a handicapped person finding it easier to work at home), and affiliated businesses and production facilities were distributed nation-wide and even world-wide. Therefore, even if it were possible to set-up and manage the library as a single concentrated site, it was clear that arranging it in a distributed framework made it much more suitable to the real need of present day programmers. Hence the library became a distributed multimedia telematics application.

The main problem connected to the software entity library was how to allow people to easily identify what is in it and what is not. Have you ever experienced how big a traditional library subject catalogue can be and how hard it can be to locate a desired book in it? The problem in a software entity library is much worse since not only is the number of elements in the library much greater, but also the aspects according to which elements are classified are much broader and more elusive. Several techniques were proposed to deal with the problem, the most notable being the faceted classification mechanism and the free text classification mechanism, but the problem of classification and retrieval is far from being solved. Primarily it is an open and ill-defined problem incorporating many aspects of language engineering as well as computer and information sciences. The goal is to provide a mechanistic linguistic framework to store and describe entities in such a way that matches later queries made by potential re-users. Its main complications come from the fact that because classifiers and retrievers may use different terminology, and, since reuse-with-modification is also permissible, close matches are desired failing exact matches. Thus, many clever enhancements on top of the classical information storage & retrieval techniques enter the picture from AI, Logic Programming, Automated Deduction, and related fields.

In addition, a more mundane innovation also greatly helped the situation: a special separate software entities library system was implemented for handling tangible items (stored non-electronically) such as reference books and manuals that inevitably are necessary to the software engineering process as well. Though with much the same appearance as the electronically accessible items, it was useful to keep separate track of physical items not directly presentable in an on-line format - but nonetheless easy to reference and to retrieve.

Now, a more refined high level architectural view of the system is suggested. There are three main components of this architecture:

• the multimedia interface
• the identification engine
• the distributed repositories interface

The multimedia interface is concerned with interacting with the user, mainly for two purposes:

• to allow the user to define a form to submit to the identification engine in order to locate the desired software entity,

• to present the user with each entity of interest found according to its nature, i.e., calling the proper editor if it is a text file or document, display the picture if it is a drawing, playing the sounds if it is an audio track, showing the movies if it is an animation clip, and so on.

The identification engine is set up to process the query submitted by the user and to locate the desired entity. The desires of the users are expressed in terms of a request form. The information used to identify the desired entity can be either located in a local database or can be stored in a remote one, or even organized across many distributed archives.

The access to the storage, be it local or distributed, is handled by the distributed repositories interface. This module mediates any access to data and has the capabilities to fetch the desired entity: in a fully transparent way it may retrieve an element stored in a local disk by means of a simple “cp” command or one placed in a remote archive through anonymous ftp, or
via "uucp," it can also access information not readily accessible on the Internet. The only difference that will be perceived by the user will be in the response time: obviously a direct copy will require seconds to be completed, an anonymous ftp minutes, while a "uucp" command hours.

3.0 OTHER PROBLEMS WITH REUSE
Roughly 16 months of requirements gathering, analysis, design, prototyping & initial implementation invested in the project funded by the Software House for the construction of a preliminary reusable software item library system, the telematics tool which was based on the ideas presented in the preceding sections. Despite the initial forecasts, the application of the library per se was not a success: while some people started using it skeptically, others were enthusiastic, but in the end, no real usage resulted. It is important to remember that the Software House is a well motivated, new software firm, born dedicated to the commitment of getting the software development process right with upper management being (miraculously) instilled with a corporate reuse imperative, so one should not pass off the lack of success to any misalignment of the firm as a whole. Thus, the cause of the failure was not immediately apparent.

Access to the reuse library system was monitored through a set of previously engineered routines intended largely for system maintenance, tuning and management reporting on effectiveness of the system. As well, interviews and polls were conducted. The results revealed the following problems that were unclear at the beginning when the reuse oriented telematics tool was being designed.

- There was an overwhelming natural reluctance towards introducing the new tool. The new application was considered disruptive because it changed the usual (ostensibly "efficient") routine by which people had operated. People needed help in quickly migrating through their personal learning curves, but they were left to fend for themselves.

- There was an intrinsic distrust of reusing others' code: people tended to distrust a piece of code developed by someone else, a manifestation of the so called NIH (Not Invented Here) syndrome. Reuse (like object oriented programming) represented a completely new perspective for developing software. To them, that new perspective goes against people's natural suspicions towards humankind. In the New York City subway system, the local police post signs that read something like: «Beware! Do not trust people you do not know: they may try to rape or rob you!». However, with reuse, something completely different was being advocated: «Appreciate someone else's work so much that you will make it part of your own work!» Such concepts are not fashionable nowadays and putting them in a production process without care caused the whole process to fail. This problem turned out to be even more acute with the poorly educated programmers in the Software House.

- The programmers in the company had an extremely competitive nature, but reuse required people to be available to cooperate. Creating high quality reusable software needs to have the cooperative effort of multiple individuals working concertedly, not in isolation.

- Almost always, more information was needed to convince someone that reuse was not going to get them into "trouble." For instance, for situations of critical reliability or performance, if no information was known, the programmer chose to rewrite the component rather than look for or produce the information needed to justify reuse.

- Similarly, copyright issues arose on several occasions (not all legacy code can be freely used without checking the terms of the contract under which it was developed). In the lack of further information being readily available, the programmers systematically elected to rewrite software components from scratch rather than make further inquires or risk being "caught" on a copyright infringement. Addressing this fear led to the realization that a well structured reuse library without a clear copyright observation policy would simply cause a series of not-too-easily solved problems for the Software House.

- The programmers' productivity was previously measured partly in terms of the number of lines of code produced. Unfortunately, this concept did not promote the process of reusing pre-existing elements. Programmers failed to see any tangible benefits to reuse, and even saw themselves as being penalized when they did practice reuse. The problem with substituting "lines of code produced" with an overly naïve "lines of code reused" (thus granting extra cash per line of reuse code) was that it encouraged people to reuse existing software.
even when it was not useful at all, lowering the quality of the code produced and increasing its development time. Keying the production of the code reused to cross-verification by somebody else, simply created clusters of friends and enemies among workers: those reusing and those not reusing each others’ work.

The essence of this discussion is that a naive introduction of a reuse library may be a starting point for applying a reuse policy in a software company, but it does not solve per se very many of the problems connected with the new way of producing software products. However, in the present case, it served to flush out deeper issues since the usage and the monitoring of the usage of a reuse library produced feedback to the managerial level of the company that was used to redesign the software development process (and even the business process to a lesser extent) within the company. This means that the real core of the reuse library, far from being the technical details of how the library is built, is the problem of organizing the whole work around the reuse policy, and this must be done at the managerial levels of an organization, not at the technical ones. However, such a task proceeds primarily from a good understanding of the potential benefits of the new approach and on what time scale they are likely to kick-in since otherwise the entire effort could simply be a big loss of money.

4.0 SOLUTIONS IMPLEMENTED

Within the Software House, after the initial sluggish attempts to promote reuse through the tool alone, reuse started to become more effective with the introduction of an expanded, more structured corporate reuse policy which was strongly facilitated by a Reuse Support Organization (RSO) and changes to the Software Development Process.

The primary revision of the Software Development Process occurred at the middle and lower levels of management aiming at the setup of the RSO to support the new telematics tool to promote design with reuse. Training and promotion of reuse (and the use of the reuse tool) were made possible by reuse advocates (based in the RSO) who operated throughout the company to give occasional lectures and often one-on-one help in effecting reuse. In this way, the reuse perspective was nurtured through specially designated personnel. Only after this support organization was in place did it make sense to introduce a revised version of the reuse tool which addressed some of the shortcomings of the previous tool:

- help facilities and expanded search facilities were added
- a means for recording and getting additional information regarding copyrights was added
- a set of statistics and figures of merit (number of past reusers, maturity level of software, performance/reliability statistics, etc.) were attached to items in the repository so that potential reusers could gain confidence that the items to be reused had some intrinsic value that justified their reuse.

Many other issues were found not to be addressed just by the application of a tool. They required a commitment from the top management of the company to enforce a reuse policy. However a proper redesign of the distributed reuse library did help the process:

- an easy-to-use, well tutored and well documented user interface overcame the natural diffidence of the targeted users
- adding a facility for management tracing of the process of locating, evaluating and reusing a software component helped in improving people skills and productivity
- a nice, shared environment acted as a people integrator, allowing users to communicate their feelings and to be in touch even when they were not at their standard locations (always more frequently the case for today’s programmers)
- institution of semi-formal reuse review boards to evaluate reuse quality and programmer performance (somewhat similar to change review boards in a configuration management process) responsible for controlling the validity of any reuse claimed on an item by item basis was useful, and on some occasions, it was found that there need be no explicit requirement for reuse when adequate justifications were provided
- attaching copyright notes and clearly explaining the international and/or cross-national laws helped people afraid of committing a crime if they would reuse a portion of code
- the Software House gave intensive training to further introduce their people to the world of reuse and draw the guidelines for the development of the
reuse centered software factory.

On this last point, the idea was to start up some elementary reuse education, trying to get to a stage where at least ad-hoc reuse was common place, through the introduction of the software reuse library and preliminary orientation sessions on reuse. Then later, a higher plateau would be reached by refining a specific reuse-centered software process and putting forth company-wide standard guidelines for producible, reusable designs and code.

5.0 A REUSE ORIENTED STRUCTURE

The idea coming out of this experience is to reorganize the whole business process [6] and software development process to take into account reuse and reusability.

Needless to say, the center of the whole process is the programmer. It is important to set up a suitable environment around programmers, where:

- specific training is performed to make them understand the general philosophy surrounding the usage of the reuse promoting tools

- the interactions among people are kept as nice as possible

- there is a specific support team in charge of addressing all the possible problems a programmer may face in connection with the application of the reuse policy and its guidelines

- the whole software production process is reengineered in order to facilitate reuse and avoid letting it be just a small isolated part of a global system completely disconnected from the rest

- lastly, there are economic incentives which reward people devoting themselves to the new reuse perspective of software development

6.0 PROBLEMS STILL NOT ADDRESSED

The final results of the project are good: future cooperation with the University are sought by the Software House and this is probably the best metric to judge the customer satisfaction.

Still lots of problems were fixed but not completely solved. We do not go through all the problems connected with requirement gathering and definitions, since lots of papers have already been written on it. We concentrate on a big issue arising from our intrinsic nature, that of a university lab.

This project well exemplifies the situation that it is likely to occur any time that a private company purchases from a research institution not just a personal consultancy of one single professor, but a «structured consultancy» from a research group; we think that this kind of activities are going to be very common in the future at least in Europe, since public funding for research are getting very limited. The point is that it is very difficult to ensure people from private company about the «quality» of the work produced by a team from a research institutes: not a lot of them have been certified ISO 9000 or have reached the higher levels of CMM, (none yet in Italy) and the only means left is to have a Software House reviewer controlling the evolution of the project. The role of reviewer is:

(a) to ensure that the people working on the project are doing their job properly, taking adequate measure to prevent errors,

(b) to find inconsistencies of the project being developed with respect to the customer requirements,

(c) to identify lacks in the requirements and to correct them redefining customer needs.

It is clear from these statement, and it is widely known, that this approach results in a nasty relationship between reviewer and developers and consequently in delays of the project, in the best cases. This is exactly what happened to us at the beginning of our cooperation with the Software House: we spent most of the time in convincing the reviewer that out programmers were not a “bunch of unskilled and uneducated hackers” and that we were doing exactly what they requested and we were not wasting their money in meaningless research, and in adapting the system to the new requirements that were emerging from reviewer analysis, mostly of the kind «the status bar should be dark blue», «the dialog box should be slightly bigger and should be placed somehow more left», one knows that just those little modifications will take long hours.

We stopped our unpleasant interaction when we decided to move our working site inside the software house itself. In this way we demonstrated visually our skills and we also established a good feeling between ourselves and our «customers»: this acted quite like a certification of our «quality», and nowadays, as it was said, the opinion this Software House has of us is high enough to “hire” us again for future problems.

However this approach did not solve the general problem and we are quite sure that next time we will have another industrial partner we will risk being
exposed to exactly the same interaction, while it is not so obvious that we will be able to hack the same solution, also because moving the working site is not such an easy task and the positive contingencies that occurred are not likely to happen again.

7.0 SUMMARY AND FUTURE WORK
This paper has described experience gained in a project on defining and applying a corporate reuse policy within an almost entirely new environment. It is the result of a joint effort of a Software House in Northeastern Italy and the Laboratory for Computer Engineering at the University of Trento. Originally, the task was seen as primarily one of introducing a multimedia, network-distributed, reuse library access tool. The two organizations involved are both very young institution, and this paper aimed to discussed the problems they faced, partly due to their own inexperience and partly due to the immaturity of the reuse field itself. Many problems needed to be addressed:

1. the problems of working with partners from university and industry,
2. the problems of definition of the needs in an ill-defined problem,
3. the problems of applying a new technology,
4. the problems of introducing and using new systems in a work place.

The project started in September 1993 and lasted until December 1994. Its major deliverable was indeed a software reuse library for which three kinds of enabling technologies were employed: classification and identification techniques similar to those used in library science, network-distributed client-server architectures, and multimedia/multi-life-cycle entity treatment of reusable items.

Unfortunately applying a reuse policy is not a such a straightforward operation. Obviously, one needs more than just a tool: one needs people that know what the tool is about, are trained to use it and are well disposed to using it. This is not to say that it is obvious how to achieve these goals. Moreover, not all of the minute consequences of the tools are very much studied. The subject of how the application of a distributed multimedia reuse library affects the entire software development process is far from being fully understood. Studies in this field are still at the empirical (practitioner) level - hence the present study. In fact, reuse is not yet even fully recognized as a research field, despite its importance and significance having been well established. Yet in the coming years, dramatic changes may occur due to the increasingly widespread usage of distributed multimedia software libraries.

In the mean time, in order to implement reuse tools and make progress towards their effective use in industry, one needs to pay attention to interpersonal and organizational aspects - especially in an immature field like software engineering, where vocabulary and practices are not unique, often differing widely even between very similar environments. Above all, one must get managers with the right attitude involved. Starting with the wrong attitude is much worse than selecting the wrong hardware or software for an application: it is much easier to have Windows NT working on a 8088 than to make a person with the wrong mindset start to appreciate the most clever intuitions about reuse. Similarly, it is hard to make lower echelon people change the way in which they work. Train people well for a systematic approach to reuse from the beginning - from the day they first set foot in the organization, make it part of their job description to be involved in reuse.

Reuse will not just happen. Reuse requires a structured corporate policy - complete with guidelines and standards and a reengineered software development process. This reuse policy requires a deep knowledge of the workers' attitudes. A corporate policy requires infrastructures. It requires promotion through advertising, examples, review processes, praise, incentives and the careful removal of reuse inhibitors. It is not just a technical issue but it involves the whole software production environment from secretaries to managers.

REFERENCES


