An Interaction Environment Supporting
the Retrievability of Reusable Software Components

Paolino Di Felice and Maurizio Properzi

Dipartimento di Ingegneria Elettrica
Università di L'Aquila, 67100 L'Aquila, ITALY

Abstract

To increase the quality of software and the productivity of programmers, the adoption of the software reusability technology is recommended. The practical application of this technology requires, on one side, a programming methodology guiding designers/programmers in the development of reusable programs and, from the other side, a system supporting programmers looking for programs, or fragments of code, to be reused.

EasyCard is a system of this kind tailored for running on personal computers. Such a system provides the user with an easy-to-use interaction environment where he can enter into and retrieve from a library information about available software components. Specifically, EasyCard features a screenframe-based data acquisition facility, a graphical query language for retrieving information about software components, and, finally, a browsing environment suitable to explore the retrieved information. Visual feedbacks about the current status of the exploration prevent programmer’s disorientation.

This paper offers a preview of the EasyCard’s interface.

1 General considerations

The annual bill for software is rising sharply. Many researchers believe that the software reusability technology constitutes the most promising approach to increase software productivity and quality. For instance, in [1] Horowitz and Munson say that such a technology has the potential for increasing software productivity by an order of magnitude.

The most common approach for promoting the code reusability consists in designing and implementing passive components (i.e., procedures, functions, units, and so on) to be reused through composition [2].

With respect to the "retrievability" of components (retrievability refers to the facility of being able to locate a desirable software component among those available in a library), limits of conventional programming environments are:

- the lack of emphasis on relations among files containing software components. Files are independent each other and are organized into directories. This hierarchical model is too poor to effectively support the retrievability of software components;
- the difficulty of associating attributes with files. Consequently, the information inside the files is not taken into account in any way and, therefore, they are treated alike.

The negative effects of limits above become dramatic when the programmer has to deal with a big project using hundreds of files spread in a maze of directories. In order to facilitate the reuse of passive components from an existing large library, a system supporting the retrievability of components of the library is very useful.

Genesis is an example of a software-engineering-based programming environment geared to support big software projects [3]. On the opposite, with respect to small size software projects the complexity (from the usage point of view) of a system such as Genesis is not fully justified. In the next section one of such a situation is sketched.

2 A case study

In early 1986, we started to develop mathematical software for sparse matrices (e.g., [4, 5]). Our software is specifically tailored for being used on MS-DOS personal computers. Reasons to be interested in having efficient and reliable mathematical software on personal computers were explained in a previous paper [6].

Software is developed according to the programming methodology given in reference [7] and, summarized in Figure 1. In such a methodology an abstract-data-type-(ADT)-based
code design strategy is adopted. In [7], the author showed that such a strategy is suitable for designing reusable Pascal units.

Let \( P \) be a mathematical problem of interest.

1. Define an ADT modelling \( P \).

Let \( O^p \) be the set of operations defined on the objects of the ADT and \( o^p \) a generic operation in \( O^p \).

2. \textbf{foreach} \( o^p \) in \( O^p \) \textbf{repeat}

   \begin{itemize}
   \item if a unit implementing \( o^p \) exists in the library then reuse such a unit
   \item else implement a unit by rearranging a unit existing in the library in a form close to what is required
   \end{itemize}

\textbf{end foreach}

Fig.1. A reusability-oriented code development methodology.

At present a mathematical library concerning sparse matrices (built in terms of reusable units designed according to the programming methodology of Figure 1) is available. More than 100 units compose such a library.

To enhance the usability of our methodology (in the development of new mathematical functionality to be added to the library), in 1989 we started a project devoted to design and implement a system (EasyCard) supporting the retrievability of components existing in the library.

Designing the system, we paid special care to:

usability: in the literature there is increasing evidence that what mostly determines the success or failure of a system is its usability and not just its functionality [8]; and

performances: a system that is too slow is unsatisfactory from the users' point of view.

In reference [6], Di Felice gave the basic requirements and the main features of an easy to use system supporting the retrievability of units belonging to a library of mathematical software.

In this paper, we report about the current version of EasyCard, specifically the paper gives a preview of the interface which implements the interaction environment of the system outlined in [6]. Throughout the paper, when no confusion arises, we use the word EasyCard to refer to the whole system and its interface as well.

3 EasyCard: the user interface

3.1 The metaphor

In a book library a "card" gives a description of a book, while a set of cards (usually collected in one or more drawers) represents the set of cards concerning books sharing common similarities (for instance, all books about music available in the library, if any). Designing EasyCard the idea was to keep the similarities existing between a unit library and a book library. Therefore, it should be not strange that the basic building block of EasyCard is the card.

A card is a collection of fields containing information about a software unit. Both with regard to the way data inside a card are used and by whom (i.e., either the user of the system or the system itself), the contents of each card may be seen as structured into two parts (Figure 2). The first part (of interest for the programmer) collects general information about the unit the card is related to; while the second part (of interest for the system) contains a list of keywords and phrases.

![Fig.2. The logical structure of the contents of a card.](image-url)

A minimal collection of data to be present into the first part of one of such a card concerns: unit name, version, unit purpose, algorithm implemented, basic data structures, implementation language, operating system, responsible designer/programmer, unit file name (i.e., the name of the file containing the source code of the unit), test cases file name (i.e., the name of the file containing the test cases for the unit).

Links exist among units. The only one kind of link implemented in EasyCard is the "flat relationship" between units. Let unit A and unit B be two units of a library, if there is some similarity between them in terms of problem solved,
algorithm implemented, data structures adopted, and so on, then we say that there is a flat relationship between unit A and unit B.

To keep the implementation of the system as simple as possible and, at the same time, to achieve good performances, in EasyCard the similarities between cards (and, hence, between units) are established by comparing their keywords and phrases. We will return on this point in Section 3.

3.2 The interaction style

EasyCard is based on a combination of menu selection and direct manipulation [9]. Specifically, the menu selection style is used to choose a command from a list of commands, while the direct manipulation style is used to operate on cards and keywords. As selection mechanism EasyCard uses the keyboard.

As stressed by Shneiderman [10], the adoption of menu selection does not guarantee that the system will be easy to use. In fact, effective menu selection systems emerge only after a careful consideration of numerous design issues, namely semantic organization, number and sequence of menu items, speed of interaction, and menu selection.

EasyCard implements a three levels (hierarchical) pull-down menu composed of 11 items. The semantic organization of menus is based on both the basic entities we have to deal with (i.e., cards and keywords) and the operations to be performed on them. A critical variable that may determine the success/failure of menu selection is the speed at which the user can move through the menus. The combination of the hot letter approach (i.e., the user is allowed to enter a command into the computer simply by typing a predefined single letter) and the keyboard, as selection mechanism, ensures a high speed of interaction.

As said above, EasyCard implements the direct manipulation approach, therefore the programmer is allowed to accomplish most of the tasks he/she is interested in simply by using the buttons that appear on the screen. Buttons implemented are: Esc, up and down arrows, Home, End, Return, PgDn, PgUp, Prnt, Tab, Shift+Tab, View, and F10. EasyCard is intended to be used by programmers developing reusable software components on MS-DOS personal computers. Hence, it is reasonable to assume that they know the meaning of the keys available on standard keyboards.

We paid special attention to associate to the buttons reproduced on the screen their standard meaning. For instance, Esc is used to abandon the current (editing/querying) task, the up (down) arrow is used to go up (down) on the screen, F10 is used to confirm the completion of a task, and so on. The buttons Prnt, and View do not correspond to any key available on standard keyboards. Their meaning will be given in the next section.

4 EasyCard: The basic functionality

Activities supported by EasyCard concern the archiving, retrieving, and updating of cards, as well as the displaying and renaming of keywords. In this paper, we restrict out attention to the basic operations available on cards, namely archiving and retrieval. A full description of the EasyCard's functionality may be found in [11].

4.1 Archiving of cards

This operation implies two composite tasks:

- introducing a card into the computer;
- linking such a card with similar cards already existing.

The first task above is under the responsibility of the programmer, while the second one is under the responsibility of the system (see Section 5).

The option Enter a card

To enter a new card into the system, the programmer has to select the option Enter a card from the menu Card (Figure 3).

![Fig.3. The option Enter a card.](image)

This causes a screenframe to appear (Figure 4). Screenframes (i.e., a set of question and answer fields collected in the same screen) are largely used for data entry. By using screenframes the programmer has an overview of the complete set of answers; therefore, he/she may select the answering order and make corrections before entering the screenframe.

4.2 Retrieval of cards

A programmer may interact with the system in order to:

1. gain knowledge about the contents of the library;
2. examine all the units of the library having a common specified similarity (e.g., they solve sparse linear systems);

3. know whether or not it exists in the library a unit that implements a specific mathematical problem. Notice that both activities 2. and 3. above are forced by the application of the reusability-oriented code development methodology of Figure 1.

To alleviate the task of extracting data from the database, EasyCard offers to the programmer a browsing-based retrieval environment where he/she can query the underlying database without being forced to use a formal query language. The graphical query language available in EasyCard trades simplicity against completeness; in other words, it is suitable to solve simple queries, however this is what the programmer mostly needs during the software development activity.

The adoption of the browsing technique has been proved to be particularly suitable for the retrieval of multimedia documents [12]. In our context, this technique is expected to be equally effective since the cards we deal with are a special case of multimedia documents. For a general discussion on advantages and limitations of browsing the reader is referred to the existing literature (e.g., [13]).

When the programmer selects the option Retrieve cards (from the menu Card), three possibilities may arise (Figure 5):

- the system displays all the cards stored in the database (option All cards);
- the system displays a subset of all cards stored in the database, selected according to a list of keywords entered by the programmer (option Select keywords);
- the system displays a specific card (option Give a name). Notice that in EasyCard the name of a card is identical to the content of the Unit name field.

Hereafter, we concentrate on the second possibility above since, in most cases, the programmer is interested to see the cards sharing common similarities (i.e., sharing the same set of keywords).

The option Select keywords

When the programmer chooses the option Select keywords, the system shows him/her the list of all the keywords stored in the database. The programmer can browse within these keywords (browsing-within-keywords) and select (by means of the button Return) as many keywords he/she likes.

Notice that in EasyCard the conditions on keywords are in and, therefore only conjunctive queries can be posed against the database. For instance, the query of Figure 6 may be rephrased as follows: "retrieve all the cards implementing, in Turbo Pascal, the Gustavson multiplication algorithm."

When the programmer pushes the button F10, the system computes the set of cards that satisfies the query and returns it to the programmer. Figure 7 shows an example. From such a figure, we can see that 15 cards satisfy the query of Figure 6. The cards are collected in a (logical) drawer. 5 cards at a time are displayed.

By using the appropriate buttons, the programmer can move up and down inside the set of cards appearing on the screen (browsing-within-cards). The bottom window (of Figure 7) displays the content of the field Unit name of the current card (i.e., the card that appear in reverse on the screen).

When a particular card seems to be of interest for the programmer, he/she can stop (by pushing the button F10) the browsing-within-cards activity and enter inside the card (browsing-within-a-card). Figure 8 shows an example. Buttons PgDn (Page Down) and PgUp (Page Up) have to be used to see the two screen pages that compose a generic card of the database.

By entering the hot letter v (associated to the button View), the code of the underlying unit is displayed. Finally, the button Prnt allows to print (on paper) the contents of the card on the screen.

As we can see from Figure 8, one of the items composing a card is the full path name useful to locate the source code of the unit the card refers to.

From the browsing-within-a-card activity, the programmer can jump back (by pushing the button Esc) to the browsing-within-cards activity and proceed to the exploration of other cards. As pointed out in the literature (e.g., [10]), it may be a source of confusion allowing the user of a system to switch from one incomplete activity to another one. To prevent programmer disorientation, EasyCard adds a check mark to cards already examined (Figure 7).

5 EasyCard: an overview of the system

Figure 9 depicts the architecture of EasyCard. The external level of the system is the programmer interface, i.e., that part of the system that takes care of data acquisition, data manipulation, and data presentation; while the internal level of the system is the part responsible for the internal representation and storage of cards and links. The implementation of the external level is finished. At present, we are working on the implementation of the internal level.

Figure 10 gives a pictorial representation of the three random files used for storing the information the system has to deal
Dasic data stmctures:

Classical LU factorization
Classical row by column multiplication
Direct method
Gauss Jordan reduction
Gaussian elimination
Gustavson multiplication algorithm
Iterative method
Linked row-wise data stmcture
Matrix addition
Matrix inversion
Matrix multiplication
Sequential row-wise data stmcture
Sparse matrix

Algorithm implemented:

Each sparse matrix is represented by using a compact data stmcture composed of three arrays.

Implementation language:
Turbo Pascal rel.4.0
Operating System:
MS-DOS
Responsibile designer:
Paolino Di Felice
Responsibile programmer:
Maurizio Properti
Unit's DOS name:
PSMGAS89
Unit's test cases file name:
PSMGAS89.IT

Fig.4. The (first) screenframe of a card.

Fig.5. The option Retrieve cards.

Fig.6. An example of a query.

Fig.7. The set of cards satisfying the query of Fig.6.

Fig.8a. The first screen of the card PSMGA89.

Fig.8b. The second screen of the card PSMGA89.
with; namely, cards, keywords, and links among them. Some comment about these data structures is sufficient to understand how cards are linked each other in our system.

The External Level

The Internal Level

Fig. 9. The architecture of EasyCard. In the figure, rectangles represent software modules.

Each component of the file cards contains the first part of a card (Figure 2), that is the only part to be returned to the programmer during a retrieval session. For a given keyword (say kj) belonging to the file keywords, the associated numeric field is a pointer to a chain of records of the file links containing the pointers to records of the file cards having kj as a common keyword. The value "-1" (conventionally associated to the first record of the file links) denotes the end of a chain.

Major characteristics of the data structures of Figure 10 are:

- simplicity; that is, to implement operations on them is immediate;
- no data redundancy. Storage optimization and prevention of data inconsistency are the major benefits inherited by the absence of data redundancy.

6 Conclusions

In software projects reusability should concern all stages of the project, in other words an attempt should be made to reuse, whenever possible, requirements, design, code, and test cases. The card-based system presented in this paper is appropriate for supporting the reuse of code and test cases with respect to small size software projects.

Major features of the EasyCard interface are the following:

1. a screenframe-based card acquisition facility useful for programmers during the introduction of data about software components;
2. a graphical query language useful for the retrieval of information (i.e., cards) about software components. Such a query language trades simplicity against completeness;
3. a browsing-based environment useful for the exploration of the retrieved cards. To prevent programmer’s disorientation visual feedbacks about the current status of the exploration are used;
4. the adoption of the direct manipulation approach. The in a sense obvious consequence of that is a satisfactory usability of the system. A long term experimentation of use of EasyCard will be essential to see if our first impression is supported by facts;
5. a large use of colors (we are implementing EasyCard on a 286 MS-DOS personal computer equipped with a color monitor). We expect that this feature of the system will produce good reactions on its users.
**EasyCard** is being implemented by using Turbo Pascal (version 5.0) [14] and by adopting the text mode. We preferred the text mode, in place of the graphics mode to keep the implementation of the external level of the system easier and to achieve better performances. In the implementation of a complex mathematical functionality is not possible to say, a priori, how many times a programmer has to switch from the standard Turbo Pascal Integrated Development Environment to the EasyCard environment. However, it is obvious that if the total switching time (from one environment to the other) is long this will cause programmers dissatisfaction.

The EasyCard system resembles to the standard Turbo Pascal IDE that we currently use in the implementation of mathematical software. For instance, both interactive environments use hot letters to select an option from a menu. Special attention has been paid to keep as much as possible similarities between these two environments because our final objective is to present to the programmers (involved in the construction of reusable software components in accordance with the reusability-oriented methodology of Section 2) the EasyCard interaction environment as a natural extension of the standard Turbo Pascal IDE.

The cost of a hard disk, useful to store the database of cards, is the only expense required to adopt such a system, in fact its implementation may be easily "done at home" as we are doing. By the way, the size of a card is 1165 byte, therefore with respect to a software library composed of 100-1000 units we need from a minimum of 113.7 Kbyte up to 1.1 Mbyte. The availability of a hard disk on a personal computer allows to keep track of a library of 1000 units without any problem.

At present, we lack data that give evidence of the effectiveness of EasyCard in the whole process of reusable software development. As a first step, we plan to collect empirical data about this matter by providing students of the Computer Science Laboratory of the Engineering Faculty of our university with the EasyCard system.

**Acknowledgements.** This work has been supported by the Ministry of Public Education under project Mathematical Software for Sparse Matrices on Personal Computers.

**References**


