Human-Machine Interaction in a CASE Environment
Paulo Gomes, Francisco C. Pereira, Paulo Paiva, Nuno Seco, Paulo Carreiro,
José L. Ferreira and Carlos Bento
CISUC - Centro de Informática e Sistemas da Universidade de Coimbra
Departamento de Engenharia Informática - Polo II
Universidade de Coimbra - Portugal
pgomes@dei.uc.pt

Abstract
The complexity of software design raised the need for intelligent design assistants, especially in Computer Aided Software Engineering tools. But an efficient and effective design system capable of helping the software designer must address important issues in human-machine communication. At the AI Lab of Coimbra University we are developing an Intelligent Computer Aided Software Engineering tool intended to work like a design assistant. In this paper we address several issues in human-system interaction, which we explore in the context of our work.

1 Introduction
Design is a complex and ill-defined task [Tong and Sriram, 1992]. Most of the times designers have to deal with several different types of knowledge and reasoning processes, leading to a design process that has no well defined development rules. These findings are also true for the area of software design, especially in the last decade, where the complexity and dimension of software systems has increased significantly. One of the first solutions to this problem was the development of Computer Aided Software Engineering (CASE) tools. These tools can help the automation and management of some basic design processes, functioning more like an electronic paper and pencil tool. But the pressure on the development companies has increased due to competition, making them to develop software in less time and resources.

Several solutions have been proposed to the software development problem, software reuse [PrietoDiaz, 1993; Coulange, 1997] was one of them. The reuse of previous developed software was one of the big promises made by the software engineering area. But it did not work as it was planned, which in our opinion was due to several issues. One of the reasons for this failure was the lack of integration of the reuse tools with CASE tools used for development. Reuse tools need to be a ‘natural’ part of a CASE environment, so that a designer can use them in the normal development environment. In this paper we focus in the interaction between the software designer and the CASE tool.

Designers tend to improve with experience. Most of the experienced designers reuse old solutions in new projects in innovative ways, developing creative artifacts. We think that Case-Based Reasoning (CBR [Aamodt and Plaza, 1994; Kolodner, 1993; Maher et al., 1995]) is a good reasoning mechanism for implementing an Intelligent CASE tool. Some of our initial assumptions, is that in order for a system to be used, it must have to be integrated in a common CASE tool, and it would have to be human-centered. Another important issue is the tool flexibility and control. This issue required that the tool would have an adequate reasoning framework that would be flexible enough for the designer to use. The control of these tools would have to be made by the designer. Basically the designer would have to decide when and how she would like to use the design assistants.

To achieve these requirements we have used the reasoning framework provided by the CBR methodology. This framework is flexible enough to combine and integrate different types of reasoning and design subtasks. It also can integrate different types of knowledge. The CASE tool developed is called REBUILDER.

In this paper we explore the interaction between members of the design team, which involves: the software designer, REBUILDER and the Knowledge Base Administrator, which will be described later on. In the next section we describe REBUILDER focusing in it’s architecture, knowledge base, CASE environment and knowledge base maintenance module. Along with the presentation of these modules we discuss several issues related with mixed-initiative intelligent systems. Section 3 describes and example of REBUILDER’s use, from the designer’s viewpoint, and the next section compares our system with related systems. The paper ends with a conclusion section summarizing the most important points of our system and presenting some future plans.

2 REBUILDER
REBUILDER has two main goals: centralize the corporation’s design knowledge, and provide the software designer with a design environment capable of promoting software design reuse. This is achieved with CBR as the main reasoning process, and with cases as the main knowledge pieces. This section describes REBUILDER, detailing it’s architecture, knowledge base (KB), CASE environment and KB maintenance module.
2.1 Architecture

REBUILDER is based on a client-server architecture comprising two servers and two clients (see Figure 1). The KB used in REBUILDER comprises the WordNet server and the file server, while the clients comprise similar modules. The main difference between clients is that the manager client has an extra module allowing the KB maintenance. There can only be one server of each type, and only one manager client. The number of designer clients is limited by hardware resources.

The WordNet [Miller et al., 1990] server comprises the WordNet ontology and the Case Indexes. WordNet is a general ontology used in REBUILDER to index cases using semantics. It also enables the assessment of semantic similarity between concepts, used in REBUILDER for case similarity. The case indexes are used for fast retrieval of cases from the case library. These indexes are associated to cases and to pieces of cases also, enabling flexible retrieval.

Clients request cases from the file server, which are in a centralized repository called case library. Each file represents an Unified Modelling Language (UML, [Rumbaugh et al., 1998]) design. This enables the client to work only with the strictly necessary cases. For the optimization of this process the case indexes play a crucial role. The data type taxonomy is used for comparing data types, and is a simple taxonomy of the main Java data types.

The UML editor, the KB manager module and CBR engine constitute the manager client (the designer client is equal to this client except that it does not have the manager module).

The CBR Engine is the reasoning module of REBUILDER. This module comprises six different parts: Retrieval, Design Composition, Analogy, Design Patterns, Verification, and Learning. All these modules are detailed in subsection 2.3.

As defined before there are two types of system users: KB administrator and software designers. While there should be only one KB administrator (or a KB administration team), there can be several software designers. The KB administrator has the responsibility of keeping the KB updated, consistent and functioning with a good performance level. The software designers use the system to develop software models. The system tries to be a design assistant for both type of users. For the administrator it acts by providing advices and suggestions about what cases should be added or deleted from the case library, or providing other KB maintenance operations. The designer and the system work more like a design team.

REBUILDER is very flexible, providing the designer several design actions, like: retrieving similar designs, reusing designs, verifying or evaluating designs. The main logic for this cooperation, is to delegate in REBUILDER tasks that the designer has some problems to achieve (or are even impossible), like searching for similar designs in a huge design repository. The same happens with the KB administrator, it would virtually impossible for the administrator to inspect all the cases in a case-base with several dozens of design cases, or even to compute the coverage of the case-base. These more demanding tasks are left to the system, and final decisions based on the information provided by the system are made by the human parts of the team.

2.2 Knowledge Base

The Knowledge Base (KB) module comprises two different servers: file server and WordNet server, reflecting the main type of requests to the KB: WordNet related requests and case library requests. Most frequent client requests to WordNet server are: retrieving case indexes and computing conceptual distances between software objects. Predominant requests to the file server are: computing data type similarities and retrieving case files.

REBUILDER uses UML to describe the design cases. UML is a software design specification language that has various types of diagrams. These diagrams are used to describe different aspects of a software system. In our system we have selected to focus on class diagrams, since they are one of the most used type of UML diagrams. Class diagrams can have four type of elements: packages, classes, interfaces and relations. A case comprises a package that can have several other class diagram objects. An example of a design case describing part of a public library application is shown in figure 2. Cases are stored in a case library in a linear way.

Communication is a very important point in the development of REBUILDER. Most of approaches to software reuse use specific or design languages that are not standard in software development. We have decided to use one of the most used software specification language: UML. The system must adapt to the designer and not the opposite. So, one of the initial requisites of REBUILDER was that it could work with UML diagrams. For the software designer the system look as an UML editor with some additional functions and tools that manipulate UML, making the system usage natural and easy.

This is related to the issue that the system must be used by designers, otherwise it can be a very good system, but if it is not used then it is worthless.

WordNet is used in REBUILDER as a common sense ontology. It uses a differential theory where concept meanings are represented by symbols that enable a theorist to distin-
BUILDER, which makes case storage and manipulation, critical. Figure 3. It shows part of the WordNet structure with synsets identified by nine digit numbers. Each index has the file name of the case to which the indexed object belongs. Since an object has a context synset, we use this synset to index the object in WordNet. This way, the retrieval of cases is performed as WordNet being an indexing structure, and then the client can request the file server for the retrieved cases.

The data type taxonomy is a hierarchy of data types used in REBUILDER. Data types are used in the definition of attributes and parameters. The data taxonomy is used to compute the conceptual distance between two data types.

2.3 CASE Environment

The CASE environment comprises two parts: the UML editor and the CBR engine. The UML editor is the front-end of REBUILDER and the environment where the software designer develops designs. Apart from the usual editor commands to manipulate UML objects, the editor integrates new commands capable of reusing design knowledge. These commands are directly related with the CBR engine capabilities.

The CBR engine is the responsible module for all the cognitive functionalities of REBUILDER. It comprises six different submodules, each one implementing a different cognitive process that helps the software designer. These submodules are:

Retrieval The retrieval submodule selects from the case base a set of cases ranked by similarity with the designer’s query. It enables the software designer to browse through the most similar designs in the case library, exploring different design alternatives and reusing pieces of cases if wanted. This module works like an intelligent search assistance, which first retrieves the relevant cases from the case library and then ranks them by similarity with the query. The cases are presented to the designer only after the cases are ranked. For more details see [Gomes et al., 2002a].

Analogy The analogy submodule generates new solutions using analogical reasoning, which involves selecting case candidates from the case library, then mapping them with the query diagram, and finally transferring knowledge from the source case to the problem diagram, yielding a new diagram. This mechanism generates solutions using only one case, which constraints the type of solutions that it can generate. For more details see [Gomes et al., 2002b].

Design Composition The design composition submodule also generates new solutions from cases in the case library. The main difference to analogy generated solutions is that it can use more than one case to generate a solution. This mechanism can select pieces of cases and then compose them in a new diagram, yielding a solution to the designer’s query.

Design Patterns The design patterns submodule applies software design patterns to diagrams using CBR to auto-
mate the selection process. This functionality is a main-
tenance or evolution mechanism that can be used by the
software designers to expand or modify an existing sys-
tem. For a more complete description see [Gomes et al.,
2002c].

**Verification and Evaluation** This submodule comprises two functionalities: verification and evaluation. While verification checks the design coherence and correctness. The evaluation mechanism is used to assess the diagram’s properties. The verification is mainly used in combination with analogy or design composition to look for errors in the generated diagram and to correct them. The evaluation mechanism is at the designer’s disposal for listing the design properties, trying to identify shortcomings in a diagram. See [Gomes et al., 1998] and [Gomes et al., 2003a] for more on case verification. REBUILDER does not have an awareness about the cognitive state of the human designer. But if it had, then it could have a proactive behavior or act according to the designer preferences. There is one aspect of REBUILDER that has some similarities with awareness, and that is the verification mechanism. In the verification process the systems learns through the storage of verification cases what the designer considers to be correct or incorrect semantically. Every designer has its own verification case-base, which allows the CASE environment to adapt to the designer preferences. This was one way to cope with the different ways in which software designers model the same problem.

**Learning** The learning submodule implements several case-based maintenance strategies that can be used by the KB administrator to manage the case library contents. This submodule presents several assessment measures of the case library performance, which provide an important advice to the administrator regarding the addition or deletion of cases. See [Gomes et al., 2003b] for more on the learning mechanisms of REBUILDER.

The main philosophy of REBUILDER is that it should act as an intelligent assistant and that all the main decisions are taken by the human users (designers or administrator). REBUILDER can provide the software designer several tools that help her/him to develop the software model. The usage of these tools is very flexible, and there is no imposition of a methodology for using these tools. This is how we view the design process, like a cognitive task that has no rigid or pre-defined steps that must be taken (if there were then it can be partially automated or even fully). The tools provide design alternatives to the designer helping him to explore the design space. Or they can also perform verification and evaluation tasks providing mechanisms for verification. The system has no proactive behavior, but it is a good direction for future improvements.

### 2.4 KB Maintenance Module

The KB Manager module is used by the administrator to manage the KB, keeping it consistent and updated. This module comprises all the functionalities of the UML editor, and it adds case-base management functions to REBUILDER.

**Table 2:**

<table>
<thead>
<tr>
<th>KB Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KB Functions</strong> Create, open and close a new KB.</td>
</tr>
</tbody>
</table>

**Case Library Manager** Opens the Case Library Manager, which comprises functions to manipulate the cases in the case library, like adding new cases, removing cases, or changing the status of a case.

**Activate Learning** Gives the KB administrator an analysis about the contents of the case library. REBUILDER uses several case-base maintenance techniques to determine which cases should be added or removed from the case library.

**Settings** Adds extra configuration settings which are not present in the normal UML editor version used by the software designers. It also enables the KB administrator to configure the reasoning mechanisms.

One last issue that we think it is important, is the maintenance of REBUILDER, which is essential in order to the system to work properly and efficiently. This task is responsibility of the KB administrator that uses the KB manager module to do it, together they form another team, the maintenance team.

### 3 Example of Use

This section describes an example of how the system can be used, focusing the most important modules of the CBR Engine. This example is about the designer client of REBUILDER, showing how the designer can be aided by the system.

Suppose that a designer is starting the design of a information system for a high school. She has already the system’s analysis done, in the form of use cases1. From these use cases, some initial entities are extracted by the designer and drawn in the REBUILDER system. Figure 4 shows the initial class diagram, representing one of the system’s modules (scheduling). This module is responsible for handling the information data about teachers, classes and rooms timetables.

One of the tools available to the designer is the retrieval of similar designs from the case library. The designer can

---

1 It is a UML diagram type used for describing functional requirements.
Figure 5: The most similar case from the retrieved ones, the number in brackets is the similarity score.

retrieve three types of objects: packages (a complete class diagram, what we call a case), classes or interfaces. Imagine that she selects the package object and clicks on the retrieval command. REBUILDER retrieves the number of diagrams defined by the designer (in this situation three cases). Figure 5 presents part of one of the retrieved cases (the most similar one), notice that in the diagram’s name there is the similarity score with the target design.

The retrieved designs can help the designer exploring the design space, aiding the assessment of the different alternative designs. Or she can go further and use the adaptation mechanisms of REBUILDER to generate new designs. The adaptation mechanisms are: analogy, design composition and software design patterns. Suppose that the designer selects design composition, figure 6 shows part of a diagram generated by this mechanism. As can be seen, it used the most similar case (case in figure 5) to build this new case, and then completed it with the missing objects.

Generated diagrams can have some inconsistencies, which can be fixed using the verification module. For example, suppose that, in the generated diagram (figure 6) the relation between class Teacher and class Timetable is out of context. The verification module checks four different knowledge sources to assess the relation’s validity: WordNet, design cases and verification cases. In WordNet the system looks for a relation between the Teacher synset and the Timetable synset, if it is found then the relation is considered valid. Otherwise, the system searches the design cases for a similar relation (a relation of the same type between the synsets of classes Teacher and Timetable). If the algorithm fails to find it, the next step is looking in the verification cases. It searches for a verification case describing the validity of a similar relation. A verification case can have two outcomes: success or fail. This way, if the algorithm finds a similar verification case and the outcome is success, the relation is considered valid, otherwise is considered invalid. If in the end the relation is considered invalid, then the designer is asked for a judgement about the relation and a new verification case is generated and stored in the case library.

After verification, if the designer considers the diagram correct and ready for being stored in the KB, for later reuse, then she can submit the diagram to the KB administrator. This new diagram goes into a list of unconfirmed cases of the case library. The KB administrator has the task of examining these diagrams more carefully, deciding which are going to be transformed into cases, going to the list of confirmed cases (ready to be reused by REBUILDER), and which are going to the list of obsolete cases not being used by the system.

4 Related Work

One main difference of REBUILDER to other research works on CBR for software reuse is that most of previous works are on code reuse and do not address design reuse. Nevertheless there are some systems that can be considered related.

Déjà vu [Smyth and Cunningham, 1992] is a CBR system for code generation and reuse using hierarchical CBR. This work is tailored for a very specific type of software reuse: plant vehicle control software. Thus dealing with a different type of programming paradigm (procedural one). CAESER [Fouqué and Matwin, 1993] is another code reuse CBR tool. It works at the code level and uses data-flow analysis to acquire functional indexes. CAESER is also developed for procedural code, thus not dealing with Object-Oriented design.

Another CBR system was developed by González [Fernández-Chamizo et al., 1996], which presents a CBR approach for software reuse based on the reuse and design of Object-Oriented code. This system does some design reuse, but it’s main focus is on code reuse, design reuse is as a byproduct of code reuse. Besides this difference, González work is based on a specific description language, while our approach is based on UML, which is a standard and widely used software design language.

The approach developed by Althoff and Tautz [Tautz and Althoff, 1997] is one of the few approaches that address all the phases of software development. They reuse system re-
requirements and associated software development knowledge, in the form of electronic documents. This approach is very different from the previous ones and from our approach also, because it deals with document retrieval and indexing, thus having a more knowledge management approach. One important limitation of Althoff’s approach is the lack of reuse or revise reasoning phases.

5 Conclusions

This paper presents REBUILDER, a design system that forms a design development team with the software engineers. We have described it’s architecture and various reasoning modules. Several issues concerning the mixed-initiative on intelligent systems have also been discussed from REBUILDER point of view.

Two important functional rules of REBUILDER are: the designer has always the decision power, and the control sequence of events is flexible but is decided by the designer. These rules stem from the view of REBUILDER as an intelligent design assistant. Though we think that the decision capabilities should always be on the designer side, the system can gain with a more proactive behavior. This behavior could help the human-machine communication, because it would model the designer’s cognitive state and will try to suggest actions based on it. Another future development is to develop more cognitive tools that could help the designer in her/his task, trying to perform tasks that are hard for the human (due to an overload of knowledge or due to the repetitive aspect), leaving her more time for creative tasks.

Acknowledgments

This work was partially supported by POSI - Programa Operacional Sociedade de Informação da Fundação Portuguesa para a Ciência e Tecnologia and European Union FEDER, under contract POSI/33399/SRI/2000, by program PRAXIS XXI. REBUILDER homepage is http://rebuilder.dei.uc.pt.

References


