Application of an object repository as an example of co-operative knowledge processing

G. L. Kovács, CIM Research Laboratory, Computer and Automation Institute, Hungarian Academy of Sciences, H-1518, Budapest, POB 63, Hungary, Phone: 36 1 209 6143, Fax: 36 1 1667 503, email: Gyorgy.Kovacs@sztaki.hu

G. Jacucci, Laboratorio di Ingegneria Informatica, Dipartimento di Informatica e Studi Aziendali, Università di Trento, via Zeni 8, I-38068 Rovereto, Italia, Phone: 39 464 443140, Fax: 39 464 443141, email: Gianni.Jacucci@lil.unitn.it

A. Lo Surdo, E. Mambella, c/o Sodalìa S.p.A, v.Valentina Zambra, 1, 38100 - TRENTO (Italy); Phone: 39 461 316 200 – 116 ; Fax: 39 461 316201;email: {als;mambella}@sodalìa.it

M. Ronchetti, G. Succi, C. Uhrik, G. Cardino, Laboratorio di Ingegneria Informatica

Abstract

In our paper we discuss a system which takes advantage of the possibilities of object oriented software design and implementation on the one hand and it assists the users in using the same pieces of software several times on the other hand. This is the so called software reuse, which is applied not only to the code and executable parts of software, but to all results appearing during the life-cycle of a software product. A multi-user system is with persistent knowledge about all aspects of the software to be implemented during its life-cycle. The persistent knowledge of object repositories is used by possibly different users having a web of shared understanding to result in collaborative problem solving.

Keywords

Object oriented, reuse, software reuse, repository, design, development, searching, browsing

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: [10.1007/978-0-387-35357-9_22](https://doi.org/10.1007/978-0-387-35357-9_22)

A. B. Baskin et al. (eds.), *Cooperative Knowledge Processing for Engineering Design*
© IFIP International Federation for Information Processing 1999

1 INTRODUCTION

Software reuse is getting to be a critical issue in software development. (K. Allen, W. Krutz and D. Olivier , 1990), The past years produced a great advancement in the science and application of both "design for reuse" and "design with reuse". There are different estimates on the amount of reusable software in big software factories, and most experts agree that it is quite unnecessary to start all new software projects from scratch (M. Lentz, H. A. Schmid and P. F. Wolfe , 1987 , M. Slovin and S. Malik ,1991, W. Tracz 1987).

On the other hand, the object oriented (OO) way of managing software in all phases of its development from problem definition, via requirements specification to testing, including design and implementation, offers great possibilities. The OO way of thinking and of software management makes the possibilities of reuse really useful, as this is the only way to define such objects at the very beginning of problem definition which will be alive during the whole life-cycle of the software, and which can eventually be reused in other projects later on (C. McClure, 1992).

We suppose having an Object Repository to facilitate software development. i.e., there is a repository built up from different objects mostly by means of "Design for Reuse" activities. In the following, this repository will be called OR.

OR will be a logically and physically completely distributed system working on different computers of a network, allowing several dependent, or independent users to work on the system simultaneously. It should support the application of different software production (CASE) and other (management, interface, DBMS, etc.) tools under different operating environments. It will be an object oriented (OO) system, consisting of different objects, which we call "objects" in the following. Object orientated view and OO software engineering supports software sharing, so its application seems to be the best solution to build up a repository, such as OR.

2 DESIGN WITH REUSE

In the following, we discuss some issues of the effective application of OR in software development; i.e., the "Design with Reuse" issue will be tackled. The most important issues of design with reuse are the following. The users have to:

- Make appropriate decomposition and grouping (if applicable) of knowledge about the problem to be solved (specification, code, etc.) to form parts to be used as a basis of finding solutions to the given parts of the problem. This requires a good knowledge of OR and perfect knowledge of the task.
- Find "matches" in OR, i.e. define which reusable objects of the repository correspond to the appropriately grouped problems (parts).
- Decide which object can be reused unchanged and which should be modified for reuse, decide what kind of necessary modifications should be done, and do

them by keeping the reusability of the modified objects, if it is worthwhile. To make these decisions, some reusability measures should be defined and used.

- Add all parts to the product which were not found in OR, put them into OR if it is worthwhile, and, finally make system integration.

Now we deal mostly with the "grouping" and "match-making" tasks of the Repository, as we believe that these are the most sophisticated and most important parts of reuse. Grouping is not necessarily an OR task; at the beginning, it will be done by separate means and mostly manually.

We suppose that some AI methods, such as machine learning could be applied and are applied to make classifications and clusterings in OR to increase its applicability.

Grouping of knowledge about the problem to be solved should be done using similar ideas, methods and tools to those used in the repository. However, the size of the task is different, as the Repository contains objects of several products of different levels, and in the case of a new product, we have to group together only the requirements specification.

To avoid a general misunderstanding and misinterpretation in the following, instead of the commonly used "project" we prefer to use the expression "software product", or simply "product" to characterise software products produced by "software development projects". Such projects generally mean a series of modelling and transformation activities to reach executable code, documentation, etc., from demands and definitions collected and organised into a requirements specification (RS).

3 INFORMATION CONTENTS AND PURPOSES OF AN OBJECT REPOSITORY

OR contains objects which have information about software products on different levels of abstraction, some of which are the following (from the most general to the most specific):

- Requirements specification (texts, figures, charts, etc.);
- Requirements model (e.g., use-case model and/or problem domain model);
- Problem domain model (e.g., object model);
- Analysis model (e.g., object model);
- Design model (e.g., object model and/or interaction diagram and/or state transition diagram);
- Implementation model (source code and executable code);
- Test model (test plan, code and executable code and results);
- Maintenance model (maintenance plan, code, executable code, results);

Not all the products must have all levels of abstraction. Any combination can be imagined, depending on the size of the product, and mostly, on the design and development style, how the product is done.

The reusable objects can belong to any product and to any level of the abstraction and they may be product-independent, too.

The objects are:

- sets of objects belonging to given products;
- reusable objects of all levels, not necessarily belonging to any product in OR;
- data, databases;
- other objects (see later).

Most of the above objects are not homogenous, but they are composed from various other objects, such as:

- texts (explanations, manuals, etc.);
- figures, diagrams, tables, charts, etc.;
- algorithms, pseudo-algorithms;
- pseudo-code, code, executable code;
- objects, object models;
- movies, sounds, music, etc.

To read (view, show) and write the objects text-editors and graphical editors will be used. However as a multi-media and hyper-media application, video-movies, music, other sounds, etc. will be supported, too.

There are possible other objects in OR, as:

- business plans and strategies;
- management task plans, estimates;
- baselines, change specifications;
- reuse strategies, reuse guidelines, reuse suggestions;
- reuse inhibitor reports;
- subject matter libraries;
- product shells, their documentation,
- domain model documentation;
- share objects, special software contracts;
- design audit trials;
- justifications, design rationales;
- change specification scripts;
- etc.

These are mostly non project and non product-specific and mostly textual objects.

It is obvious that until this point two basic directions were taken into account to make OR management and applications relatively simple:

- Product view, when (all) objects belonging to the same software product are taken into account together. In this case "product" could be a root super-class in the OO view.
- Abstraction level view, when (all) objects of the same level of abstraction - independently of their origin, from where they were generated belong together. In this case the "abstraction levels" could be the OO root super-classes.

Using any other classification/clustering methods, arbitrary number of further object-groups can be created and managed. Consider just two simple examples:

- alphabetic order of the identifiers of the objects (it does not seem to be very practical, unless for simple searching);
- fields of application (in this case several products may be bound together).

Machine learning possibilities can effectively assist in making more useful, applicable clusters, than our simple examples. ((It could be the topic of detailed studies, how to make useful, logically consistent, manageable size clusters.))

A very simple example of such process is the following: we consider the events of having the elements of the repository (given objects) that are accessed by some users within a short amount of time. If the same event frequently occurs, a link will be established among the objects related to the given event, so that each time a user accesses one of these objects, he or she can see all the other objects connected to it.

Now we suppose to have a big OR full of reusable objects of all types, mentioned earlier, clustered and classified appropriately. Our main task is to start a new project, i. e., a new "product development process".

The new product is understood from the point of view of applying OR. If it is a "really" new product development, it means starting with the requirements specifications. In this case all objects at all levels could be reused, for which matches can be found for the requirements specification of the new product.

On the other hand the application of OR may be started at any level of abstraction (for example only for design and below, only for coding, or only for testing, etc.). For different sizes (granularities) of products (product, part-product, block, module, etc.), this problem will be discussed later on in detail.

4 SEARCHING THE OBJECTS

To address the main task, we have to find an optimal (or at least good) search strategy to use as many objects of OR as possible, or as many as are worthy. Actually the goal is neither to maximise the number of used objects, nor to minimise the length (or weight) of the necessary new development for the product. Actually the goal is to find a way to guarantee a fast, cheap, reliable software development with good, reliable, understandable software results that correspond

to all requirements and expectations on the one hand and to all general and local standards on the other hand.

In order to identify an object inside the library, there is the need to define some terms, using which the object can be searched. Such terms can be partitioned into three classes. The first class specifies the identity of the object, i.e., what it is by itself. The second class describes the relations between objects: such relations can be structural as well as semantic: structural relations are the ones which deal with the structure of an object as it is (examples of this class are inheritance and aggregation) whereas semantic relations are the ones that target the concept captured by the object (an example can be two objects being related by a cause-effect connection). The third class comprises the associations between objects that result from the genetic process. In the proceeding of this paper we will call *identifier of an object* the set of terms of these three classes, that help in finding the desired object.

4.1 Basic search and browse strategies

In the case of "small" products, some simple manual search and browse methods could yield good results. However this could be time consuming in the case of a big Repository.

Suggested **browsing** strategies:

- through all project names;
- through a given project;
- through a given level of abstraction;
- through given level of abstraction of a given project;
- through all identifiers;
- through all programs of a given language;
- through all objects of a given reuse level;
- through all objects of a given identifier;
- through the objects of any of the type;
- etc.

Searching can be done according to any definition applicable, such as names of objects (objects), identifiers or object-object relations, etc.

Showing means to show an object of any kind. Generally it will mean the activation of text and graphical editors to show details, as deep as needed. I. e., there may be several layers of information, where the deeper levels may be accessed by means of choosing icons or other symbols on the upper levels.

Showing (and/or searching) activities:

- show an object;
- show two objects;
- show given objects;

- show objects with given object-object relations;
- show objects with given descriptors;
- show connections;
- show projects (in which a given object is involved);
- show requirements (show the objects with the given requirements);
- show documentations (show the objects having the given documentation);
- show identifiers (all identifiers connected to the given object).

4.2 More intelligent searching

For "big, complex" products, some intelligent assistance should be given, as in these cases the above given basic search, browse (and show) procedures might not be appropriate. Some expert/decision support systems (AI methods) could assist to use OR in the best way with high efficacy.

As most software products are different, we have to evaluate whether it is worthwhile to develop a unique, special expert system for each product separately. The answer is not necessarily negative, as there exist such products to which the high additional investment of an appropriate expert system would result in lots of savings in the total costs of development.

On the other hand, it seems to be more promising to define the task of the planned expert (decision support) system somehow more generally:

- From the new development side, some clustering of future software products should be performed for which a common advisor (expert/decision support) system, or its appropriately tailored version can be used. It is possible if we know that most new developments will be done in the same - or a similar - field, as e.g., telecommunication or mechanical engineering design.
- From the advisor side some generality should be allowed to be tailored according to the given development to be assisted

We are convinced that a good compromise can be achieved in which such an advisor can be developed which is applicable to a relatively wide area of applications with minor tailoring for each application.

Tailoring will mean in most cases to change only some parts of the knowledge base and data base of the system, so we can speak about our planned advisor as an extended expert system shell.

5 ARCHITECTURE OF THE SYSTEM

Such an advisor will have its place between the OR and the USER INTERFACE as it is shown in Table 1. The advisor should be invisible (transparent) for the user if he or she does not need it, and it should afford further services if needed.

Table 1: Architecture of an Object Repository

<i>Layer</i>	<i>Purpose</i>
1	User
2	User Interface
3	Advisor
4	Object Repository

To define the main tasks of the advisor, we have to define what kind of services are needed for a new software product development which cannot be done by the previously given browse/search/show possibilities.

The Concerto Software factory of the SEMA Group (France) approaches new software development from a different way by giving a set of integrated services supporting development mostly in Ada language, using the HOOD object oriented analysis/design methodology. However Concerto does not stress reuse.

The application of OR and the advisors connected to it would be a good example of Co-operative Knowledge Processing as we shall see later.

Co-operative Knowledge Processing (CKP) is the recently known and still rarely used highest level of software application taking into consideration the ingenious role of humans working with software systems.

6 LAYERS IN CKP

In much the same way as the ISO seven layer model for networking, CKP can be built up from four distinct layers - each of which is built on the preceding:

1. The lowest and most fundamental is the "communications" or "data sharing" layer. The data communications layer provides the ability to store, manipulate, and share data elements. This layer contains tools for distributed DBM and inter-process communication.
2. The second layer consists of "co-ordinating" or "result sharing". At this level the participating agents proceed largely independently but they share results using the communication layer, and they are influenced by the results of other agents. Tools which operate on this level are loosely coupled and do not have much influence on the manner of problem solving used by their neighbours.
3. The third layer consists of "collaboration" or "task sharing". At this level the participating agents understand and influence the internal problem solving

behaviour of other agents. These agents must possess a shared understanding of each other, the problem at hand, and the emerging results. While this third layer may call upon communication and co-ordination tools from the lower levels, it provides a degree of integration that is not found at the lower layers. This third layer is the first level which is able to respond to the requirements of system level problems. Indeed, the pattern of interaction among the collaborative problem solvers should mirror the pattern of interacting constraints within the system level problem being solved.

4. The fourth layer is "co-operative knowledge processing" (CKP) and it is built on the lower layers. It adds a "web of shared understanding" to the shared problem solving, result co-ordination and sharing, and data exchange layers below. This shared understanding forms the context within which true CKP can occur.

Let us take a look at a possible, intelligent application of our OR system, when a large confluence of conflicting technical, market and financial expertise must be resolved. These problems appear mostly in the early phases of product design, when a new product is defined, i.e. sometime during the establishment of the requirements specification. An appropriate tool (a good advisor and its environment, as discussed earlier and will be given later) would enable different field experts (management, market analysts, reuse experts, application and domain analysts, etc.) to simultaneously sit down together at their computers using an electronic blackboard to assist in evaluating and proposing options. The same kind of tool might enable software architects, developers and reuse oriented library hierarchy managers and librarians to enter into discourses or carry on electronic meetings with specific design objectives.

As OR is highly distributed in the logical and physical sense as well, using different computers, operating systems, software tools, database management systems, etc., in the above described activities all layers and most activities of CKP can be found, as:

- communication (= data sharing);
- sharing results in co-ordination;
- collaboration (=task sharing) (it responds on requirements of system level problems);
- and a web of shared understanding is added to the shared problem solving, resulting in co-ordination (and knowledge sharing) and data exchange.

Some details will be given in the conclusion.

7 STRUCTURE AND APPLICATION OF THE ADVISOR

An advisor, as we call the planned expert/decision support system would have different tasks - taking into account the above given CKP application ideas:

- to assist in the design process at all levels of the design of new software products;
- to assist individual users and specific groups of users in using OR;
- to assist in evaluation of some results reached already;
- to provide feedback to the user and to OR.

If a new product is concerned, than most of the problems arise at the requirements specification (RS) level. RS consists of the most diverse, heterogeneous information for a new product.

To find appropriate matches between parts of the RS of a new product and RS objects in OR is a complicated task, but it is worthwhile to try to do it, as if all (possible) matches are done, then all lower level objects are automatically present, too. At least we suppose it as a first approximation, as we assume that the products - the development of which produced the RS objects of OR - were developed completely taking into account the requirements of "design for reuse". In this case mostly system integration tasks should be solved if all matches were found and all objects of all necessary levels are available. System integration will be assisted by using OR, too.

Naturally, besides, and prior to the system integration, the missing RS parts and all corresponding lower level objects should be produced, and integration cannot be done automatically, as first the RS level should be completed and then the lower levels. In most cases this will not be a straightforward activity as feed-backs might be necessary from every lower level to the higher levels, including the RS level. The final, accepted RS and the corresponding lower level objects will compose the new product after some iterations, and, with the appropriate modifications. During this activity a great number of new objects will be generated; the reusability of all should be carefully checked and only the "good" ones should be added to OR. The rest may be either thrown away or just used as parts of the product for which they were developed.

If there are lower level objects belonging to the Repository, for which the appropriate RS pieces are not present in OR, the task to be solved is more difficult, as matches should be looked for and found not only on RS level, but on all other levels. This will result in a more heterogeneous way of building up the new product.

Based on the above given hints the basic tasks (and inputs) of our advisor can be defined, as the following:

- It has to be a distributed system, or at least a system which is automatically accessible by all users of the OR.

- It has to be an object oriented system to understand and manage the objects of the Repository.
- It has to be able to make the necessary matches during search/browse activities, and the evaluation of matches should be done by it, too.

A more sophisticated searching/browsing should be supported by the advisor and the target of the search/browse activities should be the approximately grouped (clustered, organised, classified) set of artefacts. By the nature of the task, not only the search, but the grouping should be supported intelligently.

Hence there could be at least three intelligent procedures to be performed by the advisor system:

- grouping the artefacts of the repository;
- grouping the RS (and other, if there are any) artefacts of the product under development;
- match-making (intelligent search) between the new product's artefacts and repository artefacts.

To assist in object management in OR all artefacts (objects) will have meta-data, such as descriptors, relations, etc.

These meta-data are the attributes of the objects and all search/browse activities will use them.

A possible set of useful descriptors is the following - divided into three groups:

1. Behavioural description: type, assumptions, external data consulted, other routines used, side effects, results, definitions,
2. External properties: written for reuse ?, written with reuse ?, status, certification, validation, quality, interoperability
3. Operating environment: product/project, author, dates, applied formalism, version, operating environment, free software ?

Without going into details about the meaning of the above information we can imagine several possibilities of their combination for grouping and match-making.

And the advisor should have problem specific knowledge, organised into rules to make automatic selection of subsets of descriptors (and relations, etc.) to assist multidimensional search, as only multidimensional search can be successful enough.

Grouping within the repository can be changed according to the demands of the problem to be solved and then the grouping of the target product's artefacts and of the repository's artefacts should be done according to the same criteria defined by the same subsets of descriptors.

Then the same criteria would guarantee successful match-making between groups of the target artefacts and groups of OR artefacts.

Figure 1: Users working with OR.

8 CONCLUSIONS

As a conclusion we try to show, how the application of an advisor supported OR can meet the requirements of CKP.

Figure 1 shows a general, idealistic structure of our OR system consisting of different repositories on different computers connected by appropriate network. Users working on the same project at the same time use GUI's to communicate with the distributed OR.

These users can be developers, designers, managers, eventually with different access rights. As an example let us suppose a new product in the requirements phase, when the requirements specification is already organised and formalised into a useful, coherent set of information which has already been distributed into three subparts as Figure 2. shows. Three designers are working on their distinct tasks with the assistance and supervision of a manager. They have to find reusable parts in the OR and they have to define the necessary new parts (objects) to put the new product together.

The connections between the 3 parts are precisely defined, and everybody has to take care of them (a12, b12, a13, a23 in our example)

Figure 2: Three parts of a requirements specification.

Let us see now, how the different CKP layers are present in the work of our designers and manager:

- Communication and data sharing is represented by the networked repositories which form the OR and which have the same view from the point of view of each GUI (user).
- Sharing results is present, as at each step of searching for, and finding existing artefacts to be used, and at each step of defining necessary new objects (artefacts) all the users are equally informed and all of them can take advantage of all of the results.
- Collaboration and task sharing means that independent users can work together on the same problem (part-problem), knowing and understanding the ideas of each other. Task sharing can be based on human communications (mental, written, etc.) of the agents (users), but the appropriate communication of different software packages can be imagined, too. Task sharing may mean to work on different aspects of a problem. This can be properly done by expert systems of different knowledge- bases and different (or similar) inference mechanisms. In our example we plan to combine the "human" and the "computerised" task sharing.
- The real CKP means the presence of all the three previous ways of co-operation at the same time (sharing data, results and tasks) in a network (web) of shared understanding. In this shared understanding not only computers and programs play important roles, but the role of the governing humans must be the most

important. A kind of optimal co-operation and support of each other has to be achieved.

9 REFERENCES

- K. Allen, W. Krutz and D. Olivier (1990), *Software reuse: Mining, refining and designing*, in TRI-Ada '90 Proc., Dec. 1990, pp. 222-226.
- G. Caldiera and V. R. Basili (1991), *Identifying and qualifying reusable software components*, IEEE Computer, pp. 61-70, Feb. 1991.
- M. Lentz, H. A. Schmid and P. F. Wolfe (1987), *Software reuse through building blocks*. IEEE Software, vol. 4, no. 4., pp. 34-42, July 1987.
- C. McClure, (1992), *The Three Rs of Software Automation: Reengineering, Repository, Reusability*, Englewood Cliffs, NY 1992.
- D. L. Parnas (1972), *On the Criteria to be Used in Decomposing Systems into Modules*, CACM vol. 15, no 12, Dec. 1972.
- M. Slovin and S. Malik (1991), *Reengineering to Reduce System Maintenance: A Case Study*", Soft. Eng., July-Aug. 1991, pp. 14-24.
- W. Tracz (1987), *RECIPE: A reusable software paradigm*, in Proc. 20th Hawaii Int. Conf. Syst. Sci., B. D. Shiver and R. H. Sprague, Jr., Eds., Kailua-Kona, HI, 1987, pp. 546-555.

10 BIOGRAPHIES

George L **Kovács** works in the Computer and Automation Research Institute of the Hungarian Academy of Sciences since 1966. Recently he is the head of the CIM Research Laboratory, and at the same time he is professor at the Technical University of Budapest. He got his Masters' and Ph.D. in 1966 and 1978 respectively and D.Sc. in 1997. He has more than 150 publications, he is member of different national and international scientific organisations and committees. He has been working in the U.S.A., Russia and in Germany as visiting researcher and in Mexico and Italy as visiting professor for periods of 6 months to 2 years. . His main interest is the application of AI tools in simulation, design and control of flexible manufacturing systems.

Gianni **Jacucci** received a Laurea in Physics at University of Rome - Italy (1967). Since then he has been Researcher of the Italian National Research Council (1971-1985) Professor of Physics at the School of Engineering, University of Trento - Italy (1986-1992). Professor of Informatics at the School of Engineering, University of Trento (from 1992). Visitor at CECAM (European Computing Center for Physics), Paris - France (1973-1977). Responsible of a sw group at Istituto per la Ricerca Scientifica e Tecnologica - Trento (1981-1985). Director of

the Centro di Fisica degli Stati Aggregati ed Impianto Ionico in Trento of the Italian National Research Council, active in materials science, and artificial intelligence (1983-1988). Responsible of research contracts between Università di Trento and Consorzio CAD-CAM Bolzano - Italy (1988- 1989), towards the establishment of a Research Centre for Computer Applications. Adjunct prof. of physics and supercomputing applications, University of Illinois at U-C - USA (1990-91). Director of Laboratory of Informatics Engineering, Dept. of Computer and Management Sciences - University of Trento (from 1991). Responsible of research funding in UE Framework Programs for research, development and technology transfer: ESPRIT, RACE, BRITE-EURAM, CAMAR/AIR, COPERNICUS, evaluation expert for same. Member of IFIP TC5 WG 5.2, 5.3. Prof. Jacucci's current research and academic interest are in the area of introduction of new information and communication technologies and of its interaction with the enterprise organisation.

Angela Lo Surdo was born in Rome, Italy (1953) and graduated in Mathematics from the University of Rome. In 1978, after three years of research at the University, she started working in Information Technology: for more than 10 years, she designed and implemented advanced software for large Operation Systems, based on strong Software Engineering practices; then, for about four years, Lo Surdo worked at Telesoft, as Project Manager for Telecom Italia projects. She joined Sodalia in 1993, in charge of Methodologies and then, as the manager for Software Development Infrastructure, she led the company to SEI-CMM Level 3 to become an advanced Software Factory. Lo Surdo is an authorised Lead Assessor of SEI. She is currently the Director of Research & Technology at Sodalia.

Eliseo Mambella, born 1931 in Pescara (Italy), received the Sociology Doctor Degree from the University of Rome. He has been working in Telecom Italia for 39 years, spending a tangible part of his long career with the Information System at Headquarter Department. There he has been in turn responsible for Corporate Software Development, Information Technology Planning, Architectures and Methodologies. After his retirement, since 1992, he was a full-time Information Technology Consultant to Telesoft, mainly dealing with software architecture, software engineering and tool. In 1993 he was the Research & Technology Department Director at Sodalia where is now acting as Research and Technology Consultant. He also taught software engineering methodology at Trento University and acted as project reviewer on behalf of European Community. IEEE Member.

Marco Ronchetti Assistant Professor at the Università di Trento since 1983, he has been active for long time in physics in the field of computer simulation in condensed matter. In recent years has developed an interest in computer science, in particular in Java, object-oriented methodology and Web technology.

Giancarlo **Succi** Associate Professor Department of Electrical and Computer Engineering at the University of Calgary Degree in Genova, 1988, PEng (Genova, 1989), MSc (Buffalo, 1991), PhD (Genova, 1993), Research Interests: effects of integrating a reuse policy inside a software, development process, in terms of programming language, process maturity, productivity, quality, long term returns of investment, effects on the market structure, relevant accounting techniques, and legal issues.

Carl **Uhrlik** received his Ph.D from the University of Illinois in ComputerScience, working under Ryszard Michalski and Arthur Baskin at the A.I. Laboratory. He was a Fulbright scholar in Budapestwith Joe Hatvany, applying AI to Flexible Manufacturing Systems. Subsequently, at the University of Trento in Gianni Jacucci's group at the Laboratory for Informatic Engineering, he applied computers in fields varying from software reuse repositories to agricultural expert systems. He taught briefly at the University of Phoenix in Denver, and for the past 2 years, he has worked for Berdy Medical Systems developing speech recognition front-ends to medical databases.

Guido **Cardino** is a Ph.D. student at the University of Rome in Italy. He is currently part of the research staff at the Laboratory of Information and Communication Technologies at the University of Trento in Italy. He has six years of experience in the design and programming of object-oriented system. His research interest are mainly in software architectures, especially for distributed and object-based systems, and in software measurement. He has participated, as scientific consultant and project responsible, to several projects co-founded by the European Commission in the area of software process improvement.