Network Of Reengineering Expertise – NOREX

1 Summary

In an information technology society that is increasingly relying on software, software productivity and quality continue to fall short of expectations: software systems suffer from signs of aging as they are adapted to changing requirements. The main reason for this problem is that software maintenance and reengineering is still undervalued in traditional software development processes. The only way to overcome or avoid the negative effects of aging in legacy software systems and to facilitate their smooth evolution is by providing engineers with a fully automated and integrated support for the entire reengineering process.

Unfortunately, the reengineering of large scale software systems is a highly complex activity, which demands both multiple scientific insights (*e.g.*, metrics, visualization, meta-modelling etc.) and various engineering skills (*e.g.*, compiler techniques, graphics, database engineering etc). Consequently, tackling all the issues involved in a reengineering process is excessively challenging, and barely impossible to be addressed by a single research group.

The three research groups proposing this project, and other European research groups have developed over the last 7 years valuable artifacts to support in different manners the reengineering process. Each of them have focused their attention on specific issues and provided remarkable solutions. Yet, all these scientific and engineering "gems" have a reduced impact if used in *isolation*, as they are unable to address the entire spectrum of challenges that appear in real-world reengineering activities.

In this context, the goal of of this joint research project is to provide a comprehensive and extensible support for complex, full-fledged reengineering activities applicable on real-world systems. Specifically, we want to address these issues by building a distributed reengineering environment which is able to make all the techniques and models defined and implemented by each of the three research teams to complement each other. Then we want to use this environment to integrate different reengineering techniques to support complex reengineering techniques and validate based on large-scale experiments the feasibility of the approach. Consequently, the project consists of the following tracks:

Creating a Distributed Reengineering Environment. Currently, the existing reengineering artifacts are used in isolation, due to the fact that they are separated by the usage of different representation formats of the system, different implementation paradigms, various programming languages etc. We aim to build the *network of reengineering expertise*(NOREX) as an inter-communications framework that would allow our tools to become interoperable.

Enabling Complex Reengineering Activities. Following this track, we want to explore the possibilities open by the NOREX environment to build complex reengineering analyses and workflows by combining tools built by the participating groups. Examples of planned analyses are: (1) combining problem detection with visualization to get an overview of how problems are distributed over a system, (2) combining evolution analysis with problem detection to understand the evolution of quality.

Performing Large-Scale Reengineering Experiments. The aim of this track is to validate the aforementioned project goals by performing large-scale distributed reengineering experiments. This is reached by combining the unique experience obtained by the three teams both at a human as well as at an infrastructural level, made possible on one hand by the NOREX environment and on the other hand by the complex reengineering activities defined and automated during this project. We believe that these large-scale experiments will validate the feasibility of our approach for usage in the software industry.

Aside from its scientific value, this project will maintain and consolidate the already long-lasting collaboration links between the two research groups in Switzerland and the LOOSE Research Group in Timişoara (Romania). The main reason for this is given by the intrinsic nature of the project proposal, as the proposed reengineering network is an "open-ended", continuously growing technical and scientific challenge. Due to its significant expertise in reengineering and due to the value of the produced artifacts, the Romanian research group will definitely play a key role in the network, even after this project will be finished.

We also strongly believe that impact of this project goes beyond an academic interest and it will have a significant impact also on the software industry in Romania (especially in Timişoara), an industry which is currently very much facing the *outsourcing* phenomenon, which is in many ways related with reengineering activities. By providing local software companies with an integrated and feasible support for reengineering, we would not only contribute to better software products, but also to an ehanced professional qualifications of the engineers that would take advantage of it.

2 Background

2.1 Participating research teams

In this section we want to provide a quick overview of the three research groups of the proposers and also emphasize their complementary character that contributed to their successful collaborations from the past.

2.1.1 University of Lugano - Faculty of Informatics (USI)

Background. Established in October 2004, the Faculty of Informatics [1] is dedicated to high quality research and teaching. The faculty is active in research with focus on all aspects of software-intensive systems, with particular strengths in software engineering, distributed systems, and networks. The main focus of the software engineering research group lies in software reengineering, reverse engineering, with a special focus on software visualization and metrics. This group is continuing the development of CodeCrawler, a general purpose visualization environment [LD05].

People. The USI team in Lugano will consist of 2 Ph.D. students, coordinated by Prof. Michele Lanza. The two Ph.D. students that are involved in research activities related to visualisation, reengineering and software evolution. They are:

- *Mircea Lungu (24 y.o)* – His main research interest is directed towards the study of visualizations for architecture recovery.
- *Marco D'Ambros (24 y.o.)* – He is mainly focused on research related to software evolution and visualisation.

Prof. Lanza was actively participated in the writing of a landmark book in reengineering, namely Object-Oriented Reengineering Patterns [DDN02] by Demeyer, Ducasse, and Nierstrasz. It describes a series of lightweight techniques for understanding, assessing and restructuring complex software systems. He has mainly experience in building information visualization tools and approaches to deal with reverse engineering and evolution: His Ph.D. thesis, titled *Object-Oriented Reverse Engineering - Coarse-grained, Fine-grained, and Evolutionary Software Visualization* [Lan03] discusses several visualizations to understand complex software systems at various levels. He received the *Ernst Denert Software Engineering Award* in 2003 for the thesis[2]. He is the developer of *CodeCrawler*, an information visualization tool which implements *polymetric views*, semantically enriched visualizations of information *e.g.*, software. Prof. Lanza is involved in the visualization and the reverse engineering communities by contributing to the main venues such as SoftVis, WCRE, ICSM, TSE, etc.

2.1.2 Software Composition Group - University of Bern (SCG)

Background. The Software Composition Group[3] carries out research in diverse aspects of how to make systems more flexible with respect to changing requirements. Current research is focussed on (i) programming languages and mechanisms to support software evolution, and (ii) tools and environments to support the reverse- and re-engineering of complex software systems. One of the key achievements of the SCG team is Moose, a long-living reengineering environment [DGLD05]. It started in 1997 and since then it has evolved constantly. In its current form it supports many research areas like: concept analysis [ADN05], dynamic analysis [GD05], evolution analysis [GDL04], semantic analysis, dependency analysis.

People. The people that will be involved in this project by the SCG team in Bern consists of 2 Ph.D. students, coordinated by Prof. Oscar Nierstrasz. The two Ph.D. students that are planned to work on the NOREX project are:

- *Tudor Gîrba (27 y.o)* – His main research interest is directed towards understanding and assessing the quality of a systems by analyzing its evolution.
- *Orla Greevy (35 y.o)* – Her research focusses on the use of dynamic traces to analyze relationships between software artifacts.

[1] see http://www.inf.unisi.ch/
[2] see http://www.denert-stiftung.de
[3] see http://www.iam.unibe.ch/∼scg/

Oscar Nierstrasz is a Professor of Computer Science at the Institute of Computer Science (IAM) of the University of Bern since 1994, where he leads the Software Composition Group together with Stphane Ducasse. Prof. Nierstrasz is the author of over a hundred publications and co-author of the book Object-Oriented Reengineering Patterns [DDN02] (Morgan Kaufmann, 2003). Prof. Nierstrasz has been active in the international object-oriented research community, serving on the programme committees of the ECOOP, OOPSLA, ESEC and many other conferences, and as the Programme Chair of ECOOP '93 and ESEC '99. He is president of CHOOSE, the subgroup on object-oriented systems and environments of the Swiss Informatics Society.

2.1.3 LOOSE Research Group - Institute e-Austria Timişoara (LRG)

Background. In spite of its name, the Institute e-Austria (IeAT) is a Romanian research institute in computer science established in December 2002 in Timişoara, Romania. Its goal is to encourage young Romanian talents to make a career in research, and to strengthen the links between people from academia and software development by technology transfer and knowledge sharing. Long-term goals are to create an environment for high quality research and technology transfer in the field of information technologies, and to stimulate the scientific, technological, and economic cooperation between Romania and the other European countries. Although in the beginning it was mainly funded by means of an Austrian governmental project, beginning with January 2006 it will rely totally on Romanian funds.

The LOOSE Research Group (LRG)[4] is one of the several departments of this institute and focused on the evolution and reengineering of object-oriented software systems. In this context, the main fields of expertise of the LRG team are: measurement and quality in object-oriented design; meta-models; high-level system restructuring; parsing of C++ and Java code. LRG has developed since 2000 iPlasma[MMM+05], a reengineering environment that currently supports in an integrated fashion various types of analyses like: problem detection, code duplication detection, data flow analysis etc.

People. The Romanian team (LRG) is planned to consist of 5 members: the team leader (Prof. Radu Marinescu) and 4 junior scientists. Three out of the 4 junior scientists planned for the team are already employed (since 2003) at the e-Austria Institute (IeAT), and all of them are also Ph.D. students (on reengineering topics) at the "Politehnica" University of Timişoara, which is a partner of IeAT. Nominally, they are:

- *Cristina Marinescu (25 y.o.)* – Her main research focus is the reengineering of object-oriented enterprise applications.
- *Petru Florin Mihancea (25 y.o)* – His main research interest is centered in combining structural analysis with data flow analysis for a more accurate assessment of a design and implementation quality in object-oriented systems.
- *Richard Wettel (30 y.o.)* – His main research goal is to asses the impact of different types of codes duplication on its quality and also on the consequent system restructuring.

In addition to the three aforementioned people, we plan to hire a fourth junior scientist (either a master or a Ph.D. student) who's main focus should be the design and implementation of distributed systems.

Radu Marinescu is a currently working both as an assistant professor at the "Politehnica" University of Timişoara and as a senior researcher at the Institute e-Austria, being a co-founder of the LRG group. In 2002 he received his Ph.D. (with "magna cum laude" honors) in computer science at the faculty where he is currently employed in collaboration with the University of Karlsruhe (Prof. Goos). Prof. Marinescu has published over 20 paper on software metrics and reengineering topics in international conferences and workshops; he is also co-authoring the book *Object-Oriented Metrics in Practice* to be published at Springer Verlag until the end of this year. This year, Prof. Marinescu acted as a member of the Program Committee at the ICSM conference, and as co-organizer of the ESF/ERCIM CHASE Workshop (in Bern).

2.1.4 Past collaborations between the teams

The three teams proposing this project have already worked successfully through the last seven years as proved by the achievements summarized below:

- Between 1997-1999 Prof. Marinescu first met and collaborated with Prof. Lanza and Prof. Nierstrasz in the context of the ESPRIT FAMOOS project, one of the first pan-european projects on reengineering. In that context, they were co-authors of the *FAMOOS Reengineering Handbook*[DD99]

[4] see http://loose.utt.ro/

- Currently, the three research groups of proposers are collaborating within the European Science Foundation network of excellence RELEASE (Research Links to Explore and Advance Software Evolution)[5] and also within the ERCIM Working Group on Software Evolution[6].
- During their collaboration Prof. Lanza (in 2004) and Prof. Ducasse (from SCG Bern, in 2002) have visited the LOOSE Research in Timişoara. Prof. Marinescu has also visited several times the research group in Bern and one time also the group in Lugano, In several occasions he was invited to give talks on topics related to reengineering.
- Prof. Lanza and Prof. Marinescu are now working together on a book (also co-authored by Prof. Ducasse), entitled *Object-Oriented Metrics in Practice* which will is expected to become available at the end of 2005 by *Springer Verlag*. Also, last year Prof. Marinescu co-authored several conference papers [RDGM04] [GDMR04] in collaboration with members of the SCG in Bern.
- Two students from Timişoara wrote their diploma theses as collaborations between the LRG Timişoara and the SCG in Bern. One of them, namely Mircea Lungu, is now a Ph.D. student in Lugano, under the supervision of Prof. Lanza. Another former member of the LRG group, namely Tudor Gîrba is a Ph.D. student at SCG under the supervision of Prof. Nierstrasz. Both these two romanian Ph.D. students, working now in the two Swiss teams are planned to be actively involved in this project.

2.2 Account of the state of research in the field

Object-oriented reengineering is an important matter in todays software industry, and it will definitely continue being a vital matter in tomorrow's software industry. The law of software entropy dictates that even when a system starts off in a well-designed state, requirements evolve and customers demand new functionality, frequently in ways the original design did not anticipate. A complete redesign may not be practical, and a system is bound to gradually lose its original clean structure and deform into an unmainteinable, rigid and hard to understand bowl of "object-oriented spaghetti" [WH92] [Cas98], [BMMM98]. In order to postpone the software decay and at least partially save its technical and economical value reengineering is a must.

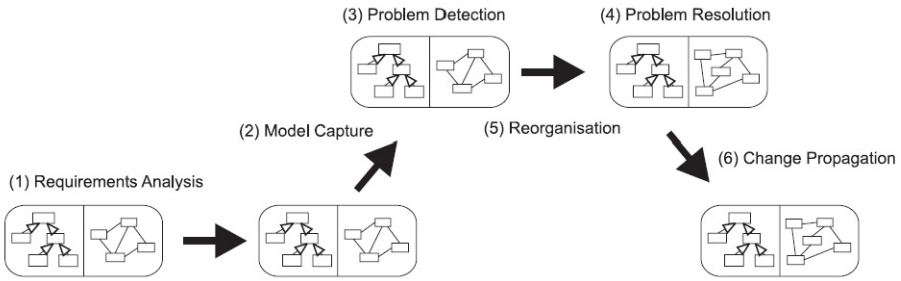

Figure 1. The Reengineering Lifecycle

In this section, following the main phases of the reengineering process, we are going to review the state-of-the-art in the research field of object-oriented software reengineering.

2.2.1 Reverse Engineering

In general, reverse engineering seeks to recover information at a higher level of abstraction, such as design information, from source code and it usually is an important stage in the reengineering process (model capture).

Software representation Source code is the most common representation of software but it has the disadvantage that it is at a low level of abstraction. Thus it can be hardly manipulated in analyzing large projects. Although, in order to extract design information, source code representation is needed and that is why strong tool support for source code representation is a must. Columbus [FBTG02] is a reverse engineering tool that can analyze large C++ software systems and that presents the extracted information in a common specification called the Columbus Schema for C++ [FB02]. There are also many other similar tools such as: Datrix that extracts information from

[5] See http://labmol.di.fc.ul.pt/projects/release/ for more information.
[6] http://w3.umh.ac.be/evol/

C/C++/Java source code files according to the Datrix ASG Model [Inc00], CPPX that produces very precise data about the analysed system because it relies on the output of a real C++ compiler [DMH01].

At a higher level of abstraction, software is represented using meta-models. In informal terms, a meta-model is an attempt to describe the world around us for a particular purpose [Pid02]. In the context of our discussion the "world" is the structure of object-oriented programs, while the "particular purpose" is identified with the goal to support reengineering activities. FAMIX [DTD01] is a language independent metamodel for modelling object-oriented software. The FAMIX core contains the basic elements of object-oriented languages i.e., classes. There are subsequently relations among these entities i.e., an attribute belongs to a class. In addition to the core elements of FAMIX it is possible to use plugins in order to model some language specific aspects.

Historical analysis Metrics have traditionally been used to deal with the problem of analyzing the history of software systems. In 1970ś Lehman used them in order to analyze the evolution of the IBM OS/360 system [LB85]. Lehmann, Perry and Ramil explored the implication of evolution metrics on software maintenance [LPR98] [LPR+97]. They used the number of modules to describe the size of a version and defined evolutionary measurements which take into account differences between consecutive versions.

Other historical approaches analyze the influence of changes in an evolving software system: Burd and Munro analyzed the influence of changes on the maintainability of software systems by defining a set of measurements to quantify the dominance relations which are used to depict the complexity of the calls [BM99]. Gold and Mohan defined a framework to understand the conceptual changes in an evolving program [GM03]. Based on measuring the detected concepts, they could differentiate between different maintenance activities. In terms of change effects, impact analysis approaches attempt to determine, given a point in the source code involved in a modification task, all other possible points in the code that are transitively dependent upon this seed point. Many of these approaches are based on static slicing (e.g. [GL91]) or dynamic slicing (e.g. [AH90]).

Visualisation Information visualization is defined as "the use of computer-supported, interactive, visual representations of abstract data to amplify cognition" [CMS99]. Among the various approaches to support reverse engineering that have been proposed in the literature, graphical representations of software have long been accepted as comprehension aids. That is because visualization is the preferred way of getting acquainted with and navigating large data pools [War00].

In the more specific field of reverse engineering, visualization soon proved to be an effective technique, yielding many tools such as Rigi [MK88] and ShrimpViews [SM95]. Visualization has also proven to be a key technique of research in software evolution, mainly due to the huge amounts of information that need to be processed and understood. Rysselberghe and Demeyer used a simple visualization based on information in version control systems to provide an overview of the evolution of systems [VRD04]. Similar to [JGR99], Wu et al. describe an Evolution Spectrograph [WHH04] that visualizes a historical sequence of software releases.

2.2.2 Problem Detection

Problem detection is another important phase in the reengineering process. Its goal is to detect areas in the legacy system which are characterized by undesired quality properties of the software system, properties which are defined in the requirements phase of the reengineering process. This requires methods and tools to inspect, measure, rank and visualize software structures.

Software metrics are frequently used in problem detection. They quantify simple properties of design structures and can be used to identify abnormal properties of these structures. Software metrics that address the most important characteristics of good object-oriented design like cohesion, coupling and inheritance are defined in [LH93] [CK94] [BK95]. Quantization of the complexity of object-oriented software is also addressed in [HS96]. An important work on object-oriented software metrics is [LK94] where empirical threshold values which dignify abnormal characteristics of design entities are also presented. They were established based on the author's experiences with some C++ and Smalltalk projects. In order to support automatic analysis based on software metrics, tool support for metrics calculation is required. Such tools are presented in [ARK05] [Web05].

Another approach for problem detection can be found in [Ciu99]. The author presents a technique for analyzing legacy code, specifying frequent design problems as queries and locating the occurrences of these problems in a graph-based model derived from the source code.

A special problem that may appear an object-oriented software system is the duplicated code. It is one of the factors that severely complicates the maintenance and evolution of large software systems. In [DRD99] a language independent and lightweight approach to code duplication detection is introduced. A different approach on code duplication detection appears in [Kri01] that is based on finding similar subgraphs in attributed directed graphs. This

approach is used on program dependence graphs and therefore considers not only the syntactic structure of programs but also the data flow within.

2.2.3 System Restructuring

Restructuring refers to transforming a system from one representation to another while remaining at the same abstraction level. At implementation level, this usually means changing the code structure without changing the semantics. In an object-oriented reengineering context, restructuring a system in order to eliminate the problems found in the problem detection phase is expressed in terms of refactorings [FBB+99].

Although in these days there are a number of approaches to identify problems and to remedy them, there is however a conceptual gap between these two stages: there is no appropriate support for the automated mapping of design problems to possible solutions. An important step in order to resolve this problem is described in [Ker05] where it is shown in a step by step manner how to eliminate different design problems by refactoring towards design patterns. The same problem is addressed in [TSG04] where the notion of correction strategy is introduced as reference description that enable a human-assisted tool to plan and perforce all necessary steps for the safe removal of detected design problems.

A different restructuring problem in the context of reengineering is the migration of an application to a new API of the library/framework on which the application depends. Although some tools and ideas [TM03] [RH02] have been proposed to solve the evolution of APIs, most updates are done manually. To better understand the requirements for migration tools, the API changes of some frameworks and libraries are analyses in [DJ05]. The study shows that more than many of these changes are in fact refactorings and suggests that refactoring based migration tools should be used to effectively update applications.

2.3 Account of one's own research in the field

While in the previous subsection we presented a global view on the state-of-the-art in the field of object-oriented software reengineering, in this section we are going to present the group of proposal's contribution to the state-of-the-art.

In the field of problem detection, in [Mar04] a novel mechanism based on metrics is proposed called detection strategy for formulating metrics-based rules that capture deviations from good design principles and heuristics. An extension of this work is presented in [RDGM04] where detection strategies are refined by using historical information of the suspected flawed design entities. It is also shown that using this approach some detection strategies can become more accurate. Another way to improve the precision of detection strategies is presented in [MM05] where concrete examples of structural problems are used in order to tune the threshold values of the corresponding detection strategies.

The problem detection phase of the reengineering process can also be addressed with visualization techniques. In [LD01] [DL05] is presented a new approach on code and design understanding, and more precisely on the understanding of classes based on metrics combined with a new visualization technique called the "class blueprint". The technique may be also applied in the context of problem detection while it allows the identification of some of suspicious class blueprints. This visualization technique is only one instance of what is called polymetric view [LD03]. Such a view is a lightweight software visualization technique enriched with software metrics information. Polymetric views help to understand the structure and detect problems of a software system in the initial phases of a reverse engineering process.

The Proposing teams have also contributions in the field of historical analysis. In [GLD05] some history measurements for characterizing the evolution of classes and class hierarchies are introduced and used to detect different evolution patterns of the evolution of class hierarchies. In [GDL04] the term Yesterdayś Weather is used to depict the retrospective empirical observation that at least one of the classes which were heavily changed in the last period will also be among the most changed classes in the near future. It was computed on two case studies and showed how it can summarize the changes in the history of a system. The approach was used to pinpoint classes in the latest versions which would make good candidates for a first step in reverse engineering. One of the most difficult aspects of historical analysis is to reduce the amount of data one has to deal with during the analysis. A simple visualization approach to resolve this problem was introduced in [Lan01].

Due to the large amount of legacy code, all the aforementioned analysis techniques would be useless without strong tool support. That's why the members of the proposing team has been paying special attention to this aspect of their research.

Code duplication detection is important in the context of proposing team's activity. DuDe [Wet04] is a tool that uses textual comparison at the level of lines of code in order to detect portions of duplicated code. It has a powerful detection engine that can also cover some fine changes to the duplicated code and that can also detect chains of duplicated code [WM05].

Mc'C [Mih04] is a model capture tool that extracts detailed design information from C++ software systems. The extracted information is organized in ASCII tables according to the MEMORIA meta-model. MEMORIA [Rat04] is a unified meta-model (and also an implementation of this meta-model) whose purpose is to facilitate the analysis of the software systems written in Java and C++. Some of the most important goals of MEMORIA are: (1) to offer a consistent model even in the presence of incomplete code or missing libraries; (2) to allow the analysis of large systems and (3) to ease the navigation within a system. HisMo history meta-model [GFD04] is an extention of the FAMIX meta-model [DTD01] that enables history analysis on large industrial object-oriented software systems. It is implemented in the Van history analysis tool.

CodeCrawler [LDGP05] is a language-independent software visualization tool written in Smalltalk. It supports reverse engineering through the combination of metrics and software visualization. Its power and flexibility, based on simplicity and scalability, has been repeatedly proven in several large scale industrial case studies.

All these reengineering tools are integrated in MOOSE reengineering platform [DGLD05] and iPlasma reenginerring platform [MMM+05].

3 Research Plan

3.1 Objectives

Reengineering is a complex task which requires multiple techniques and models. More than that, the past few years have shown that to come up with a working solution for reengineering industrial systems, single persons and even single groups are not enough to tackle all problems, such as parsing, modelling, data mining, visualization, etc.

Moreover, as research groups tend to specialize in one particular field, e.g., data mining or parsing, they must rely on external, often commercial, tools to be able to perform a full analysis.

We want to close this gap, as the participating research groups have all built powerful research prototypes in complementary areas: the tools should collaborate with each other at an infrastructural level. Not only that, we want the tools to collaborate the tools in real-time by enabling the tools as web services which can be freely accessed by the other tools.

We want to build a distributed reengineering environment which is able to make all these techniques and models complement each other. Then we want to use this environment to integrate different reengineering techniques to obtain complex reengineering techniques:

- combine problem detection with visualization to get an overview of how problems are distributed over a software system
- combine evolution analysis with problem detection to understand the evolution of quality
- combine code duplication detection with semantic analysis to detect high level clones in the systems

3.2 Scientific and technical description

We foresee 3 tracks structured as work packages (WP):

1. Creating a Distributed Reengineering Environment (WP1)
2. Enabling Complex (Distributed) Reengineering Activities (WP2)
3. Performing Large-Scale Distributed Reengineering Experiments (WP3)

3.2.1 WP1: Creating a Distributed Reengineering Environment

Introduction. The goal of this work package is the building of a prototype of the NOREX environment. The groups participating in this project are internationally known proactive tool builders:

- LRG builds iPlasma[MMM+05], a reengineering environment written in Java, and further tools for parsing and model extraction.
- SCG builds Moose[DGLD05], a reengineering environment, and several other tools built on top of it;
- USI also participates in developing Moose and it builds CodeCrawler[LD05], a general purpose visualization environment. CodeCrawler is built on top of the Moose environment and makes use of the Moose meta-meta-model to let the user specify the visualization.

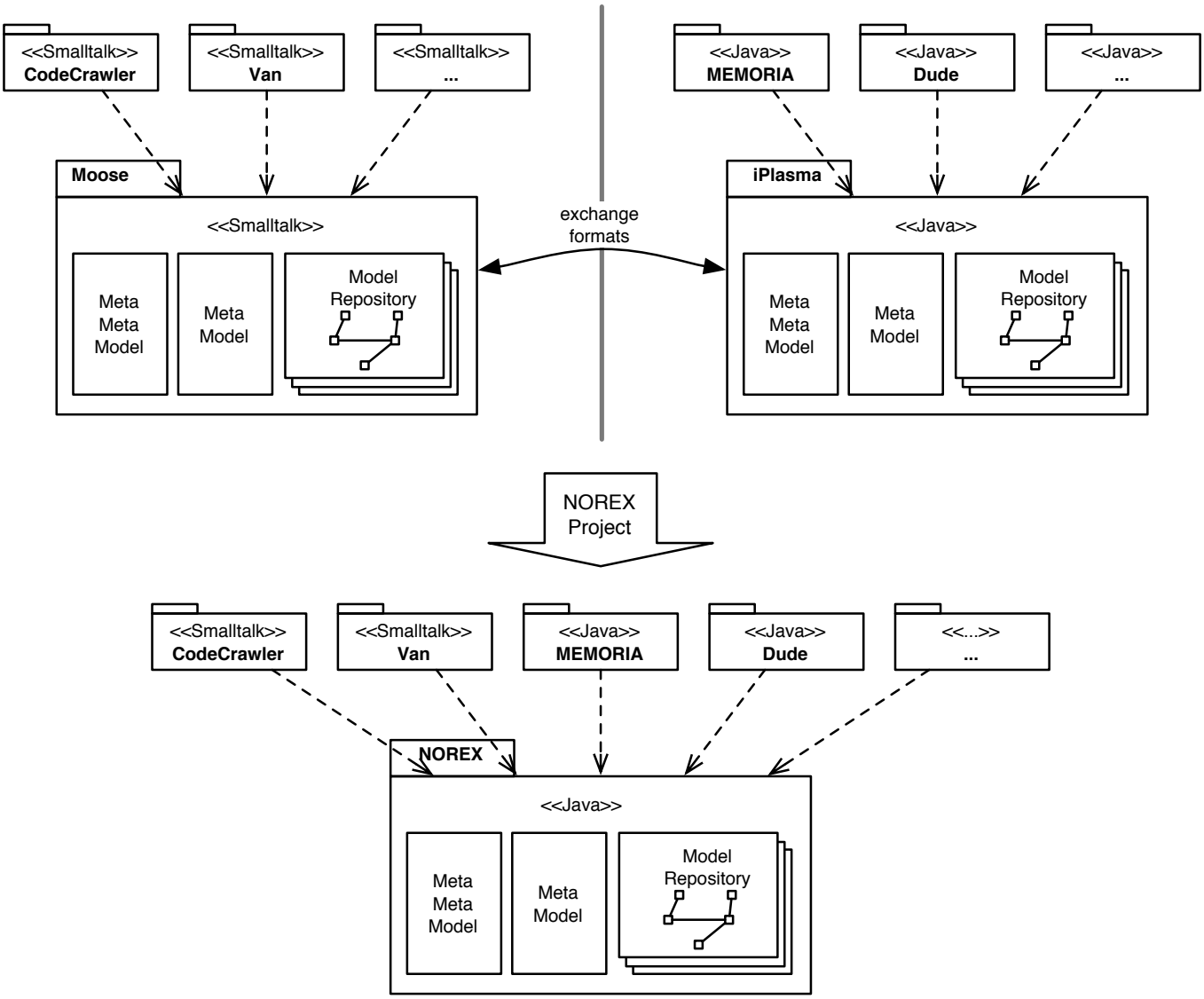

Figure 2. Moose and iPlasma are two reengineering environments, one written in Smalltalk and one in Java. The two communicate through exchange formats at the model level. NOREX will be a distributed environment aiming to integrate reengineering tools built in different languages. It offers a common meta-meta-model, an extensible meta-model and a model repository for storing and manipulating the actual models.

Plan. Figure 2 shows schematically the idea behind the NOREX environment. The first task of the environment is to ensure that different tools can understand each other by understanding the semantics of their models [BG01]. The semantics of a model are expressed in its meta-model [Sei03]. Moreover, to understand the meta-model one must understand the description of the meta-model expressed in a common meta-meta-model. Thus, the first step in the implementation is the meta-meta-model. As a reference we will take the implementation of Moose and iPlasma which already have implemented similar meta-meta-models.

Furthermore, the tools need to be implemented in different languages. A distributed environment needs to ensure that the different tools are able to communicate with each other.

We aim to allow for different groups to build tools in a loose fashion, that is without imposing hardcoded dependencies between these tools. Thus we will continue developing our tools in separate ways but aim to implement the whole NOREX inter-communications framework between the tools. In this case too, we will use as reference the implementation of Moose and iPlasma which already have implemented similar mechanisms.

A good candidate for implementation is to use the Eclipse Rich Client Platform[7] because it offers the necessary infrastructure for plugin-architecture and for visual integration of the tools. As support for the communication protocol we plan to use the WebService technology[8].

Deliverables. As major deliverables we foresee:

- (D 1.1) A specification and implementation of the common meta-meta-model.
- (D 1.2) A specification and implementation of the communication protocol.
- (D 1.3) Design and implementation of a distributed registration mechanism for reengineering tools.

3.2.2 WP2: Enabling Complex Reengineering Activities

Introduction. The goal of this workpackage is to exercise the capabilities of the NOREX environment and to provide feedback for further implementation.

Plan. We want to explore the possibilities open by the NOREX environment to build complex analyses by combining tools built by the participating groups. Examples of planned analyses are: (1) combining problem detection with visualization to get an overview of how problems are distributed over a system, (2) combining evolution analysis with problem detection to understand the evolution of quality, and (3) combining code duplication with semantic analysis to detect high level clones.

To exercise the reengineering capabilities, we plan to build a tool for performing model-based transformations. The idea behind this tool would be to exercise different reengineering scenarios at the level of the model, without affecting the source code, thus allowing the engineer to test the consequences of the reengineering effort. This tool would then bridge between source code analysis and the actual reengineering.

During this phase we plan to also create a central website for putting together the experience gained by this project.

Deliverables. As major deliverables we foresee:

- (D 2.1) A catalogue of complex (high-level) detection techniques for design flaws based on a combination of metrics, visualization, history analysis and semantic analysis.
- (D 2.2) A catalogue of complex reengineering activities (workflows) that would describe the entire process of identification and correction of design specific design problems.
- (D 2.3) A tool prototype for simulating refactorings that would operate model-based transformations of a system (rather than code transformations as usual refactorings browsers). This would allow us assess the impact of complex reengineering activities.
- (D 2.4) A central reengineering website which links to all the available reengineering infrastructures.

3.2.3 WP3: Performing Large-Scale Reengineering Experiments

Introduction. The aim is to validate the project goals of the first two WPs by performing large-scale distributed reengineering experiments. This is reached by combining the unique experience obtained by the project partners both at a human as well as at an infrastructural level over the web, made possible by the deliverables of WP1 and WP2.

[7] http://www.eclipse.org/rcp/
[8] http://java.sun.com/webservices/index.jsp

Plan. Typically experiments are developed to primarily show features; yet, we will apply techniques to enable learning as integrative part of this research project. Consequently, this WP will convincingly show the applicability and scalability of our developed technology and methodology. We will take up to three representative case studies (in terms of size and/or developers involved) and use open source projects implemented in various programming languages and with different business domains (*e.g.*, Mozilla, Eclipse, NetBSD, JBoss, or Jun) as the main applications.

The idea then is to exercise some of the the complex reengineering activities defined as workflows in WP2 (see deliverable D 2.2) in order to provide insights into the design problems of the case studies and on their potential restructuring. We plan to use the NOREX environment to automatize in a unified and transparent manner all the phases of the reengineering lifecycle (see Fig 1, from parsing and model extraction up to visualizations and elaboration of restructuring plans.

By replacing the isolated usage of the various reengineering techniques and tools with an integrated approach, we expect to improve the ability of those interested in our work to understand the broader value of such reengineering artifacts and to be able to understand its usability in everyday software engineering. By creating a reengineering "microworld", we will provide our technology to allow others to reflect in the light of their own experiences.

Additionally, we anticipate that people within this project and in research networks such as RELEASE will be able to integrate or assimilate the technology. Our technology will enable others to either build on top of our analyses results or natively apply their techniques and tools with the support of our evolution control platform. We expect that others will also benefit from exploiting our diverse meta-models and, therefore, contribute to the maturity of the research community on reengineering. Further, this workpackage will allow a broader dissemination of the results of our project work and improve its leveragability across particular communities such as program comprehension, reverse engineering, source code analysis, or software maintenance. The experiments will be set up in the second phase of the project, where all tools and techniques must be validated.

Deliverables. As major deliverables we foresee:

- (D 3.1) A setup of 2-3 case-studies in order to demonstrate the feasibility of using the distributed reengineering environment (see WP1) to address complex reengineering activities (see WP2).
- (D 3.2) A catalogue of frequent design problems and how to address them in terms of reengineering activities.
- (D 3.3) A methodology for the effective usage of the various types of analyses defined and implemented within the distributed reengineering environment defined in WP2.

3.3 Planning, co-ordination and management

Each workpackage will be performed by all three project partners, while co-ordination and management will mainly be dealt with by the USI partner. The work packages are overlapping as depicted in Figure 3 where we see a sketched time plan of the project. We will now take a more closer look at the three workpackages described in Section 3.2 from

Workpackage	Task	Partner	1st Year				2nd Year			
			1-3	4-6	7-9	10-12	13-15	16-18	19-21	22-24
WP1	1	1 + 2 + 3		D1.1						
	2	1 + 2 + 3			D1.2					
	3	2 + 3				D1.3				
WP2	4	1 + 2 + 3					D2.1			
	5	1 + 3								D2.2
	6	1+ 2+3						D2.3		
	7	1 + 2						D2.4		
WP3	7	1 + 2 + 3								D3.1
	8	1 + 2 + 3								D3.2
	9	1 + 2 + 3								D3.3

Figure 3. The time plan for the NOREX project

the perspective of planning coordination and management.

WP1: Creating a Distributed Reengineering Environment. This first workpackage is planned to start when the project begins (01.09.2005) and should be finished after 12 months, with the accomplishment of the three deliverables described in Section 3.2.1. During the first month of the project we plan a meeting will all three partner teams

focused on the specification and prototype implementation of the common meta-meta-model (D1.1) and communication protocol (D1.2). These first two deliverables should need 6 months of work and are planned to be finished at the end of the 9th months of the project. All three partner teams will be involved. The design and implementation of the actual framework (D1.3) will start after all the conceptual problems related to D1.1 and D1.2 are clarified, and it is planned to take 9 months to be finished. The main actors involved in this task are 2 junior scientists from the LRG group in Timişoara in collaboration with 1 junior scientist from SCG Bern.

WP2: Enabling Complex Reengineering Activities The effort on this second workpackage (Section 3.2.2) is planned to start in the second part of the first project year (month 7) by a joint work on a catalogue containing the useful combinations of individual reengineering techniques (D2.1). This task will involve basically all the Ph.D. students from the three teams, under the supervision of their team leaders. At this point another general meeting would be needed to settle down a common approach. This task is planned to last for 9 months but the heavier workload will be in the first 3 months (7-9) on the conceptual side, and in the last 3 months (13-15) on the experimental side using the NOREX environment. The implementation of the refactorings simulator (D2.3) is planned to start in month 10, last 9 month and be carried out by the LRG and USI team. When D2.3 is accomplished, the central reengineering website (D2.4) will be set up by the SCG and USI team. Eventually, all the three teams will join again their efforts to define a suite of relevant complex reengineering activities by using the NOREX environment. This task will last till the end of the project (9 months).

WP3: Performing Large-Scale Reengineering Experiments The three teams will define together the experimental setup for the large-scale reengineering experiments to be conducted. The actual experiments are planned to be done in parallel with the development of the NOREX environment and of the tools built on top. This workpackage is planned to start in month 10, after the development of different prototype tools supporting the catalogue of complex reengineering activities (D 2.1), and last until the end of the project. The conclusions of these experiments will be continuously distilled by the three teams in form of the D3.2 and D3.3 deliverables (see Section 3.2.3, which are also expected to be finished at the end of the project.

3.4 Dissemination/exploitation of results, follow-up

We plan to publish our results in top-ranked conferences (such as WCRE, ICSM, OOPSLA, ECOOP, IWPSE, IWPC, CSMR, ICSE, ESEC-FSE) and journals (such as TSE, TOSEM, and JSME). Further we will disseminate our results along the already established research links in the RELEASE network and the ERCIM WG on Software Evolution.

We also plan to offer automated reengineering support to several large software companies in Timişoara that face large-scale legacy systems, as a result of outsourcing. The LOOSE research group has collaborated in the past with two of these two companies (*i.e.*, Alcatel and Siemens Automotive) and they found out that reengineering activities are especially relevant for them (in the context of outsourcing) because of two reasons: (1) the need of understanding the outsourced system; (2)the need of restructuring in order to assure its successful evolution.

The partners in Bern and Lugano also have extensive past (Nokia, Bedag AG, Mediagenix, BMW) and present (Siemens, Harmann-Becker) consulting project experience and plan to offer the functionalities realized through this project to them, as well as looking for new industrial partners with large reengineering projects.

4 Intended impacts of the planned work

4.1 Impact on strengthening of capacities

This project will have a threefold impact on strengthening and improving the capacities of each research group:

1. *Enlarging horizons for young scientists.* Almost all members of the three research teams involved in this project are young scientists (under 35 years), many of them at the beginning of their Ph.D. studies. The central idea of this project is to stimulate an enlargement of horizon for all the participants, by allowing them to interact (via the reengineering network) with related achievements of their colleagues from the other groups. We are confidently looking forward to the synergetic effect that could result from this type of idea exchanges, that will not remain at the abstract level of ideas, but will easily made experimentable by means of the distributed reengineering environment.

2. *Overcoming the isolation of Romanian scientists.* One of the major problem with scientists from countries like Romania is a high level of isolation, especially among young scientists. This lack of feedback and interaction with

other fellows working on related subjects makes them in many cases to abandon promising research projects. The current project will stimulate Ph.D. students from the research group in Timişoara to keep working on their ideas, as these ideas will become accessible through the proposed network, and will also be disseminated at an international level (*i.e.*, renowned conferences) increasing thus their visibility.

3. *Enhancement of software engineering teaching.* Over the last years, all three applicants (*i.e.*, Prof. Lanza, Prof.Nierstrasz and Prof.Marinescu) have held lectures on object-oriented reengineering in Bern and resp. in Timişoara (at the "Politehnica" University). The implementation of the distributed reengineering environment proposed by this project, would significantly enhance the concrete, "hands-on" experience of students, by means of students' projects that could address a larger variety of issues (*i.e.*, use all types of analyses and tools developed by all three participating teams).

4.2 Impact on national and international networking

This project will maintain and consolidate the already long-lasting collaboration links between the two research groups in Switzerland and the LOOSE Research Group in Timişoara (Romania). The main reason for this is given by the intrinsic nature of the project proposal, as the proposed reengineering network is an "open-ended", continuously growing technical and scientific challenge. Due to its significant expertise in reengineering and due to the value of the produced artifacts, the LOOSE Research Group group will definitely play a key role in the network, even after this project will be finished.

Additionally, there is an aspect of the proposed project that we expect to produce a significant positive impact. In the future, even before the end of the project, this network of reengineering expertise could be extended with further members, *i.e.*, other European research groups belonging to the reengineering or software evolution communities (*e.g.*, partner groups from the ESF/RELEASE network or the ERCIM EVOL working group on software evolution) and even further interested groups from Eastern European countries.

4.3 Impact on economy and/or society of partner countries

We strongly believe that the impact of this project goes beyond an academic interest and it will have a significant impact also on the software industry in Timişoara, an industry in a continuous evolution, growing through its maturity. The past and current industrial consulting experiences confirm the great need of tools and methodologies to reengineer large scale software systems.

Finally, outsourcing became in the last decade more and more a significant phenomenon in Eastern European countries, especially for the Romanian software industry. Several large companies (*e.g.*, Alcatel and Siemens Automotive) have outsourced their software products to Timişoara. In this context, we could offer automated reengineering support to Romanian software companies that face large-scale legacy systems, as a result of outsourcing. This is a key aspect because the outsourcing process is intrinsically related to reengineering on one hand due to the need of understanding the outsourced system and on the other hand due to the need of restructuring in order to assure its successful evolution. This initiative is feasible due to the previous research collaborations of Prof. Marinescu with the two aforementioned software companies.

By this initiative we would not only contribute to better software products, but also we believe that it will enhance the professional qualifications of the engineers that would take advantage of it, as they will get acquainted with state-of-the-art knowledge on best practices related to object-oriented design and methodologies to improve their design and implementation style.

References

[ADN05] Gabriela Arévalo, Stéphane Ducasse, and Oscar Nierstrasz. Discovering Unanticipated Dependency Schemas in Class Hierarchies. In *Proceedings of CSMR '05 (9th European Conference on Software Maintenance and Reengineering)*. IEEE Computer Society Press, March 2005.

[AH90] H. Agrawal and J.R. Horgan. Dynamic program slicing. In *Proceedings of the Conference on Programming Language Design and Implementation*, pages 246–256, 1990.

[ARK05] J. Alghamdi, R. Rufai, and S. Khan. OOMeter: A Software Quality Assurance Tool. In *ICSM*, 2005.

[BG01] Jean Bézivin and Olivier Gerbé. Towards a precise definition of the omg/mda framework. In *Proceedings of Automated Software Engineering (ASE 2001)*, pages 273–282. IEEE Computer Society, 2001.

[BK95] J.M. Bieman and B.K. Kang. Cohesion and reuse in an object-oriented system. In *Proceedings ACM Symposium on Software Reusability*, April 1995.

[BM99] Elizabeth Burd and Malcolm Munro. An initial approach towards measuring and characterizing software evolution. In *Proceedings of the Working Conference on Reverse Engineering, WCRE '99*, pages 168–174, 1999.

[BMMM98] William J. Brown, Raphael C. Malveau, Hays W. McCormick, III, and Thomas J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley Press, 1998.

[Cas98] Eduardo Casais. Re-engineering object-oriented legacy systems. *Journal of Object-Oriented Programming*, 10(8):45–52, January 1998.

[Ciu99] Oliver Ciupke. Automatic detection of design problems in object-oriented reengineering. In *Proceedings of TOOLS 30 (USA)*, pages 18–32, 1999.

[CK94] Shyam R. Chidamber and Chris F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, June 1994.

[CMS99] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman, editors. *Readings in Information Visualization — Using Vision to Think*. Morgan Kaufmann, 1999.

[DD99] Stéphane Ducasse and Serge Demeyer, editors. *The FAMOOS Object-Oriented Reengineering Handbook*. University of Bern, October 1999.

[DDN02] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. *Object-Oriented Reengineering Patterns*. Morgan Kaufmann, 2002.

[DGLD05] Stéphane Ducasse, Tudor Gîrba, Michele Lanza, and Serge Demeyer. Moose: a Collaborative and Extensible Reengineering Environment. In *Tools for Software Maintenance and Reengineering*, RCOST / Software Technology Series, pages 55 – 71. Franco Angeli, 2005.

[DJ05] Danny Dig and Ralph Johnson. The role of refactorings in api evolution. In *Submitted for publication at ICSM*, 2005.

[DL05] Stéphane Ducasse and Michele Lanza. The class blueprint: Visually supporting the understanding of classes. *IEEE Transactions on Software Engineering*, 2005.

[DMH01] T. R. Dean, A. J. Malton, and R. Holt. Union schemas as a basis for a c++ extractor. In *WCRE*, pages 59–67, 2001.

[DRD99] Stéphane Ducasse, Matthias Rieger, and Serge Demeyer. A language independent approach for detecting duplicated code. In Hongji Yang and Lee White, editors, *Proceedings ICSM '99 (International Conference on Software Maintenance)*, pages 109–118. IEEE, September 1999.

[DTD01] Serge Demeyer, Sander Tichelaar, and Stéphane Ducasse. FAMIX 2.1 — The FAMOOS Information Exchange Model. Technical report, University of Bern, 2001.

[FB02] Rudolf Ferenc and Árpád Beszédes. Data exchange with the columbus schema for c++. In *CSMR*, pages 59–66, 2002.

[FBB+99] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Addison Wesley, 1999.

[FBTG02] Rudolf Ferenc, Árpád Beszédes, Mikko Tarkiainen, and Tibor Gyimóthy. Columbus - reverse engineering tool and schema for c++. In *ICSM*, pages 172–181, 2002.

[GD05] Orla Greevy and Stéphane Ducasse. Correlating features and code using a compact two-sided trace analysis approach. In *Proceedings of CSMR 2005 (9th European Conference on Software Maintenance and Reengineering*. IEEE Computer Society Press, 2005.

[GDL04] Tudor Gîrba, Stéphane Ducasse, and Michele Lanza. Yesterday's Weather: Guiding Early Reverse Engineering Efforts by Summarizing the Evolution of Changes. In *Proceedings of ICSM '04 (International Conference on Software Maintenance)*, pages 40–49. IEEE Computer Society Press, 2004.

[GDMR04] Tudor Gîrba, Stéphane Ducasse, Radu Marinescu, and Daniel Raţiu. Identifying entities that change together. In *Ninth IEEE Workshop on Empirical Studies of Software Maintenance*, 2004.

[GFD04] Tudor Gîrba, Jean-Marie Favre, and Stéphane Ducasse. Using meta-model transformation to model software evolution. In *2nd International Workshop on Meta-Models and Schemas for Reverse Engineering (ATEM 2004)*, 2004.

[GL91] Keith Brian Gallagher and James R. Lyle. Using Program Slicing in Software Maintenance. *Transactions on Software Engineering*, 17(18):751–761, August 1991.

[GLD05] Tudor Gîrba, Michele Lanza, and Stéphane Ducasse. Characterizing the evolution of class hierarchies. In *Proceedings of European Conference on Software Maintenance (CSMR 2005)*, 2005.

[GM03] Nicolas Gold and Andrew Mohan. A framework for understanding conceptual changes in evolving source code. In *Proceedings of International Conference on Software Maintenance 2003 (ICSM 2003)*, pages 432–439, September 2003.

[HS96] Brian Henderson-Sellers. *Object-Oriented Metrics: Measures of Complexity.* Prentice-Hall, 1996.

[Inc00] Bell Canada Inc. Datrix Abstract Semantic Graph Reference Manual (version 1.4), 2000.

[JGR99] Mehdi Jazayeri, Harald Gall, and Claudio Riva. Visualizing Software Release Histories: The Use of Color and Third Dimension. In *Proceedings of ICSM '99 (International Conference on Software Maintenance)*, pages 99–108. IEEE Computer Society Press, 1999.

[Ker05] Joshua Kerievsky. *Refactoring To Patterns.* Addison-Wesley, 2005.

[Kri01] Jens Krinke. Identifying similar code with program dependence graphs. In *Proceedings Eigth Working Conference on Reverse Engineering (WCRE'01)*, pages 301–309. IEEE Computer Society, October 2001.

[Lan01] Michele Lanza. The evolution matrix: Recovering software evolution using software visualization techniques. In *Proceedings of IWPSE 2001 (International Workshop on Principles of Software Evolution)*, pages 37–42, 2001.

[Lan03] Michele Lanza. *Object-Oriented Reverse Engineering — Coarse-grained, Fine-grained, and Evolutionary Software Visualization.* PhD thesis, University of Berne, May 2003.

[LB85] Manny M. Lehman and Les Belady. *Program Evolution – Processes of Software Change.* London Academic Press, 1985.

[LD01] Michele Lanza and Stéphane Ducasse. A Categorization of Classes based on the Visualization of their Internal Structure: the Class Blueprint. In *Proceedings of OOPSLA '01 (International Conference on Object-Oriented Programming Systems, Languages and Applications)*, pages 300–311. ACM Press, 2001.

[LD03] Michele Lanza and Stéphane Ducasse. Polymetric views — a lightweight visual approach to reverse engineering. *IEEE Transactions on Software Engineering*, 29(9):782–795, September 2003.

[LD05] Michele Lanza and Stéphane Ducasse. Codecrawler - an extensible and language independent 2d and 3d software visualization tool. In *Tools for Software Maintenance and Reengineering*, RCOST / Software Technology Series, pages 74 – 94. Franco Angeli, 2005.

[LDGP05] Michele Lanza, Stéphane Ducasse, Harald Gall, and Martin Pinzger. Codecrawler - an information visualization tool for program comprehension. In *Proceedings of ICSE 2005 (27th IEEE International Conference on Automated Software Engineering)*, pages xxx – xxx. ACM Press, 2005.

[LH93] W. Li and S. Henry. Maintenance metrics for the object oriented paradigm. *Proceedings of the First International Software Metrics Symposium.*, pages 52–60, May 1993.

[LK94] Mark Lorenz and Jeff Kidd. *Object-Oriented Software Metrics: A Practical Guide.* Prentice-Hall, 1994.

[LPR+97] M.M. Lehman, D. E. Perry, J. F. Ramil, W. M. Turski, and P. D. Wernick. Metrics and laws of software evolution - the nineties view. In *Metrics '97, IEEE*, pages 20 – 32, 1997.

[LPR98] M. M. Lehman, Dewayne E. Perry, and Juan F. Ramil. Implications of evolution metrics on software maintenance. In *Proceedings of the International Conference on Software Maintenance (ICSM 1998)*, pages 208–217, 1998.

[Mar04] Radu Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In *Proceedings of ICSM '04 (International Conference on Software Maintenance)*, pages 350–359. IEEE Computer Society Press, 2004.

[Mih04] Petru Florin Mihancea. The Extraction of Detailed Design Information from C++ Software Systems. Master Thesis, "Politehnica" University of Timisoara, 2004.

[MK88] H. A. Müller and K. Klashinsky. Rigi – a system for programming-in-the-large. In *ICSE '88: Proceedings of the 10th international conference on Software engineering*, pages 80–86. IEEE Computer Society Press, 1988.

[MM05] Petru Florin Mihancea and Radu Marinescu. Towards the optimization of automatic detection of design flaws in object-oriented software systems. In *Proceedings of CSMR 2004 (European Conference on Software Maintenance and Reengineering)*, pages 92–101, 2005.

[MMM+05] Cristina Marinescu, Radu Marinescu, Petru Florin Mihancea, Daniel Raţiu, and Richard Wettel. iPlasma: An Integrated Platform for Quality Assessment of Object-Oriented Design. In *Submitted for publication at ICSM (Tool Demonstration)*, 2005.

[Pid02] W. Pidcock. What is Meta-Modelling, 2002.

[Rat04] Daniel Ratiu. Memoria : A Unified Meta-Model for Java and C++. Master Thesis, "Politehnica" University of Timisoara, 2004.

[RDGM04] Daniel Raţiu, Stéphane Ducasse, Tudor Gîrba, and Radu Marinescu. Using history information to improve design flaws detection. In *Proceedings of CSMR 2004 (European Conference on Software Maintenance and Reengineering)*, pages 223–232, 2004.

[RH02] S. Roock and A. Havenstein. Refactoring tags for automatic refactoring of framework. In *in Proceedings of Extreme Programming Conference*, 2002.

[Sei03] Ed Seidewitz. What models mean. *IEEE Software*, 20:26–32, September 2003.

[SM95] Margaret-Anne D. Storey and Hausi A. Müller. Manipulating and Documenting Software Structures using SHriMP Views. In *Proceedings of ICSM '95 (International Conference on Software Maintenance)*, pages 275 – 284. IEEE Computer Society Press, 1995.

[TM03] T. Tourwe and T. Mens. Automated support for framework based software. In *ICSM*, pages 148–157, 2003.

[TSG04] Adrian Trifu, Olaf Seng, and Thomas Genssler. Automated Design Flaw Correction in Object-Oriented Systems. In *CSMR*, pages 174–183, 2004.

[VRD04] Filip Van Rysselberghe and Serge Demeyer. Studying software evolution information by visualizing the change history. In *Proceedings of The 20th IEEE International Conference on Software Maintenance (ICSM 2004)*, 2004. to appear.

[War00] Colin Ware. *Information Visualization.* Morgan Kaufmann, 2000.

[Web05] SDMetrics Website. Sdmetrics, 2005.

[Wet04] Richard Wettel. Automated Detection of Code Duplication Clusters. Diploma Thesis, "Politehnica" University of Timisoara, 2004.

[WH92] Norman Wilde and Ross Huitt. Maintenance Support for Object-Oriented Programs. *IEEE Transactions on Software Engineering*, SE-18(12):1038–1044, December 1992.

[WHH04] Jingwei Wu, Richard Holt, and Ahmed Hassan. Exploring software evolution using spectrographs. In *Proceedings of 11th Working Conference on Reverse Engineering (WCRE 2004)*, pages 80–89. IEEE Computer Society Press, November 2004.

[WM05] Richard Wettel and Radu Marinescu. Archeology of Code Duplication: Recovering Duplication Chains From Small Duplication Fragments. In *Submitted for publication at ICSM*, 2005.