Editor: **Christof Ebert**
Vector Consulting Services
christof.ebert@vector.com

Reengineering Technologies

Ricardo Pérez-Castillo, Ignacio García-Rodriguez de Guzmán, Mario Piattini, and Christof Ebert

Software systems must continually evolve to meet ever-changing needs. However, such systems often become legacy systems as a consequence of uncontrolled maintenance combined with obsolete technology. To control maintenance costs and preserve complex embedded business rules, companies must evolve their legacy systems. This article introduces technologies for software reengineering. I look forward to hearing from both readers and prospective authors about this column and the technologies you want to know more about. —*Christof Ebert*

REENGINEERING IS THE most widely used approach to support evolutionary maintenance. It examines and alters a legacy system to reconstitute it in a new form.[1]

> Reengineering consists of three stages: reverse engineering, restructuring, and forward engineering.

Reengingeering Stages

Reengineering consists of three stages: reverse engineering, restructuring, and forward engineering.

Reverse Engineering

Maintainers use reverse engineering to obtain one or more abstract representations of the legacy systems—for example, a set of class diagrams representing the system design—that identify the system's components and their interrelationships at a higher abstraction level.[2] Reverse engineering techniques fit into two categories: static and dynamic analysis.[3]

Static analysis syntactically analyzes the source code's structure to highlight weaknesses, complexities, and opportunities for improvement. Maintainers can also apply static analysis to database scripts to determine the database design.

Dynamic analysis examines legacy systems at runtime. It instruments source code to enable the code to register traces during execution. It then analyzes those traces to retrieve abstract representations of the legacy system. It can retrieve particular information—for example, performance data and values of data variables—that's known only during system execution.

Restructuring

Restructuring transforms the abstract representations that reverse engineering provides into other representations at the same abstraction level. It aims to improve certain properties such as the representation's structure or quality as well as to introduce new business requirements. For example, maintainers can use restructuring to apply design patterns to maintain the system design. Restructuring techniques include *refactoring* to restructure source code or a database schemas by preserving its external behavior, *clone detection* to detect duplicate fragments of source code, and *dead-code detection* for removing inoperative code.

Forward Engineering

Maintainers use the restructured representations of legacy systems to generate physical implementations of the target system at a low abstraction level. Forward engineering involves using generative and transformational techniques to automatically obtain source code according to new platforms or programming languages.

0740-7459/11/$26.00 © 2011 IEEE

TABLE 1

Software reengineering tools.

Tool	Type	Reengineering stages supported	Techniques supported	Model-driven	Input artifacts	Output artifacts	Usefulness for maintainers	Pricing (US$)
Amelio	Commercial	All	Static analysis, refactoring, and automatic code generation	No	Source code and relational databases	Data-flow and process-flow graphs, along with source code	Reliable planning, calculation, and accomplishment of modernization projects	$0.42–$2.10 per LOC, depending on system complexity
Avalanche	Academic	Reverse engineering	Dynamic analysis	No	Binary code	Control and data-flow graphs and metrics	Program comprehension and detection of bugs and vulnerabilities	Free (open source code)
Blue Age	Commercial	All	Static analysis, refactoring, and automatic code generation	Yes	Source code (various languages)	System design diagrams and source code (different languages)	Migrates and transforms applications	n/a
CloneDR	Commercial	Reverse engineering and restructuring	Static analysis and clone detection	No	Source code (various languages)	Removing clone code	Maintainability improvement by removing duplicate code	$0.005–$0.03 per LOC
DB-MAIN	Academic	All	Static analysis, refactoring, and code generation	No	Relational and non-relational databases and database scripts	Database design, database scripts, and source code	Migration, integration, and federation of legacy databases; impact analysis; and data wrapper design and generation	Free
Microfocus Modernization Workbench	Commercial	Reverse engineering and restructuring	Static analysis	No	Application portfolio	Business knowledge	Locates where changes should be made to an application without disrupting stability	n/a

Model-Driven Reengineering

Software engineers have applied model-driven development (MDD) technologies to software reengineering to deal with standardization and automation challenges. In 2007, the Object Management Group (OMG) launched the Architecture-Driven Modernization (ADM) initiative to standardize model-driven reengineering. ADM advocates following MDD principles on reengineering projects, such as treating all the artifacts involved as models and providing model transformation among them. These let maintainers carry out more automatic and reusable reengineering projects because models are based on standard metamodels and model transformations are implemented using purpose-specific languages.

Maintainers adapt reengineering techniques to manage models using model refactoring rather than traditional refactoring. They also use static and dynamic analyses to generate models with the information retrieved according to metamodels, and so on.

The ADM initiative has also created the Knowledge Discovery Metamodel (KDM), which has been recently adopted

TABLE 1 (CONT'D)

Software reengineering tools.

Tool	Type	Reengineering stages supported	Techniques supported	Model-driven	Input artifacts	Output artifacts	Usefulness for maintainers	Pricing (US$)
MoDisco	Academic	Reverse engineering and restructuring	Static analysis and refactoring	Yes	Source code, databases and other artifacts	Models representing artifacts	A generic and extensible metamodel-driven approach to model discovery	Free (Eclipse plug-in)
Moose	Academic	Reverse engineering and restructuring	Static analysis, refactoring, and clone detection	Yes	Source code and user interfaces	System design diagrams	Allows understanding of legacy systems and evaluating their maintainability	Free (open source code)
Obeo Agility	Commercial	All	Static analysis, refactoring, and automatic code generation	Yes	Source code (various languages)	System design diagrams and source code (different languages)	Migrates and transforms applications	n/a
Resource Miner	Commercial	Reverse engineering and restructuring	Static analysis and refactoring	No	Source code (various languages)	Code inventory, requirements, and quality audits	Software asset management and application measure	$999 single license per year
Visual Paradigm	An integrated development environment supporting reengineering	Reverse engineering and forward engineering	Static analysis and automatic code generation	No but based on the Unified Modeling Language (UML)	Java, C++ java code or UML class diagrams	UML class diagrams, Java, or C++ code	Round-trip between Java or C++ and system design based on UML class diagrams	$699 single license and $67,000 corporate license (free community edition)

as the ISO/IEC 19506 standard.[4] KDM is the cornerstone of the ADM initiative and provides a common format for the interchange of information between reengineering tools. KDM facilitates the representation of information retrieved by reverse engineering from different legacy software artifacts and system viewpoints (for example, source code, database, and user interfaces). KDM is comparable to the UML standard. Whereas UML is used to generate new code in a top-down manner, a reengineering process involving KDM starts from the existing code and builds higher-level models in a bottom-up manner.

KDM-based model-driven reengineering alters the construction and use of reengineering tools. Traditional reengineering tools have been built as silos from which each tool recovers and analyzes a particular proprietary content in a single silo—for example, one tool would be used for the source code (such as Cobol), and another would be used for the legacy database (such as CICS/DB2; Customer Information Control System/Database 2). So, two proprietary and independent models exist at the end of the process—a Java code model and a DB2 database schema model—which should also be analyzed independently.

ADM encourages maintainers to build reengineering tools in a KDM ecosystem. This means that reengineering tools recover different information related to different legacy software artifacts and that a KDM repository is progressively populated with the information retrieved. Software analysis tools can be homogeneously plugged into the KDM common repository to generate more valuable information in a standard way.

Software Reengineering Tools

Table 1 compares 13 tools that support reverse engineering and reengineering.

REENGINEERING WITH MODISCO

MoDisco (http://eclipse.org/MoDisco) supports model-driven reengineering and focuses on reverse engineering and restructuring. MoDisco facilitates modernization of existing systems by providing

- metamodels to describe existing systems—for example, the Knowledge Discovery Metamodel (KDM) or Unified Modeling Language (UML),
- discoverers to automatically create models of existing systems—for example, from Java, C, or PHP projects, and
- generic tools to understand and transform complex models created from the existing systems—for example, graphical UML editors.

Figure A shows a reengineering project with MoDisco aimed at evolving IceScrum, a tool that supports the Scrum methodology. First, you select the source Eclipse project, or a piece of a project, that represents the existing system. IceScrum is a Java project, although MoDisco supports other technologies. Second, you analyze the source code and generate a KDM model, which is integrated into a KDM repository. Third, you choose KDM models and transform them into UML models representing the existing system. Fourth, you can apply refactoring patterns and introduce new functionalities to the UML models, at a higher abstraction level. You can make such modifications using the available MoDisco editors or other Eclipse plug-in tools. Finally, because MoDisco is an Eclipse plug-in, you can easily extend or integrate it with other tools—for example, code generators such as MOFScript, which can automatically generate source code for the same platform or a different target programming language.

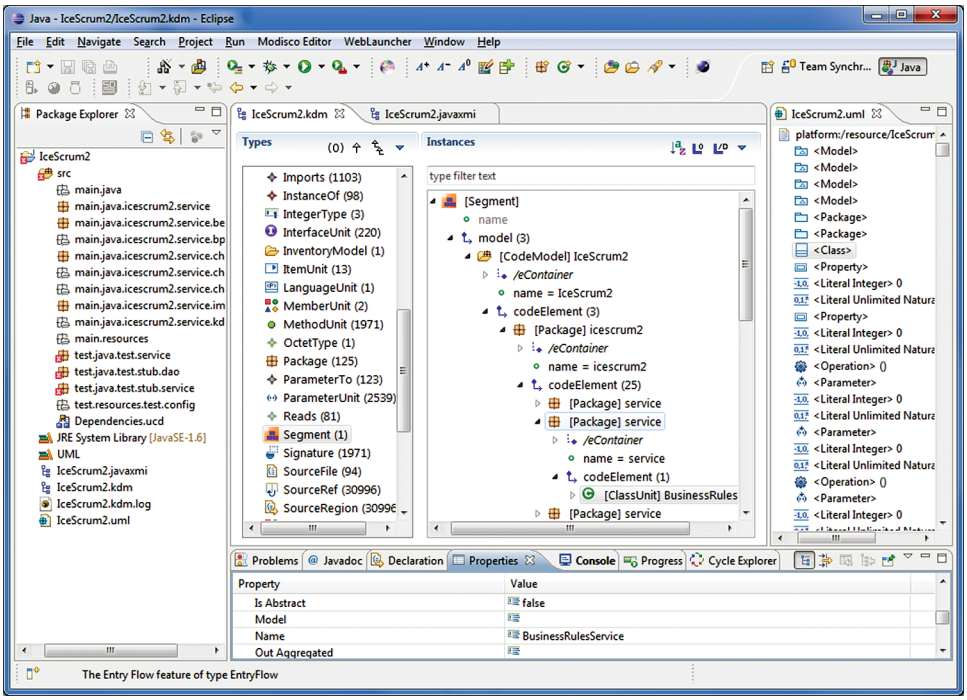

FIGURE A. Reengineering IceScrum with MoDisco. The interface shows source code, Knowledge Discovery Metamodel (KDM) representation, and Unified Modeling Language (UML) models for IceScrum.

Several support both traditional and model-driven reengineering.

Some commercial tools—for example, Microfocus Modernization Workbench, Amelio, BlueAge, and ObeoAgilit—focus on most of the reengineering stages available for I/O platforms and programming languages. Their costs can vary depending on the legacy system's platform, size, and other features. Others focus on one technique. For example, Resource Miner supports static analysis to obtain a code inventory and metrics, whereas CloneDR focuses on detecting and removing duplicate code.

Commercial integrated development environments, such as Visual Paradigm, are progressively incorporating more reverse-engineering techniques, such as automatic code generation from UML models (related to forward engineering) and reverse engineering from code to system design.

Reengineering research has also generated open source tools—for example, MOOSE, MoDisco, and Avalanche—that support ad hoc techniques proposed from academia. Some, such as MoDisco (see the sidebar), are also available as Eclipse plug-ins and can thus be combined with other related plug-ins.

The latest releases of some of the tools, such as Blue Age, Obeo Agility, MoDisco and MOOSE, already follow the model-driven approach.

Here are some hints for successful reengineering:

- Formalize the methodology, and use standards to carry out repeatable reengineering projects.
- Use well-known reengineering techniques and tools.
- Treat reengineering as a change management project with a concrete budget, skilled resources, and management support.
- Align the reengineering project with the company's strategic direction.
- Use techniques for decision support (for example, portfolio analysis) to focus on where reengineering has the biggest yield.
- Use good software engineering processes, such as for configuration control, documentation, and automatic verification.

The software industry has widely used reengineering. However, approximately 50 percent of reengineering projects fail because they produce unsatisfactory results or cost overruns.[5] This is owing to traditional reengineering, which is usually related to ad hoc solutions, and has two main limitations: standardization and automation. Standardizing and automating reengineering projects is necessary to better reuse legacy code and ensure that reengineered code can be further improved over time—independently of the chosen tools.

References

1. E.J. Chikofsky and J.H. Cross, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software*, vol. 7, no. 1, 1990, pp. 13–17.
2. G. Canfora, M.D. Penta, and L. Cerulo, "Achievements and Challenges in Software Reverse Engineering," *Comm. ACM*, vol. 54, no. 4, 2011, pp. 142–151.
3. C. Ebert and R. Dumke, *Software Measurement: Establish—Extract—Evaluate—Execute*, Springer, 2007.
4. *ISO/IEC DIS* 19506, *Information Technology—Architecture-Driven Modernization—Knowledge Discovery Meta-model (KDM)*, v1.1 (Architecture-Driven Modernization), ISO/IEC, 2009, p. 302; www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?ics1=35&ics2=080&ics3=&csnumber=32625.
5. H.M. Sneed, "Estimating the Costs of a Reengineering Project," *Proc. 12th Working Conf. Reverse Eng.*, IEEE CS Press, 2005, pp. 111–119.

RICARDO PÉREZ-CASTILLO is a PhD student at the University of Castilla-La Mancha. Contact him at ricardo.pdelcastillo@uclm.es.

IGNACIO GARCÍA-RODRIGUEZ DE GUZMÁN is an assistant professor at the University of Castilla-La Mancha. Contact him at ignacio.grodriguez@uclm.es.

MARIO PIATTINI is a full professor at the University of Castilla-La Mancha. Contact him at mario.piattini@uclm.es.

CHRISTOF EBERT is the managing director of Vector Consulting Services. Contact him at christof.ebert@vector.com.