

Technology Briefing Report

Reverse Engineering

Identifier	D5.1b Reverse Engineering
Type	Internal result
Activity	WP5.1b
Date	17/09/96
Status	Final
Partner	Engineering
Availability	Restricted

1. Overview

Although no standard definition exists, reverse engineering has traditionally been defined as a two-step process of information extraction followed by information abstraction.

The first step analyzes the subject system to identify its components and their inter-relationships. The second step creates representations of the system in another form or at a higher abstraction level.

Recently, a new approach to reverse engineering was advocated [Ti195] that refines this traditional two-step approach of information extraction and abstraction. The new three-step approach is as follows:

Model: Construct domain-specific models of the application using conceptual modeling techniques.

Extract: Gather the raw data from the subject system using the appropriate extraction mechanisms.

Abstract: Create abstractions that facilitate program understanding and permit the navigation, analysis, and presentation of the resultant information structures.

It is this definition reverse engineering is seen as an activity which does not change the subject system; it is a process of examination, not a process of alteration. It can facilitate the understanding process through the identification of artifacts, the discovery of their relationships, and the generation of abstractions. This process is dependent on one's cognitive abilities and preferences, on one's familiarity with the application domain, and on the set of support facilities provided by the reverse engineering environment.

Reverse engineering may be performed at any abstraction level. It has been used primarily in transforming source code to higher-level representations, such as control-flow and data-flow diagrams, interconnection models, and virtual subsystems [MOTU93]. However, it can also be used at a lower abstraction level, for example to transform binary programs into source code [CG95].

Three of the most important types of reverse engineering are redocumentation, structural redocumentation, and design recovery.

Redocumentation is one of the oldest forms of reverse engineering [Sne84]. It is the process of retroactively providing documentation for an existing software system. If the redocumentation takes the form of modifying commentary within source code, it can be considered a weak form of restructuring. However, it can also be classified as a sub-area of reverse engineering because the reconstructed documentation is typically used to aid program understanding. One can think of it as a transformation from source code to psuedo-code and/or prose, the latter of which is usually considered to be at a higher abstraction level than the former. The documentation produced is typically inline text. However, it can take many other forms, including that of linked documentation accessible via hypertext [TM91], cross-reference listings, or graphical views of the software systems artifacts and relationships [TMO92]. Some of the newer reverse engineering environments also support augmenting the source code with multimedia annotations.

Structural redocumentation consists in using reverse engineering to reconstruct the architectural aspects of software. As a result, the overall gestalt of the subject system can be derived, and some of its architectural design information can be recaptured. As with redocumentation, structural redocumentation does not involve physically restructuring the code (although this might be a desirable outcome).

Design recovery is a sub-area of reverse engineering that uses domain knowledge, external and/or informal information, and heuristics, in addition to traditional source-level analyses, to aid program understanding [Bigg89]. Its aggressive goal is to reproduce all the information needed for someone to fully understand the subject system.

2. Role in evolution process

The understanding of the system, both the current and the desired system state, is the technical basis for determining the particular reengineering strategy to be chosen [Ti196]. It requires analysis,

considering alternatives, and making engineering tradeoffs. Such a technical engineering analysis consists of two major components: choosing the degree of legacy leverage (what can be taken over and what has to be newly created), and choosing the approach for migrating over to the desired system, (how to introduce the changes into the system). Rarely is any single approach appropriate by itself, but engineering tradeoffs need to be considered.

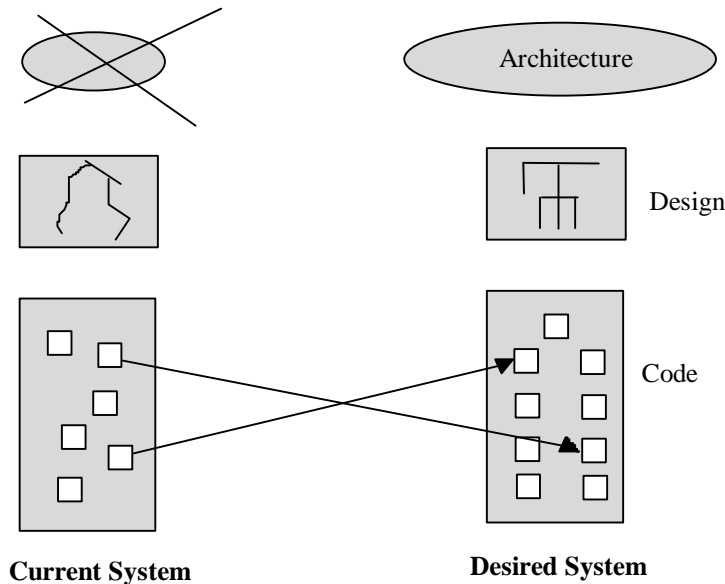


Figure x.1: System evolution

The ability to utilize as much as possible of the existing system in the process of evolving to the desired system may be termed legacy leverage. Both the existing and the desired system can be described in terms of a collection of models. For the legacy system, code exists. Other models may have to be derived from the code or other information sources. Certain abstractions may not exist in the legacy system or may reflect undesirable properties. The goal is to eliminate undesirable properties while at the same time introduce desirable properties. Choices have to be made as to which legacy system models to ignore, which ones to transform, and which ones to leave intact. This is illustrated in Figure x.1.

The choices are driven by the understanding of the legacy and desired system properties as well as their reflection in the different models. In concrete terms, this means that in some cases undesirable properties of legacy systems can be eliminated by massaging the code or transforming the data representation, while in other cases a new architecture or data model has to be developed and only a few system components can be translated into the new implementation language. The change can be introduced in a number of ways. The following are three classic approaches, although hybrid solutions are possible:

New:

This approach ignores the existing legacy system as a solution. The desired system is newly developed, separate from the legacy system (although parts of the legacy system may be recycled). Once completed, the new system is put into operation and the old system is shut down.

Incremental:

This approach incrementally phases out pieces of the legacy system. The architecture of the desired system may be created and a skeleton implementation developed. A mapping between the data

representation of the current and desired system, implemented as a two-way transformation filter, allows the skeleton desired system to run as a shadow of the “live” legacy system. Parts of the desired system implementation are completed and incrementally added to the skeleton.

Evolutionary:

This approach evolves the current system towards the desired system by phasing in new features over time. The legacy system code may be restructured to introduce modularity and partitioning. Desired system properties are incrementally introduced into the current system resulting in an incremental evolution of both the architecture and the system components.

From these considerations, the central and basic role of reverse engineering techniques and the relevant need for supporting technologies is evident either to help deciding and appointing the most adequate evolution approach for a given software system and to help identifying components that can be re-used in the new system.

3. Relevance to RENAISSANCE

The Renaissance project to which reverse engineering technology may be relevant are:

1. support understanding of the current design of the legacy system to be evolved.
2. support the adoption of the of the most suitable evolution policy,
3. support identification of reusable components of the system.

4. Available support

The present section analyses the features of a relevant sub-set of commercial products offering support for reverse engineering activities. The analysed tools are:

- Recycle, from Software Reuse Company
- Refine/Cobol, from Reasoning System
- Revolve, from MicroFocus
- Existing System Workbench, from Viasoft
- Maestro II, from Softlab

4.1 Recycle, from Software Reuse Company

Recycle is a set of tools for reengineering and reusing legacy software. The toolset comprise the following modules:

- Recover
- Repair
- Reslice
- Redeploy

The **Recover** module parses existing code, data definitions, and job control information and derives graphical and textual views of application design. This module supports software design and documentation recovery providing source code views, code structure views, control structure views, data model views, along with design quality and completeness reports. Human knowledge can be input to complete and complement the provided documentation. The Recover module works with various dialect of Cobol, Fortran and Pascal.

The **Repair** module enables to edit and manipulate the recovered application design representations including the source code.

The **Reslice** module 'slices' large applications into separate compilable modules each of which can be modified and maintained using the Repair module. This also enables the reusable parts of an application suite to be sliced out in order to convert them into a new environment.

The **Redeploy** module is a conversion tool tailoring service which enables the creation of any required conversion tools for application redeployment to new languages and/or hardware.

The Software Reuse Company is going to extend Recycle with a new module called **Repertoire**. This module supports an Asset Management/Reuse Library. The module will provide a central information repository that will interface all the other modules, enable multiple users access and provide storage, version and configuration management features of the software assets. Browsing capabilities will also enable to analyse the interactions between multiple application and their common data model. The repository will provide search, identification and impact analysis mechanisms that will enable to identify, retrieve and manage reusable components.

The tools are all sold separately and run on IBM PS/2 (OS2), IBM RS6000, Sun Sparc and DECStation. Versions of the tools for HP and MS-Windows are announced.

4.2 Refine/Cobol, from Reasoning System

Refine/Cobol supports understanding, evaluation, documentation and reengineering of existing Cobol application. By parsing existing programs, Refine builds up a database that contains information concerning different aspects of an application and at different levels of abstraction. This enables to generate and display graphical reports in a variety of diagrams and tables (e.g. structure charts, control flow diagrams).

The deduced application design can be exported in the repository of another tool called Software Through Picture. This is a forward engineering CASE produced by Interactive Development Environments that allows maintenance, enhancement and documentation of the exported applications.

Refine provides a reengineering API, using the features offered by the tool **Software Refinery** (Reasoning System development environment), enables to customise the original tool to meet specific reengineering requirements. Software Refinery also has a built in conversion feature that allows the conversion to other languages, database and platform.

Refine/Cobol supports various dialects of Cobol and runs on Sun Sparc, IBM RS6000, HP 9000/7. Other Refine instances are available for C and ADA.

4.3 Revolve, from MicroFocus

Revolve is a PC-based Cobol analysis tool supporting reverse engineering of large software systems. All major dialects of Cobol are supported as well as JCL, SQL, DB2, BMS, IMS and CICS.

The following principal features are provided by Revolve:

structural analysis: the application structure is represented through a variety of graphs (views) each one concerning specific aspects of the system at a given level of abstraction. Navigation features enable the user to navigate among the different but related views. Elements in each views can be coupled with user comments enabling to enrich their semantic.

impact analysis: given a particular source object, Revolve enables to identify all the location where the object is used. The result is presented in a graphical manner from which it is possible to directly access all the individuated locations.

execution simulator: Revolve provides an execution simulator enabling to navigate through the logic flow of a system,

code quality measurement: Revolve provides a variety of metrics.

4.4 Existing System Workbench, from Viasoft

ESW is a complete management and reengineering solution for legacy systems. Viasoft break down ESW's functionalities into six areas:

Program understanding: this is supported by the ESW's ability to simulate the program execution. This provides a number of ways to view, and to navigate through, complex program logic giving evidence of the program structure. Cross referencing allows to identify how control is passed between paragraphs and sections of programs. In addition, ESW's features facilitates the tracing of data usage and of the relationships between data elements, allows to perform complete impact analysis and to improve program quality by identifying anomalies.

Code change: this is supported and facilitated by an editor that provides an intelligent search facility that allows to identify all items that are directly or indirectly affected by the change.

Testing: ESW testing facility allows to determine the location of a bug, its cause and its effects on the entire source code.

Program reengineering: ESW allows to extract the business functions buried in legacy systems. By indicating a function, ESW extract the minimal set of logic and data needed to perform that function. This extracted and syntactically correct code can be reused during renewal, migration or for new developments.

Program documentation: from the source code analysis, ESW generates complete and easy to use documentation. The produced documents may include:

- graphical charts detailing program structure and logical relationship between components,
- control and data flow information,
- reports and metrics,
- source listing.

Portfolio analysis: ESW provides IS/IT managers with consistent criteria for planning and managing their activities

4.5 Maestro II, from Softlab (MTW component)

Although Maestro II is better positioned as a software engineering platform, one of its component named Maintenance Team Workstation for Cobol (MTW). The main features of MTW are:

1. Understand and analyse the existing system

Parsing tools analyse the subject system and populate the repository with information from programs, copybook, screen definitions, database definitions and JCL. This analysis allows to construct a complete map of the application components and of their relationships.

2. Sustain and enhance existing systems

MTW provides facilities for efficient editing, compiling and testing of modified and new code.

3. Prepare and synchronise the existing system

MTW allows to isolate and rationalise data and business rules encoded within the system and, consequently, to eliminate anomalies and redundancy.

5. Maturity assessment

Available commercial technologies, although sufficiently consolidated, show consistent limitations. In particular, the majority of the commercial tools focus on source code analysis techniques. These techniques, mainly finalised to provide information on the structural aspects of software system, are unsuitable for the identification of any sort of semantic information either at program and at application level.

This results in a very poor support for important reverse engineering activities like design recovery. The few existing tools capable to provide an effective support for design recovery are still available as prototypes and miss valuable commercial references.

An other main and more general problem related with existing reverse engineering technologies concerns their strong dependency with the operational domain where the systems to be analysed run. The wide majority of existing commercial tools is very specialised and only applicable to one, or at most few, operational domains.

6. Inapplicability

Existing reverse engineering technologies are not generally applicable to legacy systems built using RAD or 4GL products unless specific features are provided directly with the development technology.

An example of 4GL product supporting some reverse engineering features is UNIFACE from Compuware Corporation. Incorporating a component-based architecture, the UNIFACE development environment consists of five integrated workbenches that share information from an application objects repository. These components are:

Application Model Manager for creating and maintaining application models

Rapid Application Builder for the graphical creation of forms and reports

Deployment Manager that incorporates a system of native database drivers for transparent database access, application partitioning facilities and distributed processing through interfaces to transaction processing monitors and remote procedure calls (RPCs).

Developer Series for managing team development with version control and access privilege facilities

Personal Series that end users use to query data, build reports and transfer data into popular desktop applications

UNIFACE employs a model-driven approach to development. The model-driven approach ensures that applications reflect an organization's key business issues, not the "technology du jour." It delivers high productivity because applications are rapidly built and maintained from a centrally defined high-level application model, rather than from low-level application code. Applications are quickly adapted to business needs, because only a change in the high-level application is required rather than multiple instances of low-level platforms and database code. Moreover, changes are automatically reflected in all the related application forms and reports.

7. Future development

Reverse engineering needs further work to consolidate existing technologies for supporting design recovery activities and to combine the information provided by these kind of tools with those that can be obtained adopting tools based on parsing techniques.

8. Other comments

None.

9. References

Big89: Ted J. Biggerstaff. Design recovery for maintenance and reuse. *IEEE Software*, 22(7):36--49, July 1989.

CG95: Cristina Cifuentes and K. John Gough. Decompilation of binary programs. *Software---Practice and Experience*, 25(7):811--829, July 1995.

MOTU93: Hausi A. Müller, Mehmet A. Orgun, Scott R. Tilley, and James S. Uhl. A reverse engineering approach to subsystem structure identification. *Journal of Software Maintenance: Research and Practice*, 5(4):181--204, December 1993.

Sne84: H.M. Sneed. Software renewal: A case study. *IEEE Software*, 1(3):56--63, July 1984.

Ti195: Scott R. Tilley. Domain-Retargetable Reverse Engineering. PhD thesis, Department of Computer Science, University of Victoria, January 1995. Available as technical report DCS-234-IR.

Ti196: Scott R. Tilley. Perspective on Legacy System Reengineering (Draft Version 3), SEI Reengineering Center Publication, Summer 1996.

TM91: Scott R. Tilley and Hausi A. Müller. INFO: A simple document annotation facility. In Proceedings of the 9th Annual International Conference on Systems Documentation (SIGDOC '91), (Chicago, Illinois; October 10-12, 1991), pages 30--36, October 1991.

TMO92: Scott R. Tilley, Hausi A. Müller, and Mehmet A. Orgun. Documenting software systems with views. In Proceedings of the 10th International Conference on Systems Documentation (SIGDOC '92), (Ottawa, Ontario; October 13-16, 1992), pages 211--219. ACM (Order Number 613920), October 1992.