METHODS AND APPROACHES FOR REVERSE ENGINEERING

Martin Chlumecký
Czech Technical University in Prague, Dep. of Computer Science and Engineering, Karlovo náměstí 13, 121 35 Praha 2, Czech Republic;
chlumma1@fel.cvut.cz

ABSTRACT:
Legacy Systems refer to outdated computer systems or application software which is still used in many corporations and institutions. Legacy systems persist because of the expense and sometimes it is only way to use their knowhow. Maintenance of these systems is a very challenging part. System documentation is often out-of-date or missing. This paper aims to procedure and methods for reversing of the legacy systems. It is described how to transform legacy procedural systems into an object-oriented architecture. Some described methods were applied to a real instance of a legacy system. Finally, it is summarized an observation during the process of reversing and there is evaluated a chance to make process more efficient.

KEYWORDS:
Legacy system, procedural source code, object-oriented design, re-engineering, FORTRAN

1. Introduction

This work describes a procedure how convert the procedural software into object-oriented architecture.

Programming in an object-oriented structure has many advantages. It is easy way to create applications. These applications are highly flexible, adaptive to change requirements and to extension exists applications. Wide collections of mechanisms for classes, instances, an inheritance and a polymorphism which are built in object-oriented languages enable to create a reuse source code. Last but not least, an advantage of object-oriented systems is its maintenance.

However, a procedural paradigm has many disadvantages. The legacy system is an old application that continues to be used. A coast for maintenance of the legacy systems is rapidly escalating. An orientation in thousands lines of procedural code is very painful. These systems are user unfriendly, because an input and an output of the legacy systems are limited. Nevertheless, today many legacy systems which are still using exist. The legacy software systems are still in use because it implements a useful business tasks. Therefore, the institutions want migrate a legacy system in to the object-oriented architecture, and so remove disadvantages of such a system.

Reverse engineering is used to transform the legacy system to a new object-oriented system. There are many approaches and methods of the reverse engineering. In this paper is described one of the methods which was applied to the legacy software used in a hydrodynamics. The Institute of Hydrodynamics of the Academy of Sciences of the Czech Republic (IH) had used legacy software for simulation of hydrological models. The legacy software was written in FORTRAN in 1970. A size of an input and an output data is limited. A graphic interface is only in a text mode, which complicate a work with an input and a result of hydrological models. A format of an input data is represented by a text file. In the file all items has a strict position what can easily create parsing errors.

It is better to use software architecture of a legacy system then to write new software however, extracting architecture from the legacy software is very challenging task.

This paper is a technical overview and is organized in 3 sections. In section 2 are described methods which were used. In the section 3 is described analysis of the source code and create a legacy architecture that is presented by UML diagrams. The section 4 provides detail about problems joined with reverse engineering.

2. Reverse engineering

This chapter describes methodology of a reverse engineering and specifies a level of abstraction which this paper solves. Base process and an output of a reverse engineering are also described.

2.1 Legacy system

The legacy system that was re-engineered is used in hydrological research. A hydrological core of this system consists of the Sacramento Soil Moisture Accounting model [2] (SAC-SMA). The SAC-SMA model is a complex conceptual hydrological model.

The structure of the legacy source code is as follows. A size of source code is about 30,000 lines which are written in FORTRAN. The source code is divided into 225 files. The legacy software is modular. Each file contains one subroutine. However, a large part of the source code − about 40% − contains systems routines that are not associated with software architecture. Comments of majority subroutines are on good level and are possible to discover what each subroutine does. A documentation is available but it has being out-of-date. Nowadays, the legacy software has not any maintenance and any support.

Today, the legacy software has many disadvantages. The legacy software use only text output. It is impractical for work with an output that is represents by a graph. On a long input the graph may contain up 4,000 lines which is very user unfriendly. Processing output data is also impractical. The text output file contains many meta-data. This meta-data must be filtrate and a useful data must by transform to another applicability form e.g. XLS or XML. This formed data can be used in modern tools.

The reason for transformation is that the legacy software cannot be executed on new operating systems – Windows XP or highest – but the IH institute needs this legacy software. A question “Why not just to recompile the source code of the legacy system by new FORTRAN compiler?” may arise. The answer is simple. The legacy source code uses program constructions and structures which are not compatibles with new FORTRAN compiler. That is a main reason why the legacy software has been re-implemented. The next reason is range of input and output data. The input data is limited by FORTRAN 95 compiler. Re-implementing gets benefits e.g. possibilities improved and extend the legacy software about new functions.

Reverse engineering is defined [1] as the process of analyzing a subject system to identify the system's components and their interrelationship and create representations of the system in another form or at a higher level of abstraction. Reverse engineering involves

extracting design artefacts and building abstractions that are less dependency on an implementation.

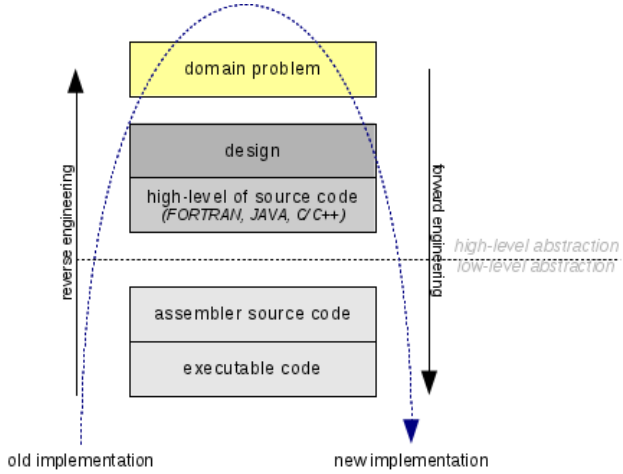

Fig. 1. The levels of reverse engineering

The reverse engineering is wide discipline of the software engineering. It aims to several levels. The figure 1 shows the levels of the reverse engineering. This paper describes re-engineering of FORTRAN source code. It is occupied by a higher-level of abstraction.

2.2 Reverse engineering

There are several methods how to use the reverse engineering:

1. Manually rewrite the existing source code,
2. Use an automatic language transformer,
3. Redesign and re-implement original software.

The first method fundamentally means to rewrite completely the legacy software to a target language. This approach does not need tools. However, it is usable only for small legacy software. This method produces many errors because humans make mistakes. Further, it is very time-consuming and has a low efficiency.

The second method is not time-consuming because it uses supporting software. This approach transforms a legacy source code to a target language. It is fast and with minimal number of errors. This way generated a new source code is confused. The possibility of intervention, enlargement and modify is minimal.

The third method eliminates the main disadvantages of previous methods. This method starts by analyzing requirements of legacy software. It continues by redesign of software architecture and the method ends by implementation of new software. The advantages of this method are a possibility fixes known bugs, enlargement about new functionality and to create a documentation of new architecture.

2.3 Methodology of reverse engineering

The method evaluated in this paper outlines steps how to take the reverse engineering on a procedural system – a source code – and transform it to an object-oriented system.

The steps to be taken in the reverse engineering process are as follows:

1. The domain expert e.g. users of legacy software, creates a function list. It contains functions that identify crucial functionality of legacy system.
2. The next step is generating the call graph. Tooling or manual analysing of legacy source code and to generate the call graph that shows a flow-control of legacy system.
3. Context list and dependency list are created. The dependency list identifies all functions and procedures invoked by other functions or procedures.
4. The fourth step is very important; it is an identification of objects. In the step are potential objects identified.
5. The sum data is collected. The sum data contains information about each function which is not in the function list. It is information about types, names of parameters, variables.
6. Objects candidates are identified. Some objects are joined or split.
7. The domain expert chooses the objects. The experts examine the object candidate and determine whether they are reasonable. Common variables of two or more functions are evaluated as object attributes. The rest of functions included in the function list can be converted into individual objects or a package.

The following process gives information about legacy system:

1. The number of domain object
2. The number of function and its lines of source code in a legacy system
3. The basic architecture of legacy software
4. The degree of coupling.

The coupling or dependency is the degree to which each program module relies on each one of the other modules. Coupling is usually contrasted with cohesion. Low coupling often correlates with high cohesion. Low coupling is often a sign of a well-structured computer system and a good design.

3. Methodology application on the legacy system

Previous chapter describes a process of the reverse engineering. In this chapter is described a procedure of real re-implementation of a legacy system into an object-oriented architecture.

3.1 Create a function list

In most case a reverser do not knows anything about a domain problem. Therefore, a reverser must get information about a domain problem from other sources. Main sources are expert users of legacy software. A next source is documentation of legacy software or a documentation of a domain problem. Often, design documentation is out-of-date and thus it is

unusable. For the domain problem of this example has been available a complete documentation. The second source of information was expert user.

The both sources of information provided the function list. The list contains main functionality which provides the hydrodynamics models.

3.2 Generating call graph

The call graph was generated manually because the legacy software has been smaller extent. However, this phase without tools was time-consuming. The graph is composed by nodes and edges and takes the flow control of legacy software. The node represents a subroutine and the edge represents a call from higher subroutine. Figure 2 depicts a part of sample the call graph.

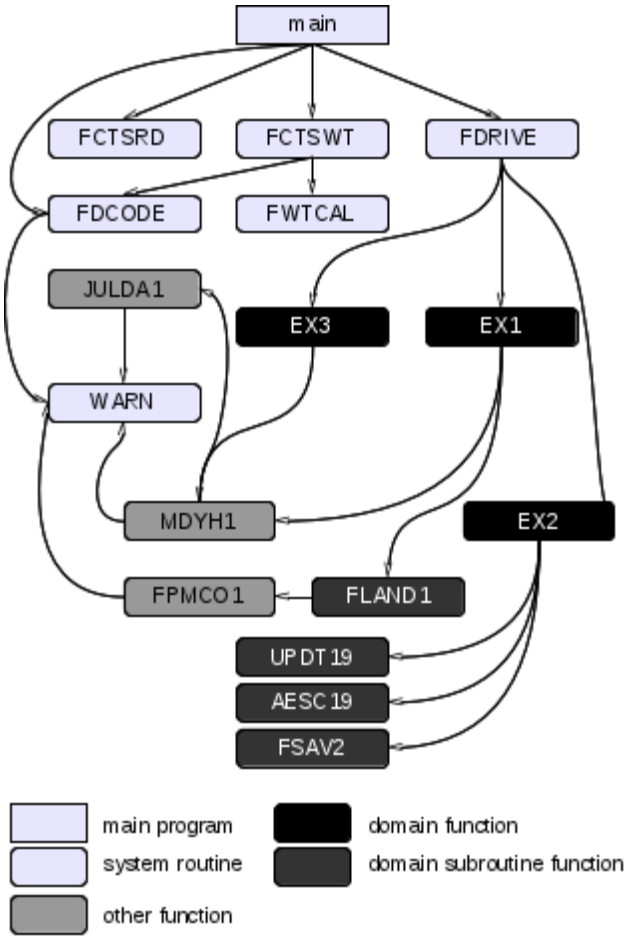

Fig. 2. Oriented graph of main procedure

3.3 Create context list

All other functions which are not contained in the function list are described and analyzed. This function does support for functions of the function list. In this steps are captured variables and parameters of functions. It was captured dependency among all context functions.

3.4 Identifying objects

At this phase is created a new object-oriented model.

First step, all functions and methods from the function list and the context list are transformed to classes. All variables are in this step ignored. Associations among the classes reflect the call graph. This created model represents mutual call of the functions – classes – and the model is analogy of the call graph.

Second step is deriving variables. It means to determine what the variables will be local and what the variables will be class attributes.

Third step determines the classes of the new object-oriented model. The classes – functions and method – which are in a close bond can be merged to one class. How determine a condition which set has close bond is a very difficult task. Often, it is dependent on a specific case and on quality of domain problem information.

Last step are separated the classes into packages. Each package includes a set of the classes which has a similar functionality.

3.5 Re-designing architecture

At this phase is the legacy system extended by new functions or can be fixed known bugs of system. The packages that were defined in previous section are extended about new classes which bring new functions or new packages added into the exits object-oriented model, example of a new package is package which provides classes of graphical interface.

3.6 Converting program language

At last phase is the legacy sources code converted into the source code of a target language, in this case C/C++. The source code inside the legacy function is converted using tool. The functions which were added in phase 3.5 must be written manually, because it is the new functionality of the new system.

4. Experimental results

In the process of reverse engineering has been noticed several problems: global variables, GOTO command, determining data types of common block, determining system's functions.

The global variables bring a following problem. If the global variables are used in some classes together then it is difficult determined which class will be holder of the global variable. Some of the global variables can be transformed as parameter of any methods, because it is only carry over a status flag of previous call functions.

The GOTO command is dangerous for design of a source code. If it is used non-consideration then a source code can become the “spaghetti code”. Because FORTRAN systems often have unstructured source code and to use many GOTO command then the call graph can be very complex. In this case is very difficult identification of objects how is described in section 3.4. Solution is to use restructure, so we are able to eliminate the GOTOs which are unnecessary.

5. Conclusions

The goal of described procedure of reengineering was re-implement the legacy system which has been used in IH. In this paper method for transforming procedural legacy system into object-oriented system has been presented. The method focuses on the identification of objects whose has been hidden in the legacy source code. Since, it was reversed only 15% of the legacy system then all described methods was used manually. However, using those methods on a large project it would be very time-consuming.

The result of this work is re-implemented procedural system for hydrodynamics. The new implemented system is based on an object-oriented architecture. During, solving this task was observed few problems. First problem is determined what parts of the legacy system should be included in the new system. Next problem is processing global variables.

There are many methods how objects identify. Future work is concentrated on the method of identifying object in procedural code and how this process can be make more effective.

LITERATURE

[1] CHIKOFSKY, E.H. Reverse Engineering and Design Recovery : A taxonomy, *IEEE Software* Vol. 7, No. 1, IEEE Press, Piscataway, NJ, USA, pp. 13-17.

[2] BURNASH, R.J.C. Calibration of the Sacramento Soil Moisture Accounting Model. *NOAA Technical Memorandum NWS*. November 1999, č. 217.

[3] ANDERSON, Eric. Calibration of Conceptual Hydrologic Models for Use in River Forecasting. *NOAA Technical Memorandum NWS*. November 1999, č. 217.

[4] BAUMANN P. Semantics-based reverse engineering. Technical Report 94.08, *Department of Computer Science University of Zurich*, Switzerland.

[5] BLAHA, M.R. The case for reverse engineering, *IT Professional*, vol. 1, Issue 2, IEEE Press, Piscataway, USA, pp. 35-41.

[6] BRIAND, L.C. Towards the reverse engineering of UML sequence diagrams, *Proceedings of the 10th Working Conference on Reverse Engineering*, pp. 57-66.

[7] JACOBSON, I., Re-engineering of Old systems to an Object Oriented Architecture, *Prec. OOPSLA*, 1991, pp 340-350.

[8] MEYER, B. Object-Oriented Software Construction, *Prentice-Hall International*, London, 1988, pp. 3-26.

[9] CANFORA, G., CIMKITILE, A., MUNOR, M.: Reverse-Engineering and Reuse Reengineering. *Journal of Software Maintenance*, vol. 6. , No.2, 1994, p. 53.

[10] HUTCHENS, D. H. System structure analysis: Clustering with data analysis. *IEEE Transactions on Software Engineering*, pp. 749–757, 1985.

[11] LIU, S.S. Identifying objects in a conventional procedural language: An example of data design recovery. *In Proceedings of the Conference on Software Maintenance*, Nov 1990. pp 266-271.