Re-animation of computer programs
Meijer, F.J.

Citation for published version (APA):
Meijer, F. J. (2001). Re-animation of computer programs

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
7 GENERAL DISCUSSION AND CONCLUSION

7.1 INTRODUCTION

The background to this study was a renewed interest in the portability and code reconstruction problem. The portability problem was recognized from the beginning of the computer era: machine code developed for one type computer cannot run on another. Technological advancements were made at such a rate that a computer became almost obsolete before installed. This further contributed to the problem. Users readily chose other systems causing computer companies to lose market share when new models did not allow old programs to be run. Emulation was offered as a solution. In the beginning users rejected this method because it was sluggish. Nowadays, with state of the art computers an emulated program runs faster than on the original.

With emulation there is no need to have the source code. However, when users want the program modified, this need becomes urgent. How to access the source code if it is no longer available is the primary purpose of this thesis. It proved difficult to find a general solution to the code reconstruction problem. In the past many formulations were proposed. The problem was still considered unsolvable. In this study a new formulation was sought. While everything in the computer environment was rapidly changing, this problem remained virtually not addressed. One reason may have been that in the beginning computer programming expenses were very high. Computerizing reconstruction of machine code could have resulted in reduced cost. It was then considered more cost effective to reprogram existing software. Much effort was spent on designing machines that can process different machine language programs. These attempts, for the most part, proved to be unsuccessful. The problem of accessing obsolete machine code moved to the background. The computer community shifted its attention to re-engineering: the re-use of routines of general interest.
Now, in retrospect, the reconstruction of obsolete machine code can be accomplished by software to preserve its contribution to the historical development of computing.

The code reconstruction problem has to be reformulated. Translation of each program statement separately has not worked. Instead, software was developed to generate data necessary to reconstruct the original code of obsolete machine code programs. In this way a step-by-step approach developed.

The huge demands on primary and secondary storage capacity have led to new technologies that removed previous constraints. This is especially true for the reconstruction of software that is only available in machine code. The step-by-step approach indicated above proved to be most appropriate.

That is not to say that a full transformation can be achieved. There will remain some problems that can only be solved by the user of the transformed software. The same is true for any new program. It is left to the user to customize computer programs. The user can select certain features that facilitate interaction. In a similar way users may have to intervene in the reconstruction of old programs when problems arise.

In the past formulations of the code reconstruction problem that permitted user involvement were considered inadequate. It was deemed necessary by the existing computer establishment that all variations and difficulties were made part of a general framework. They could then be resolved without human intervention.

In the approach developed user intervention plays an essential role. In this chapter a number of aspects of user intervention will be discussed.

### 7.2 LOOKING FOR A JUSTIFICATION

There are some aspects of the code reconstruction problem that can be represented in a general framework. Therefore, a conceptual structure is used. Concepts such as architecturally correct, error free, order of errors, discriminating data and control variables, and the distinction between human and mechanical agents in connection with the notion ‘event sequence’ play a role.

To justify the efforts undertaken to reconstruct machine code the original argument for providing software is considered. Gaining access to
aids that are no longer accessible has an intrinsic value; there is also the serendipity aspect. By reconstructing software, only available in machine code, specific structures are made visible and recognizable. MALIN offers the possibility to see how obsolete programs differ from current ones. In this sense the package offers a contribution to the accumulation of knowledge. This is especially true for earlier programming techniques.¹

Each of these advantages implies that the human user should not and cannot be fully modelled. The question is not whether the human user should be able to intervene, but where, what the user can contribute, and what can be made into a software package.

7.3 THE ROLE OF THE USER

The extra value that derives from user involvement may be achieved in different ways. One method is to educate users. They become more competent in programming using low and high-level languages by applying the package to old software. Another method is to provide users with an interface that makes full use of previous experience as well as their intelligence. The first method was relatively widespread in the early development of computers. Their use was restricted to a specific group. There were the programmers and there were the users. The latter had to provide the former with well-formed statements in their activity language to enable them to code these statements into computer language. This was the responsibility of the first group. Coding was a highly specialized and complex task.

Over time this situation changed dramatically. First, the number and variety of users increased. Secondly, the onus shifted from the technical machine language to more user-friendly languages². This expanded the group of programmers because these languages simplified the art of programming. The latter method was encouraged during the development of MALIN³. The source code was made available to experimental users. This

¹ For an introduction to the history of programming see: "In praise of Wimps" by R. Hull and "A History of Modern Computing" by P.E. Ceruzzi.
² For example, Visual Basic, a product of Microsoft, simplifies such tasks as manipulating windows and file handling.
³ MALIN is a series of programs available in source code. Adapting it to other computers, even using other languages, should not cause severe problems.
is contrary to common practice. Software houses often explicitly forbid the generation of source code.

### 7.3.1 Addressing experience

Users of MALIN can adapt the provided software and the transformed programs. This is possible because the package is written in a modular form. Its main purpose is code reconstruction. This obviously is not sufficient for most users. It is possible to change the reconstructed code because both the assembler source code and the transformed code are produced.

Adaptation to a new working environment adds an important value to the reconstructed software. For example, old programs assume, as a rule, that a black and white monitor or even less advanced I/O devices are used. So, the capabilities of color monitors are seldom used in old programs. A decision by the user is required at this point. Maintaining procedures for directing output to a punch device or cassette makes little sense in today’s environment.

Users can also start adjacent activities with the package. For example, they can modify it to enable reconstruction of programs written for another processor. They can make the program suitable for other computers. Also interventions by the user should ensure that the new software increases their competence. It implies that a user can perform a certain activity and maintain it if problems are encountered. Competence enhancement can be achieved by a variety of means. These generally imply changing the environment by using an existing support system or by creating a new one.

Competence enhancement of users of software should be an objective in transforming old computer programs. This may extend to using different assembly languages. Not only may insight be gained into old problem solving methods, but also in the way that a modern processor has to be programmed.

### 7.4 THE KIND OF KNOWLEDGE INVOLVED

Sciences, in general, strive for knowledge of the environment. In the behavioral sciences how to work with it is important. This suggests that a
different kind of knowledge is sought. The reconstruction problem was reformulated with the aim of incorporating such knowledge. It is assumed that the computer cannot be considered as only a technical device. It is primarily a support system.

An aspect that was discussed in chapter two was the importance of language and the construction of user languages. Some user languages are better than others in that they elicit more useful local knowledge. The call "timber!" used by lumberjacks is short but very meaningful to them. A tree is going to fall, so you better get out of the way. This same call in an office environment would not elicit the same response.

The form of such a language is influenced by the development of previous user languages (van Haaster, 1991). Over time a language may become more definitive and more powerful in changing the activities of those who use it. Well-known user languages can be found in the instructions for the use of technical equipment. Those used by medical suppliers are often insufficient, not geared to the user. They do indicate that something has to be done, but very seldom inform the user as to what happens when deviations occur.

More powerful languages have been developed to inform the users of software. Usually the instructions are expressed in natural language\(^4\). They must identify and define precisely those events in the mechanical agent that will become usable to the human one. The events that the mechanical agent should generate must stimulate the human one to execute the 'right' combinations of events to facilitate the user to execute the desired events. Most programs are now menu driven.

There are technical devices to facilitate this kind of stimulation. The keyboard or the mouse are examples. Such devices enable users to become more skilled. Skill levels may vary. The more restrictive technical devices are the more they will be based on the language used by experts. Few decisions by the user will be necessary. Such devices require a strict model of the user.

MALIN was constructed on the assumption that it does not require a high degree of computer expertise. It maximizes minimal skills.

\(^4\) Modern working environments like the desktop use icons to invoke commands. The quality of the icon depends on its information carrying capacity and the sequence of events it invokes.
7.5 RECAPITULATION

Discussion of the code reconstruction problem has been confined mainly to computer programs, only available in machine code. The problem is assumed to belong to the field of computer science. In this study, however, it was placed in a different context. It was assumed that mechanical and human agents both produce sequences of events, and that they aim to produce preferred sequences.

The mechanical agent should help the human one and vice versa, the human one must assist the mechanical. This implies that it is justified to expect the user of MALIN to decide certain issues that the mechanical agent cannot.

To facilitate these processes powerful special languages are required. They should address the users in such a way that they need minimal previous expertise. Designing such languages may be interesting. Growing old, as applied to computer programs, implies losing capabilities. This results in communication problems.

With every improvement of computers and their software the older ones become more difficult to access. Documentation and source code get lost. A solution to retrieve them is offered by MALIN.

The approach developed is new in the sense that it does not treat code reconstruction as mainly technical. Although that is an important aspect, this thesis emphasizes the importance of the human interface in code reconstruction. An important consideration was whether an added value is achieved by the construction of such a package.

When code reconstruction is considered, in the context of the behavioral sciences, the traditional boundaries of the field of computer science have transcended in the way the package indicates.

The research has shown that the dynamic method to analyze bit strings is a valuable strategy for code reconstruction. Chapter 4 discussed the problems to be solved and why dynamic analysis was preferred.

A general solution to the problem of code reconstruction was sought. This problem is too complex to be solved by the development of a program that is so prolific that it could solve it. In this thesis reconstruction of programs only available in a specific machine code was the main subject of interest. The field of research was further confined to machine code developed for CP/M machines. A criterion for valid reconstruction
was that the reconstructed code could be transformed into code for another computer. Chapter 5 was devoted to the implementation of the developed software called MALIN.

From the beginning of the computer era users asked for the source code. Producers usually refused to provide it. This has in various ways hampered software users and those teaching and studying software development technique. Users were not able to adapt the software to their specific needs. Those interested in the art of programming were refrained from gaining insight into how it was done in industry. Students also had difficulty in learning how problems were solved in practice.

A change in attitude gradually seems to have taken place. Some software producers provide users with their source code. So, for the present and future this problem seems to be solved. This thesis provides a solution for software developed in the past. Teachers can now demonstrate how to reconstruct and use code. They can show how problems were solved in the past, with limited sources. This thesis also indicates how software, developed for other machines, can be reconstructed and adapted to other machines.

There is still much interest in code reconstruction. Not only in the technical aspects, but also the ethical ones. According to a publication by The University of Queensland, decompilation is usually allowed:

- for the purposes of interoperability where the interface specification has not been made available;
- for the purposes of error correction where the owner of the copyright is not available to make the correction;
- to determine parts of the program that are not protected by copyright (e.g. algorithms), without breach of other forms of protection (e.g. patents or trade secrets).

Firms offer software to accomplish code reconstruction. For example, a disassembler for the Texas Instruments calculator was developed by Ticalc. The Source Recovery Company offers to disassemble assembler

---

5 Sun provides their users with the source code of its software. The success of LINUX is attributable, in part, to the fact that users have access to its source code.
6 Archive.csce.uq.edu.au
7 www.ticalc.org
code to the macro level. It claims it can recover assembler or COBOL in the IBM environment of VSE or VM programs. Interactive disassembling is available for Windows by CCSO. Software Migrations Ltd designed a powerful program to migrate IBM 379 Assembler modules into equivalent readable and maintainable COBOL. V Communications Inc. offers Sourcer, written by Jean-Louis Seigne. It is a commercial program for disassembling X86 binaries.

In the public domain many disassemblers are available. In 1992, Clark A. Calkin won first prize in a disassembly contest with his “Masterful Disassembler Intel 80486”. A free Internet download is available, but the accompanying handbook must be purchased. Other free downloads are available for Windows such as “BORG”, written by Paul Young. It is a disassembler for Windows 32-bit binaries. “WDASM”, written by Eric Grass, disassembles Windows 3.1 programs.

For the Macintosh environment a disassembler called “MacNosy” can disassemble applications, files, and the ROM contents. It is applicable to 68K machines and the PowerPC.

This thesis is an introduction to the problem of finding a general solution to the code reconstruction problem. It concentrated on accessing software of the obsolete CP/M computer. Although it restricts itself to solving a particular code reconstruction problem, it can be readily adapted to other combinations of computers. The Appendix includes a technical description of the problem solution. The entire MALIN Package can be downloaded from the author’s internet website.

---

8 www.source-recovery.com
9 www.ccsso.com
10 Is Decompilation Possible? in archive.csee.uq.edu.au
11 www.lexitech.com
12 http://home.hccnet.nl/f.j.meijer