Journal of
Information Technology & Software Engineering

Mancas, J Inform Tech Soft Engg 2013, S5
DOI: 10.4172/2165-7866.S5-e001

Editorial Open Access

Should Reverse Engineering Remain a Computer Science Cinderella?

Christian Mancas*

Department of Mathematics and Computer Science, Ovidius University, Constanta, Romania

Reverse Engineering

According to Miriam-Webster, *reverse engineer* (RE) means "to disassemble and examine or analyze in detail (as a product or device) to discover the concepts involved in manufacture usually in order to produce something similar", was used for the first time in 1973, and is exemplified, at least apparently paradoxical, only by "They *reverse engineered* the software".

Wikipedia, which contains some 10 A4 pages on RE, plus many other ones on Wikibooks and Wikimedia Commons, defines it as "the process of discovering the technological principles of a device, object, or system through analysis of its structure, function, and operation. It often involves taking something (a mechanical device, electronic component, computer program, or biological, chemical, or organic matter) apart and analyzing its workings in detail to be used in maintenance, or to try to make a new device or program that does the same thing without using or simply duplicating (without understanding) the original".

Moreover, Wikipedia considers that RE originated in the analysis of hardware for commercial or military advantage and was only later applied to legacy software systems, not for industrial or defense ends, but rather to replace incorrect, incomplete, or otherwise unavailable documentation. My opinion is that, just like Molière's Mr. Jourdain, *Le bourgeois gentilhomme* who discovered very late that he was speaking in prose for forty years without knowing it, mankind always reverse engineered and will continue to do it forever.

Buildings (from monuments, temples, churches, and devotion shrines to fortresses, castles, palaces, villas, houses, block of flats/offices), as well as weapons (from the flint pre-historic ones to today's missiles), furniture, clothes, dishes, vehicles, and craftsmanship (from pottery to jewelry), etc. were reverse engineered since the dawn of humanity, still are, and will always be.

Intelligent military and political leaders always tried to reverse engineer opponents' and allies' strategies and tactics, from their troops and weapons disposal on battle fields to their rhetoric, diplomacy, arming programs, life/social philosophy, etc.; could they ever stop it? Coaches were, are, and will always reverse engineer strategies and tactics of their competition, since at least the ancient Greek Olympic Games.

Philosophers, historians, anthropologists, ethnographers, sociologists, etc. are, axiomatically, reverse engineers of thoughts, beliefs, culture, behaviors, etc., just like physicists, mathematicians, astronomers, geologists, biologists, physicians, etc. are reverse engineering the laws, origin, evolution, and possible futures of the universe, as well as astronomic, micro-organism, vegetal, animal, and human bodies. Almost all of us living beings are very frequently, even if generally unconsciously, reverse engineering thoughts and behaviors of our children, parents, loved ones, friends, adversaries, colleagues, employers, employees, suppliers, customers, etc., and sometimes even of ourselves.

Motivation behind RE is impressive and ranges from espionage (military, technical, commercial, sports, etc.), analyzing potential patent infringement, nefarious industrial and service copying, malware and anti-malware design and development, to academic and scientific challenges, forensic investigations, dental or surgical prosthetics, tissue-engineered body parts, surgical planning, architectural, construction, archaeology, paleontology, etc. documentation and measurement, legacy software and data files/bases deciphering, and pure, irrepressive curiosity.

Although software reverse engineering (SRE) was born at least some 32 years ago (see, for example [1-3]), Y2K proved [4], among others, that SRE was unacceptably under-developed and triggered a much more coherent approach in this field, from college and university curriculum, to tools design and development, as well as research. Dually, malware proliferation [5], fierce IT commercial competition, ethical and legal concerns are significantly impacting SRE negatively.

As this is an Editorial paper for a first issue of the Journal of Information Technology and Software Engineering dedicated to SRE, everything that follows is focused on SRE, for providing an overview of this sub-universe, its tools, and its ethical and legal connotations.

Software Reverse Engineering

Basically, SRE [6] aims at figuring out what software that you have no or only partial source code (generally only "binary" one) for does, i.e. to derive its architecture and design information, to the degree that you can either modify or reproduce it in another independent software work.

SRE is generally considered to be of two types: software development and security related. In the general sense, ground-up reverse engineering is very hard, and requires several engineers and a good deal of support software just to capture all of the ideas in a system. In the first category fall, for example, interfacing to legacy code, adding/modifying functionality to/of existing software, evaluating software quality and robustness (e.g. make sure that an operating system (OS) does not contain any major vulnerabilities, security flaws, etc., and make it as hard as possible to allow hackers to crack it), breaking copy protection (e.g. disabling time trials, defeating registration, etc., to "impress" your friends, colleagues, and/or the FBI and/or save some money by getting commercial software for free), while the second includes studying OS and any other software application for attacking and infecting them, as well as, dually, studying malware for designing, developing, and maintaining anti-malware applications.

Obviously, even if rarely considered as such, inferring conceptual data schemes not only from legacy hierarchical or network, but also from relational databases (dbs) [7] is also a SRE-type task: for example,

***Corresponding author:** Christian Mancas, Associate Professor, Department of Mathematics and Computer Science, Ovidius University, Constanta, Romania, Tel: +40722357078; E-mail: christian.mancas@gmail.com

Received December 21, 2012; **Accepted** December 27, 2012; **Published** December 29, 2012

Citation: Mancas C (2013) Should Reverse Engineering Remain a Computer Science Cinderella? J Inform Tech Soft Engg S5:e001. doi:10.4172/2165-7866.S5-e001

MySQL does not provide Entity-Relationship type diagrams; *MS SQL Server* provides them, but with all foreign keys between two tables consolidated into a single one and without showing what column references which one; even in *MS Access*, which is providing all such information too, it is much more easier to understand all the details it encapsulates in its structure and associated constraints by reverse engineering its scheme into a higher level data model one (e.g., in the *Elementary Mathematical Data Model* [7], the facts that column *Capital* from table *COUNTRIES*, referencing any unique column from table *CITIES*, not allowing either nulls or duplicates are consolidated in the definition of the totally defined one-to-one mapping *Capital* : *COUNTRIES*↔*CITIES*, defined on *COUNTRIES* and taking values in *CITIES*).

Even if impossible to make it infallible, *anti-reversing* tries at least to make hackers life as hard as possible, hoping that they will eventually give up reversing, by eliminating symbolic information, encrypting code, obfuscating programs, and embedding anti-debugger code [6]. For example, *MS Windows API* also includes *IsDebuggerPresent* and *NtQuerySystemInformation*, two tools for detecting the presence of debuggers in user and kernel mode, respectively.

SRE Tools

There are different kinds of SRE tools: many are specific to the types of protection that must be overcome to reverse binaries, while several others just make the reverser's life easier. For the most part, SRE tools fit into the following 7 categories:

- ✓ *Disassemblers,* which take the binary machine language codes and display them in a friendlier format, also extrapolating data such as function calls, passed variables, and text strings, thus making executables (hex bytes sets) look more like human-readable code. Perhaps the most frequently used is *IDA* (free version available at [8]).
- ✓ *Debuggers*, considered by many the SRE bread and butter, first analyze the binary, much like a disassembler, and then allow the reverser to step through the code, running one line at a time to investigate the results, which is invaluable for discovering how programs work. Some of them even allow certain instructions in the code to be changed and then run again with these changes in place. *Ollydbg* [9] and *WinDbg* [10] are one of the most used debuggers (but only for user, not kernel mode binaries).
- ✓ *Hex editors* allow viewing and updating the actual bytes in binaries, searching for specific bytes, saving sections of binaries to disk, etc. *HxD* [11] is one of the free favorites of many.
- ✓ *Portable Executable (PE) and resource viewers/editors*: *MS Windows* run binaries (and *Linux* for that matter) have a very specific beginning data section storing for the OS how to set up and initialize them (e.g. how much memory it will require, what support DLLs the program needs to borrow code from, information about dialog boxes, etc.), which is called the PE. Obviously, PEs are very important in SRE, as they give the reverser much needed information about binaries; moreover, eventually, one will want/need to change PE data, either to make programs do something different than what they were initially designed and developed for or to change them back into what they originally were (e.g. before a protector made the code really hard to understand). Similarly, programs resource sections (e.g. graphics, dialog and menu items, icons and text strings, etc.) are altered too with such editors. From the plethora of free PE and resource viewers/editors, *CFF Explorer* [12] and *LordPE* [13] are the most frequently used.
- ✓ *System Monitoring (SM) tools*: especially for studying malware (but not only) it is of the utmost importance to see what changes a program makes to the system (e.g. are there registry keys and/or .ini files created or queried? are separate processes created, perhaps to thwart the program SRE?). *Process Monitor* [14], *Process Hacker* [15], and *RegShot* [16] are probably the most used SM tools.
- ✓ *Miscellaneous SRE tools and documentation*, include scripts, unpackers, packer identifiers, etc, as well as *Linux* source code [17] and *MS Windows API* [18].
- ✓ *Databases RE tools* are especially used for system reengineering, ranging from simple file managers (e.g. COBOL or RPG) to modern db management systems, such as the relational or object-oriented ones. For example, one might use the *Database Reverse Engineering Too*l [19] or *MatBase* [20].

SRE Ethics and Laws

Today, the vast majority of all *Fortune 500* companies have codes of ethics/conduct for helping guide of their employees' behavior. For example, *Texas Instruments* (TI), has the reputation of enforcing a longstanding such code, written and published in 1961 and regularly revised since to cope with business environment evolutions; moreover, its employees also receive a copy of the *TI Ethics Quick Test* [21] on a business-card-size mini-pamphlet to always carry with them, reading: "Is the action legal? Does it comply with our values? If you do it, will you feel bad? How will it look in the newspaper? If you know it is wrong, don't do it! If you're not sure, ask. Keep asking until you get an answer". The U.K. Institute of Business Ethics is publishing on its website [22] guidelines for content of business practice and ethics.

Such good examples abound, but, dually, as we all know, bad examples abound too, even in the U.S. (e.g. the IBM PC compatible industry starts, Intel vs. AMD, the Samba and Wine projects, etc.), not to mention China (nearly everything they manufacture is copied, from HMMWV cars, to fighter aircrafts, space rockets and vessels), Russia (e.g. their B-29 bomber copy re-baptized Tu-4, their AIM-9 Sidewinder missile copy re-baptized K-13/R-3S), Iran (their BGM-71 TOW missile copy re-baptized Toophan), etc. Generally, in the Western world learning from RE may be accepted, however, directly copying what you learned is considered as forgery and frowned upon, while in the Eastern one directly copying is rather a matter of national pride.

Legally, in the U.S. even for artifacts and processes protected by trade secrets, reverse-engineering them is generally lawful as long as it is obtained legitimately; however, SRE is generally a breach of contract and most EULAs specifically prohibit it; courts have found such contractual prohibitions to override the copyright law which expressly permits it (e.g. see Bowers vs. Bay-state Technologies). Sec. 103(f) of the 1996 *Digital Millennium Copyright Act* (DMCA) [23] states that if you legally obtain a protected program, you are allowed to reverse-engineer and circumvent the protection in order to achieve interoperability between computer programs (i.e., the ability to exchange and make use of information). Article 6 of the 1991 EU *Computer Programs Directive* [24] allows reverse engineering for the purposes of interoperability, but prohibits it for the purposes of creating a competing product, and also prohibits the public release of information obtained through SRE. In 2009, this Directive was superseded by a somewhat tougher version.

Legislation governing SRE is, for example, consolidated in [25].

[26] is an excellent paper not only on legal, but also on economic aspects of RE and, particularly, SRE.

Conclusion

RE is fundamentally about discovery and learning. Engineers learn the state of the art not just by reading publications, attending technical conferences, and working on projects, but also by reverse engineering interesting products. Better and/or cheaper products, as well as know-how advances come also from learning what has been done before. RE is generally a slower and more expensive way for information to spread through technical communities than patenting or publication, but it is nonetheless an effective and often invaluable source of information. RE is a form of dependent creation, but, if licit, this does not taint it, for in truth, all innovators, as Newton put it, "stand on the shoulders of giants", as well as on the shoulders of other incremental preceding innovators. Publication, patenting, and RE almost equally contribute to dissemination of know-how, which is the foundation of progress in science and utilitarian arts.

In the software engineering world, IBM was the founder of both open and commercial source code: their mainframe customers had always relied up to the early 1980s on the source code for reference in problem solving and to tailor, modify, and extend their OS products; then, IBM decided that it would no longer release to its customers the source code for its mainframe computer OS, which triggered the famous IBM user group Share that read: "If SOURCE is outlawed, only outlaws will have SOURCE" (a word play on a famous argument by opponents of gun-control laws). Applied to current software, this points out that hackers and developers of malicious code know too many techniques for deciphering others' software. Dually, it is trivially useful for the good guys to know these techniques, too.

As copying and distribution of digital products is essentially costless and almost instantaneous in today's digital networked environment, vulnerability of information products to market-destructive appropriations justifies some limitations on RE, but we should always keep in mind that RE is important to innovation and competition in all industrial contexts. Adapting intellectual property law so that it provides adequate, but not excessive, protection to innovations is a challenging task. Restrictions on RE ought to be imposed only if justified in terms of the specific characteristics of the industry, a specific threat to that industry, and the economic effects of the restriction. DMCA restrictions on RE propagated backwards and eroded longstanding rules, permitting RE in other legal regimes.

Even if RE and especially SRE are to some extent under legal fire, corresponding knowledge should be widespread. Especially when discussing specific RE features is illegal, we should discuss general approaches, so that it is within every motivated reverse engineer's ability to obtain information locked inside the black box. It is a pity that SRE is a computer science Cinderella, both in colleges, universities, and research labs, with so few students working on theses like [27]. It is a shame to almost abandon SRE, except for the anti-malware industry, in the hands of hackers and software pirates. I am proud to study, work, and teach in the SRE field too, to be the handling editor of this special issue of the Journal of Information Technology and Software Engineering dedicated to RE and to have written this Editorial paper for it. I know that this special issue too will contribute to pushing SRE forward. In my opinion, SRE should not continue to remain a computer science Cinderella; on the contrary, it should gain adequate, that is important status and the sooner, the better.

References

1. Lehman MM (1980) Programs, life cycles and laws of software evolution. Proc of IEEE 68: 1060–1076.
2. Chikofsky E, Cross J (1990) Reverse engineering and design recovery: A taxonomy. IEEE Software 7: 13–17.
3. Baxter ID, Mehlich M (1997) Reverse Engineering is Reverse Forward Engineering. Proc 4th Work Conf on Reverse Eng, Amsterdam, The Netherlands.
4. Muller HA, Jahnke JH, Smith DB, Storey MA, Tilley SR, et al. (2000) Reverse Engineering: A Roadmap. In: The Future of Software Engineering. ACM Press.
5. Zeltser L (2001-2012) Reverse Engineering Malware.
6. Eilam E (2005) Reversing: Secrets of Reverse Engineering, Wiley Publication.
7. Mancas C (2013) A completely algorithmic approach to data analysis and conceptual modeling, database design, implementation, and optimization. Apple Academic Press.
8. Hex rays, IDA disassembler.
9. Ollydbg debugger.
10. WinDbg debugger.
11. HxD-Freeware Hex Editor and Disk Editor.
12. CFF Explorer, NT Core.
13. LordPE.
14. Russinovich M, Cogswell B (2012) Process Monitor. Microsoft.
15. Process Hacker.
16. RegShot.
17. Linux source code, The Linux Kernel Archives.
18. MS Windows API doc. Windows API List (Windows).
19. Database Reverse Engineering Tool.
20. Mancas C, Mancas S (2006) MatBase Relational Import Subsystem. Proc IASTED Datab and Appl, Innsbruck 123-128.
21. Texas Instruments (1961) The Values and Ethics of TI.
22. Institute of Business Ethics (1986) Codes of Ethics.
23. US Congress (1996) 17 USC § 1201-Circumvention of copyright protection systems.
24. EU Council (1991) Directive 91/250/EEC of 14 May 1991 on the legal protection of computer programs.
25. Lambert M, Surhone, Mariam T, Tennoe, Susan F, et al. (2010) Software Law. Betascript Publishing.
26. Samuelson P, Scotchmer S (2002) The Law and Economics of Reverse Engineering.
27. Cipresso T (2009) Software reverse engineering education. M.Sc Theses, San Jose State University, 111 Yale Law Journal 1575.