COMPUTERWORLD

Reverse-Engineering

Mathew Schwartz

November 12, 2001 (Computerworld)

Whether it's rebuilding a car engine or diagramming a sentence, people can learn about many things simply by taking them apart and putting them back together again. That, in a nutshell, is the concept behind reverse-engineering—breaking something down in order to understand it, build a copy or improve it.

A process that was originally applied only to hardware, reverse-engineering is now applied to software, databases and even human DNA. Reverse-engineering is especially important with computer hardware and software. Programs are written in a language, say C++ or Java, that's understandable by other programmers. But to run on a computer, they have to be translated by another program, called a compiler, into the ones and zeros of machine language. Compiled code is incomprehensible to most programmers, but there are ways to convert machine code back to a more human-friendly format, including a software tool called a decompiler.

Reverse-engineering is used for many purposes: as a learning tool; as a way to make new, compatible products that are cheaper than what's currently on the market; for making software interoperate more effectively or to bridge data between different operating systems or databases; and to uncover the undocumented features of commercial products.

A famous example of reverse-engineering involves San Jose-based Phoenix Technologies Ltd., which in the mid-1980s wanted to produce a BIOS for PCs that would be compatible with the IBM PC's proprietary BIOS. (A BIOS is a program stored in firmware that's run when a PC starts up; see Technology QuickStudy, June 25.)

To protect against charges of having simply (and illegally) copied IBM's BIOS, Phoenix reverse-engineered it using what's called a "clean room," or "Chinese wall," approach. First, a team of engineers studied the IBM BIOS—about 8KB of code—and described everything it did as completely as possible without using or referencing any actual code. Then Phoenix brought in a second team of programmers who had no prior knowledge of the IBM BIOS and had never seen its code. Working only from the first team's functional specifications, the second team wrote a new BIOS that operated as specified.

The resulting Phoenix BIOS was different from the IBM code, but for all intents and purposes, it operated identically. Using the clean-room approach, even if some sections of code did happen to be identical, there was no copyright infringement. Phoenix began selling its BIOS to companies that then used it to create the first IBM-compatible PCs.

Other companies, such as Cyrix Corp. and Advanced Micro Devices Inc., have successfully reverse-engineered Intel Corp. microprocessors to make less-expensive Intel-compatible chips.

Few operating systems have been reverse-engineered. With their millions of lines of code—compared with the roughly 32KB of modern BIOSs—reverse-engineering them would be an expensive option.

But applications are ripe for reverse-engineering, since few software developers publish their

source code. Technically, an application programming interface (API) should make it easy for programs to work together, but experts say most APIs are so poorly written that third-party software makers have little choice but to reverse-engineer the programs with which they want their software to work, just to ensure compatibility.

Ethical Angles

Reverse-engineering can also expose security flaws and questionable privacy practices. For instance, reverse-engineering of Dallas-based Digital: Convergence Corp.'s CueCat scanning device revealed that each reader has a unique serial number that allows the device's maker to marry scanned codes with user registration data and thus track each user's habits in great detail—a previously unpublicized feature.

Recent legal moves backed by many large software and hardware makers, as well as the entertainment industry, are eroding companies' ability to do reverse-engineering.

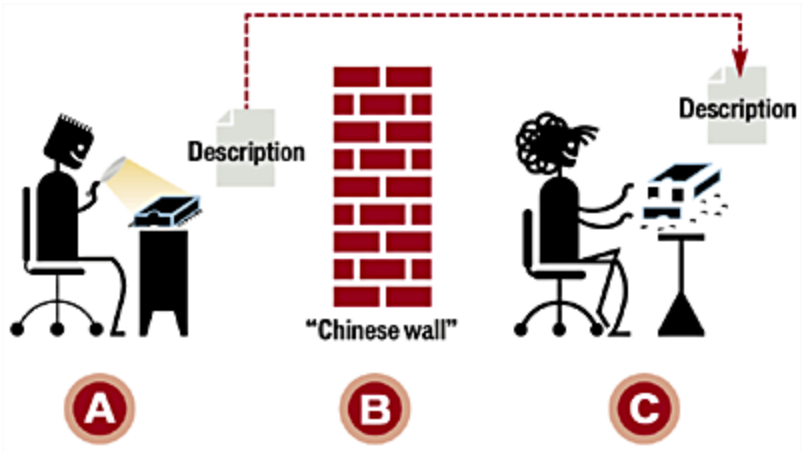
"Reverse-engineering is legal, but there are two main areas in which we're seeing threats to reverse-engineering," says Jennifer Granick, director of the law and technology clinic at Stanford Law School in Palo Alto, Calif. One threat, as yet untested in the courts, comes from shrink-wrap licenses that explicitly prohibit anyone who opens or uses the software from reverse-engineering it, she says.

The other threat is from the Digital Millennium Copyright Act (DMCA), which prohibits the creation or dissemination of tools or information that could be used to break technological safeguards that protect software from being copied. Last July, on the basis of this law, San Jose-based Adobe Systems Inc. asked the FBI to arrest Dmitry Sklyarov, a Russian programmer, when he was in the U.S. for a conference. Sklyarov had worked on software that cracked Adobe's e-book file encryption.

The fact is, even above-board reverse-engineering often requires breaking such safeguards, and the DMCA does allow reverse-engineering for compatibility purposes.

"But you're not allowed to see if the software does what it's supposed to do," says Granick, nor can you look at it for purposes of scientific inquiry. She offers an analogy: "You have a car, but you're not allowed to open the hood."

The Clean-Room Approach To Reverse-Engineering



One person or group takes a device apart and describes what it does in as much detail as possible at a higher level of abstraction than the specific code.	That description is then given to another group or person who has absolutely no knowledge of the specific device in question.	This second party then builds a new device based on the description. The end result is a new device that works identically to the original but was created without any possibility of specifically copying the original.

Schwartz is a freelance writer in Arlington, Mass. Contact him at Mat@PenandCamera.com.