Chris Verhoef

The dark side of the Millennium Bug

Chris Verhoef is an information scientist who also specializes in reverse engineering, the theory and practice of recovering information from existing software and systems. He chaired, with others, the Fourth IEEE Computer Society Working Conference on Reverse Engineering, the premier research-oriented conference dedicated to these problems.

"A troubling aspect of the Y2K problem is the fact that many people regard the problem as trivial. If one sees it as simply converting two digits into four, it does seem simple. But, this is a very narrow view of the Y2K problem."

"The crux of the Year 2000 problem is that it is a very pervasive and omnipresent problem. The usual definition of this problem concentrates on the fact that software has to be changed in order to widen two digit years into four digit years. However, if the databases are huge, this is impossible. Secondly, the problem is present in operating systems, so these need to be updated. This means that the assembler code which is geared towards these operating systems needs to be changed. In typical mainframe environments, this also implies that the old COBOL dialects need to be updated, e.g., COBOL 74 must be migrated to COBOL 85 dialects. Of course, also the two digit dates need to be 'windowed' so that correct calculations can be made."

"Another aspect of Y2K problem is that 2000 is a leap year. We know of real-world examples that do have 4 digit dates, but are still not Y2K compliant. And this all has to be done simultaneously with the Euro problem, which is also a very difficult one. In short, there is a lot of work to be done to get this problem solved and it is not simply a question of screening some noncompliant code."

"People have waited far too long to attack the renovation problems. Our approach to this problem is to carry out research in order to facilitate high speed automation processes for solving this problem. The Year 2000 problem resides in about 500 computer languages. There are search engines available for only 40 languages, and remediation engines for 10. About 30 % of the world's software is written in COBOL, 10 % is C , 10 % is C++ and 10 % is Assembler. The rest is written in more than 500 often very obscure languages. To make things worse, computer languages also have dialects. For COBOL there are more dialects than you can imagine, including homebrewed ones. Since we are dealing with systems in the range of 1 - 50 MLOC (millions of lines of code) all manual approaches break down. Therefore, the generation of analysis factories and software renovation factories is necessary since as much automation as possible is necessary."

"In other words, what we observe is a demand for:
- Very sophisticated parsing technology
- Very sophisticated data and control flow analysis technology - Very sophisticated computer aided language engineering
- Very sophisticated generic language technology

- Very sophisticated component-based technology
- Very sophisticated pattern recognition technology"

"A crucial part of solving the problem in an automated way, is to parse the code, like in a compiler. Therefore a grammar is necessary. We built a factory where we can extract the grammar from the compiler source code. If the compiler is designed carefully, this means that extracting a grammar is only about a ten minute job. Of course, when the grammar is deeply hidden in the compiler this is more work. Since so many languages are involved (500+) it is no longer possible to think in monolithic architectures for Year 2000 repair engines. This implies that for every new language we can effortlessly connect the parser to the existing Year 2000 repair engine. It is necessary to have a completely open architecture so that "any" component can be effortlessly connected to an existing tool, or that components can be switched for newer ones."

"The physical appearance of a typical software renovation factory is a multi-processor machine with enormous amounts of internal memory and giant disks for storing the old system, the new system and their abstract syntax representations. Distributed component technology is used to distribute the computational expensive process of renovation to its processors. This technology enables one to switch from one-shift year-2000 repairs to 24-hour around-the-clock repairs by establishing a global network of year 2000 repair factories, with three year 2000 repair facilities located eight time zones apart."

Sponsored by: