INTEROPERATE WR2QTP

Interoperate's Migration Technology: An Interview with CTO, Dr. Gopal Gupta

INTERVIEWER
What are legacy software systems?

DR. GUPTA
A legacy software system is an obsolete, old software program that continues to be used. The commonly heard term "legacy code" is used to refer to the code of such software systems. While hardware is easy to modernize since it can be redesigned to run the same software, software is much harder to modernize, as that would mean rewriting it. That's why one hears CIOs complaining about legacy software but not about legacy hardware.

INTERVIEWER
What are the problems associated with legacy software systems?

DR. GUPTA
Legacy systems lead to increased cost of maintenance. According to Gartner Research, at most organizations, close to 70% of the IT budget is spent on software maintenance. Lot of this is due to legacy systems. Legacy systems are costly to maintain as it is difficult to find programmers who can modify them to adapt to newer business requirements, etc. Thus, businesses get locked in to their old business logic which leads to erosion of competitiveness and business agility.
A software system can become "legacy" due to the environment/OS it is running under going out of use (e.g., Multics, IBM OS/360) or evolving significantly (e.g., Berkeley Unix). Or, the language the software is written in is no longer in active use (e.g., PL/1, Pascal, Cobol) or has been substantially revised (e.g., Fortran 77). In many cases software can become "legacy" simply because its vendor has stopped supporting it (e.g., Winrunner TSL) or it simply just lost its market-share (Powerbuilder).

INTERVIEWER
What can businesses do to avoid the scourge of legacy software systems?

DR. GUPTA
Businesses can migrate their software to modern languages, operating systems and environments. Thus, COBOL code can be migrated to C++ or Java, applications running on Berkeley Unix to Linux, Winrunner TSL scripts to QTP scripts, etc.

INTERVIEWER
How does a company go about migrating software?

DR. GUPTA
Now you come to the crux of the problem. Traditional approaches to software migration have been either too error prone or costly or both. Interoperate's approach on the other hand is both inexpensive and reliable. It also allows for migration to be achieved in a few months.

INTERVIEWER
What are the traditional approaches to migration and why are they costly or unreliable?

DR. GUPTA
Traditional approaches are based on converting the code manually or building a translator (a compiler-like program) that will translate the code automatically to its modern version. The approach based on manual conversion is costly and error prone. An army of programmers has to be employed. Programmers have to look at each line of code, understand it, and then rewrite it in the modern language or for the modern environment. Because of human involvement during conversion, the process is prone to errors: subtle mistakes are inadvertently introduced into the code that may be hard to discover.
Mistakes and bugs introduced during conversion will continue to be discovered after field deployment. What is worse is that many times the compounding effect of these programmer-induced mistakes is so much that the translation project is abandoned. The manual-migration landscape is littered with such abandoned projects. The costs can especially be enormous: assuming an optimistic estimate of 20,000 lines of conversion per programmer per year, one million lines of code will take 50 man years to convert. Thus, the manual conversion approach is costly, unreliable, and prone to failure.

INTERVIEWER
Then obviously an automated approach that builds a translator is better…?

DR. GUPTA
Absolutely. However, a translator developed using a traditional approach has its own problems. A translator is a compiler-like program that parses the legacy code, and generates target-code in the target modern language. The development of this translator is aided by parser- generator tools such as Lex, Yacc, Bison, Antlr, etc. However, developing these translators is a complex undertaking. Despite the availability of parser-generators, the backend of the translator coded in Java or C/C++, swells up to tens of thousands of lines of code. Coupled with the fact that the translation process involves a complex source language and a complex target language, very soon the translator code becomes unwieldy. As this code becomes unwieldy, bug-fixing in the translator takes longer and longer. In most instances the complexity becomes so high that the project is abandoned. In any case the cost of developing a translator can be high: about 2 to 3 man years. The advantage of a translator-based approach is that, once the translator is built for a particular pair of languages, it can be applied again and again. The output will be free of errors, or if there are bugs, then they can be removed gracefully. That is, once a bug is eliminated in the translator, it is removed once for all from the generated code (contrast this to the manual approach where discovery and fixing of a bug in the generated code doesn't mean that all instances of the bug have been removed, each instance has to be removed manually).

INTERVIEWER
How is Interoperate's approach any better then?

DR. GUPTA
Interoperate also adopts a translation based approach, however, the translator is developed using a unique methodology that I have developed through my research over the last 10 years. This methodology views the translation problem as the problem of relating the semantics of the source language to that of the target language. The translator is a program that transforms the syntax of the source language to that of the target language while preserving the semantics. The front- as well as the back-end of the translator is written using a high-level, rule-based approach. As a result, Interoperate's translators are only a few thousands lines long. Thus, they are easy and quick to develop; bugs can also be fixed with the same ease and quickness. Using its semantics-based approach, Interoperate can develop a translation in 4 to 6 man months.

INTERVIEWER
All this may work well in theory, Dr. Gupta, but have you really applied it in practice?

DR. GUPTA
Indeed I have used this technology to develop translators in situations where the prevailing belief was that a solution is not possible. An important feature of our technology is that it can even handle languages with *context-sensitive features*. We used it to build a translator for converting Braille-based code for Mathematics called Nemeth code to a print format so sighted people can read what a blind person has written in Braille. This translation problem was thought to be unsolvable; we built the system in 4 months. We also used our technology to process Nexus, a language for specifying bioinformatics systems, which allowed bioinformatics tools to become interoperable with each other. A language processor for the complete Nexus language did not exist due to its context sensitive features.

INTERVIEWER
What about commercial applications?

DR. GUPTA
We have developed a translator for converting Winrunner TSL scripts to HP's QTP scripts. Winrunner scripts can thus be translated and made to run under HP's QTP system, providing a pathway to users of Winrunner out of obsolescence. This translation is done with close to 100% automation (minor manual intervention is required to adjust the object map files generated for QTP). We have also developed translators to translate Winrunner TSL to other GUI-testing products.

INTERVIEWER
In most migration projects, the code being migrated is invariably re-architected for one reason or another. Can Interoperate's technology be used for re-architecting the code?

DR. GUPTA
The re-architecting is invariably done when the manual approach is used. As programmers understand the code and its business logic during the migration process, they naturally rewrite it in the target language in a more optimal fashion. This can also be achieved with our technology, however, one must be able to clearly state the transformation rules that must be followed for re-architecting. These transformation rules can then be incorporated into the translator and re-architecting done at the same time.

INTERVIEWER
Are there other applications of this technology?

DR. GUPTA
The translation technology can also be used for porting software, for example, porting an application written in the Unix BSD environment to Linux. Or porting programs from one version of a software system to another. It can also be used for re-architecting code: in such a situation our technology is being used for program transformation.

INTERVIEWER
What about the quality of code produced? Can the generated code be understood, modified, and maintained by a programmer?

DR. GUPTA
The quality of the code generated is very good. Our experience indicates that the code generated is understandable. We can even transfer the comments in the original program to the corresponding points in the generated target code. A programmer can thus understand the code generated in the target language, and further modify/maintain it as needed.

INTERVIEWER
Final question, what is Interoperate's future plan?

DR. GUPTA
Interoperate is currently focused on migrating scripts written for Winrunner TSL. It has already built a translator for converting Winrunner TSL scripts to HP's QTP as well as for other vendor's products. Interoperate's future plans include leveraging its translation technology to perform migrations in other areas such as migrating legacy mainframe software to more modern platforms.