

Converting Legacy Applications

General remarks

Many programs use old programming languages such as Cobol, PL/1, Adabas Natural. These *Legacy* systems are very important to business. Legacy programs are working, but because they have been modified during many years and the documentation is limited, nobody knows how and why they work.

Support and maintenance of such programs is very expensive, they are difficult to integrate with other applications and can't be web-enabled. Thus there are two main tasks:

- Reverse engineering: recovery of lost program text knowledge, of the module structure, of the connections between modules. Dead-code deleting.
- Re-engineering: transferring the applications to other platforms e.g. Java. Re-engineering is not just a compilation. It is much more difficult:
 - Platform shift. Databases must be changed into relational or object oriented databases. Applications must work under application servers
 - Code re-factoring. Instead of a spaghetti structure, we convert to a tree structure. Otherwise the new code will be less maintainable than the original legacy system.

Because of the sophisticated mathematical and algorithmic solutions developed by the company **Lanit - Tercom** at St. Petersburg State University, 80 to 90 % of the work can be done automatically.

Therefore, Lanit- Tercom's method is faster and cheaper than other solutions. About 10-20 % needs to be done manually. The exact volume depends on the particular application and the amount of architectural changes.

The conversion is done in four steps.

Step 1

1. Inventory: do we have all the files?
 2. Are there references to files that do not exist?
 3. Are there files with no reference?
 4. Measurement of complexity. Every application can be represented as a graph with nodes, main operators and edges as well as connections between operators. There are international standards for measurements (number of modules, number of source code lines, number of paragraphs, number of conditions etc.)
 5. Estimation of efforts based on complexity measurements.
- A pilot project is used to estimate the efforts. It should be a subsystem of 10-20 programs that constitutes a good representative of the application with screens and databases. It will also give a good understanding of what the output will look like; how big manual efforts are needed and whether any tuning (customization) of the tools is needed.

Step 2

Prepare improvements inside the source language without doing any conversion:

1. Dead-code elimination. Often about 30 % of the code is dead-code that can be eliminated. It is easier to re-engineer only 70 %.
2. Procedure restructuring. Cobol has no procedures but paragraphs. Paragraphs can be performed ("called") one by one or in sequences. Sequences can be overlapping. PL/1 procedures can have many entry points, making maintenance difficult. The source Cobol or PL/1 program will be converted into another Cobol or PL/1 program but without overlapping paragraphs.

3. Reduce the number of "go-to", replacing them with structural constructs - PERFORM, EVALUATE. Every program can be mathematically represented without any "go-to" at all, but this is not practical. Normal reduction of "go-to" will be about 90 %.
4. The goal is to prepare a new Cobol or PL/1 program but with strict procedural structure, to achieve a program that is similar to the source but two to three times shorter, structured, without repeats and deadlines and much more readable and simpler to maintain.

Step 3

Usually legacy applications can have something like one hundred functions and several million lines of source code. Often some parts of the functions are not interesting any more, whereas others are crucially important for business. If we can delete dead functions, the application will become shorter and easier to maintain. We can propose several strategies for such business rule instructions:

- **From left to right**

Imagine that some variable has four different possible values. The User may know that only one of them is possible and the others are negligible. We can prepare a partial evaluation that means we shall substitute this value instead of the variable in all the lines of the application. For example, if this variable was used in some switch to select one of the possible calculation directions, we shall select only one direction and omit all others.

- **From right to left**

Computational extraction - split one large program into series of smaller ones, each performing one particular business function. Imagine we are interested in some functions. First we will find the very last operator producing the value of the function. Then we will go back and select only those operators that have an impact to this selected operator. It can be done mathematically correct.

- **Structural Business Rule Extraction**

Let the user know the main paragraph of some important function. This paragraph can use some variables from other paragraphs and can call other paragraphs. We want to create an independent program (subroutine) based on this paragraph, which can be used (CALLED) as an independent entity by other programs.

Business Rules Extraction is a tool. To perform it we need a close co-operation with the customer. The customer can try several variants based on his different ideas. This is really the most powerful tool for Knowledge Mining. Lanit-Tercom's specialist can come to Denmark for say two weeks or a customer specialist can come to St. Petersburg.

Step 4

The skeleton of the final system must be done manually. Only the User knows which environment he wants to use and how to distribute actions among clients and servers etc.

Business Rules are the real asset of the old program. They can be transformed almost automatically and the entire system can be assembled from these parts. Because each component becomes separately callable, you accelerate the path to new opportunities like Web Services.

Finally, the transformation to Java, C++, Visual basic, HTML or similar takes place.