Going from `PIC 99` to `PIC 999`

—Extended Abstract—

Steven Klusener[1] Ralf Lämmel[2,3]
Chris Verhoef[3]

1 Software Improvement Group, Amsterdam
2 Centrum voor Wiskunde en Informatica, Amsterdam
3 Vrije Universiteit, Amsterdam

Yet another Y2K-like problem We discuss a case study in software renovation. The case is about a range overflow problem (like the Y2K problem). Basically, the problem seems to be a very simple one, and it can be formulated as follows: Data items of a certain kind—let us call them product codes—had to be expanded. The original software application (programmed in COBOL with embedded SQL for DB2 data management) was developed under the assumption that 99 different product codes are sufficient, and thus, the data type `PIC 99` was used for these fields, as in the following declaration:

```
01 PRODCODE PIC 99.
```

In the life-cycle of the software the range of product codes was about to see a range overflow (because the 99 product codes were almost exhausted). The new requirement became that the software had to cope with 299 different product codes. Thus, the new data type `PIC 999` would be appropriate, and the above line of code had to be transformed as follows:

```
01 PRODCODE PIC 999.
```

We did this project for a large bank. The application at hand consisted of about 100 programs with 100.000 lines of code. We developed a semi-automatic solution to the problem.

Not such a little problem The formulation of the problem might suggest that the problem was trivial. Note actually that the Y2K problem can be stated in such simple terms, too, if we only abstracted from most complications. It actually turned out that the problem was about much more than identification of product codes, and expansion of simple picture mask conversion. Let us just indicate some complications. Firstly, the actual identification process needs to be made precise as we cannot rely on the fact that product codes are always fields with name `PRODCODE`. Secondly, picture mask expansion is just one conversion rule. We also have to expand tables and literals. Again, precision is needed here because we might invalidate the software if we miss infected patterns (false negatives), or if we are too offensive (false positives). There are more of these technical complications. Some fields of product codes were used with a different type than `PIC 99`, namely longer fields or alpha-numeric fields, triggering all kinds of special conversion problems. As a kind of extreme example serves the following piece of real code which defines a memory area to store the indices of product codes:

```
01  PRODKODES-PRIJSTABEL.
    03 FILLER                    PIC X(40)
    VALUE '0102030405060708091011121314151617181920'.
    03 FILLER                    PIC X(40)
    VALUE '2122232425262728293031323334353637383940'.
    03 FILLER                    PIC X(40)
    VALUE '4142434445464748495051525354555657585960'.
    03 FILLER                    PIC X(40)
    VALUE '6162636465666768697071727374757677787980'.
    03 FILLER                    PIC X(38)
    VALUE '81828384858687888990919293949596979899'.
01  PRODKODE-TABEL  REDEFINES  PRODKODES-PRIJSTABEL.
    03  PRODKODE  OCCURS 99      PIC X(02).
```

There are also numerous process characteristics one has to take into account, e.g., the incompleteness of available sources, a fixed price requirement, and the need for layout preservation.

Contribution We discuss all relevant details of the Y2K-like project, namely details concerning requirements, problem specification, design and implementation. Thereby, we supply a well-documented case study in software renovation. The reported experience should be of value for many other mass maintenance projects, in particular, to setup a software process, to figure out the effort, to decide on suitable means for design and implementation. One should not get distracted by the rather small size of the code base underlying the reported project. We have been involved in larger projects, and we see our conclusions approved in those. We have chosen this not too large project for the sake of a self-contained discussion, so that a complete picture is drawn.

Approach We first analyse the problem to define a more detailed problem specification (as opposed to the simple statement to expand product codes by one digit), and we explain how code exploration and others are used to define the effort for the project. In this state, we stick basically to the following problem specification process:

1. Do code exploration to learn about the problem.
2. Think of an operational model to address the scenarios at hand.
3. Make an estimation for effort and cost.
4. Find out about the limitations of the current model.
5. Discuss the results from 1.-4. with experts on the client site.
6. Repeat 1.-5. until you get confident in your estimation.

Then, we systematically design rules for analysis and transformation so that we can later implement an automatic or semi-automatic conversion program. Like in most software process models, the problem specification and the design phase are somewhat intertwined. Defining the rules, we might also communicate with the client, and redefine the problem specification. Based on the design, we describe the actual implementation of the rules, and we also discuss surrounding issues such as documentation and testing. The overall idea here is to provide a rational industry strength solution. We explain the actual approach taken, and the reasons (say project constraints) for this choice. We conclude on the lessons learned, alternative implementations, and on related work.