Software Quality Journal, 12, 297–309, 2004

9210: The Zip Code of Another IT-Soap *

A.S. KLUSENER steven@cs.vu.nl
Software Improvement Group, Amsterdam, The Netherlands and
Free University of Amsterdam, Department of Computer Science, Amsterdam, The Netherlands

C. VERHOEF x@cs.vu.nl
Free University of Amsterdam, Department of Computer Science, Amsterdam, The Netherlands

Abstract. Nine-to-ten (9210) refers to the problem that the Dutch banks are running out of 9-digit bank account numbers and need to convert to 10-digit numbers. At the same time, the Dutch government wants bank account numbers to be portable to encourage competition; this may become European policy. A recent European standard for cross-border money transfers proposes totally nonportable bank account numbers. These orthogonal policies have such a high IT-soap caliber that we sometimes refer to it as *9210 Policy Nils*. Whatever the plot of "nine-two-one-o," major challenges are at stake for European banks and other "number crunchers" like tax authorities, mail-order firms, etc. This paper gives insight in cost aspects, the possibilities, and impossibilities of 9210 and related problems.

Keywords: software pasteurization, 9210-problem, bank account number portability, international bank account number (IBAN), software cost estimation, IT-portfolio analysis, automated program transformation, IT-portfolio transformation, IT-portfolio management

1. Introduction

Data types that were once envisioned never to have certain properties, cause severe problems when the need for these unanticipated properties nonetheless arises. Think of return codes that were never envisioned to be negative. Out of the blue they will become negative due to a new release of some software product ranging from a screen generator, a new compiler, to a database product. If your implementation relies entirely on positive numbers, you have a problem. Or think of a Customer-Relationship Management system that was never envisioned to deal with more than a million customers. After a merge you pass the magic line and you can no longer store knowledge about all your customers. Which is unacceptable. A hundred different financial products was unthought of in the 1960s, but after several mergers and acquisitions the company has grown and at short notice you need to monitor hundreds of financial products. But you can't.

Or constants you were sure would never change, behave unexpectedly: they *vary*! Think of the recently abandoned European currencies, but also error codes, zip codes, phone numbers, bank account numbers, addresses, names, and what have you. For instance, some European banks used the currency as the primary key for certain cross-border money transfers. So, transferring 100 DM implied that it should go to the

* Contract/grant sponsor: This research has partially been sponsored by the Dutch Ministry of Economic Affairs via contract SENTER-TSIT3018 *CALCE: Computer-Aided Life Cycle Enabling of Software Assets*.

German office. Likewise, 100 NLG would go to the Dutch office. But when the Euro was introduced, it was no longer possible to use this method, and although the systems were thought to be Euro-compliant, these banks had to transfer up to 30% of their international transactions manually for some months before this was repaired.

1.1. Software pasteurization

Solving such problems is nothing else than extending the best-before-date of the software assets. The Year 2000 problem was not just a data type no longer simulating reality accurately, but also that the best-before-date of all these systems coincided. The latter problem is solved, but the repaired systems still have a best-before-date, that will be due somewhere in the future. Some people think this is in 2050, or 2038, but it happens every day. Let us give an example of RPG-code that needs an update at the time of writing this paper.

```
H*** 28-01-98
H* THIS Y2K-FIX GOES WRONG WHEN A DATENUMBER IS
H* GREATER THAN 40000 (= 06-06-2003)!!!
CSR    N18N56          SETON                  2489
CSR     18 56IDAYLA    COMP VDAYLA            561818
CSR     18   IDAYLA    SUB  ARVDDD    DGN    24H1
CSR     56   VDAYLA    SUB  ARVDDD    DGN    24H1
CSR     02 56DORCLA    COMP ASDORC            1919
```

Mass-maintenance projects are occurring every day, and their costs are substantial. They all deal with extending the best-before-date of business-critical software assets, thereby capitalizing on existing systems to deploy and exploit them to their full extent. We call this *(software) pasteurization*: preventing the software from turning sour by prolonging its best-before-date much like Louis Pasteur's method to prolong the due-date of milk by partial sterilization that destroys objectionable organisms without *changing the functionality* of the milk. Pasteurization not necessarily implies a date-related change like Y2K, but any effort that enables software assets to keep functioning properly by removing approaching show-stoppers that otherwise render them useless (Klusener et al., 2004).

1.2. The 9210-problem

The Dutch financial industry is facing a significant pasteurization effort: solving the 9210-problem. Most Dutch banks now use 9-digit bank account numbers, except one bank that uses serial numbers starting with 1 (the account number of the Dutch National Treasury), up to 7 digits, which complicates matters further. We analyzed several IT-portfolios of large international banks. Our analyses revealed that low percentages of their source listings contain only 9-digit bank account numbers, which seems to indicate the problem is small. This is not true: the low percentages are spread over the majority of the IT-systems. We carried out a benchmark project, showing that solving

the 9210-problem using automated system modification tools can reduce the cost of change by at least a factor 4 (Klusener et al., 2004). We cannot reduce the cost of release without taking risks. Still, using mass-update 9210-tools, the total cost savings are in the order of 30–40%.

The goal of this paper is to give insight in 9210-project costs, with and without using tools. Moreover, we will put 9210 in the context of regulatory changes like number portability, and international/domestic bank account number standards. We hope the paper adds to a better mutual understanding between policy makers, the financial industry, and their customers.

2. The cost of change and release

The authors have been commissioned to pasteurize significant amounts of software in several roles: advisory roles, estimating impacts, building and deploying supporting tools, doing entire projects, etc. In addition, the authors have analyzed large IT-portfolios for several reasons: Y2K-impact analyses, Euro-impact analyses, 9210-impact analyses, and financial analyses (Verhoef, 2002, 2004a, 2004b). We have experience with the technical nuts and bolts of changing entire IT-portfolios plus their financials. Moreover we have seen several 9210 cost estimates during economical analyses of IT-portfolios (Verhoef, 2002). Without revealing sensitive data, it is possible to give you candid insight in these costs, and based on our benchmarks, we can indicate how to reduce them.

2.1. Setting the stage

Large international banks possess according to public benchmarks 450,000 function points of software (Jones, 1996, p. 51). A function point (Albrecht, 1979; Jones, 1996) is a synthetic metric giving an indication of the size of software. If we suppose that all the code of such a large international bank is written in Cobol, we multiply the amount of function points by 107, to get an impression of the amount of logical statements: a little over 48 million statements. This is in accord with our experience: indeed we analyzed IT-portfolios of 50 million physical lines of Cobol code and beyond of large international banks to measure the impact of 9210.

2.2. Cost of change

A rule of thumb to estimate the cost of outsourcing the Y2K-problem was to use between $1.00 and $2.50 per physical line of code (Jones, 1998b, p. 211) and for Euro-conversion this was between $1.15 and $2.75 (Jones, 1998b, p. 213). We note that the estimates for making such changes in-house are lower: between $0.25 and $1.00 for Y2K (Jones, 1998b, p. 211) and $0.50 and $1.25 for Euro-conversion (Jones, 1998b, p. 312). From several portfolio analyses (Verhoef, 2002) we learned that these rules of thumb were transposed to 9210, and used as a first estimate for the cost of change. The one that we have seen most is $1.00 per physical line of code. According to this rule of

thumb, a typical large international bank will at least spend in the order of 50 million dollars on 9210 for the cost of change alone.

2.3. Cost of release

The 50 million excludes the release costs of the pasteurized systems. Often decision makers and policy makers think that if only a few percent of the code is infected the total cost will be low. But, release costs can range from very high if all systems are infected to very low if all the problems are concentrated in a few systems. You will only know this after a 9210-impact analysis.

The release of a system comprises shipping, compiling, linking, testing, and putting it into production. Complete release costs can vary substantially per system. Simple single-site releases of business-critical systems easily cost 20 person days, which amounts to $20,000 when you take a daily fully burdened rate of $1000. A complex multi-site release of a business-critical system like a global transaction and settlement system can cost you easily between 4 and 6 person years which is $0.8–1.2 million release costs, assuming 200 working days per year (Faust and Verhoef, 2003). Migrating an entire IT-portfolio of a large international bank to a Y2K-compliant operating system costed between $60 and $100 million. So, IT-portfolio release costs are in the same order of magnitude as IT-portfolio change costs. Normalizing these data for a large international bank owning 450,000 function points, we found $1.00 release costs per physical line of code.

2.4. Major findings

From several 9210-impact analyses we found the following patterns:

- only between 2 and 8% percent of the source listings contains 9-digit numbers;
- these 9-digit numbers are spread over 75–100% of the IT-systems.

This implies not only substantial change costs, but also very high release costs. Thus, the assumption that the costs are low because only a few percent of the IT-portfolio contains 9-digit bank account numbers, is not in accordance with the empirical data that we collected. Summarizing, a large international bank owning about 450,000 function points of software assets will spend in the order of a hundred million dollar to solve 9210. Next we explain how these costs can be reduced.

3. Reducing the costs

To estimate potential cost reductions, it would be ideal to carry out a small 9210-project. But, in order to test the outcome in vivo, it is very hard to change only a small part of the IT-portfolio and solve 9210 in that part. This is partly due to the spread of the problem over the entire IT-portfolio, but also since it is hardly possible to test partial solutions since many IT-systems are interconnected within an IT-portfolio.

3.1. Benchmark project

As we already mentioned, these problems occur frequently, and we found a sufficiently similar benchmark project. It is called the 223-project: going from two to three digits. We carried it out, tested it in vivo, accurately measured the changes, and published all its aspects, including cost issues (Klusener et al., 2004). In a nutshell, the 223-project was about migrating a small (100,000 LOC) business-critical system that contained 2-digit product codes to a system that could work with more than a hundred product codes (3-digit numbers). This problem resembles the 9210-problem, although 9210 is more complex. For, a bank account number contains separator periods, satisfies validation algorithms, and more, unlike a 2-digit product code. Still for a 9210-cost reduction estimate enough similarities were found between 223 and 9210. For instance in the 223-benchmark, we identified about every 1000 lines of code another type declaration for the same 2-digit product code field. Similarly, bank account numbers come in many flavors. Thus, like the 223-problem, the 9210-problem is not simply changing a single data type: there are many different data types representing the same bank account number. Another additional complexity is that some banks use 10-digit saving account numbers. These saving account numbers will interfere with new 10-digit bank account numbers. Only a 9210-impact analysis will reveal such things. The technical aspects of the detailed analysis that we used for 9210 are beyond the scope of this paper and elaborately treated elsewhere for the 223-benchmark (Klusener et al., 2004).

3.2. Reduction factor

Using the rule of thumb that this type of change costs about $1.00 per line of code, an initial estimate for 223 was made. The cost of change for 223 should at least be in the order of $100,000 (ex release costs). The actual total cost of change for the 223-benchmark was about $25,000. This included the development of 223-tools to detect and change the infected code and their use to make the actual changes. Based on detailed knowledge on what was changed, how much was changed, and where it was changed, we could validate the rule of thumb: indeed at least $1.00 per LOC was necessary to make the changes, if it were done manually. For this we assumed that not a single error would be made if all changes were done without using automated support. Thus, since the tool-supported change costed $25,000, we saved a factor 4 on the cost of change by using 223-tools. This factor is not only conservative because we assumed no manual errors, but also since we did not used the actual *amount* of changes but the number of different *types* of change, to maximize for the learning curve effect (doing the same change often). We identified 48 types of changes, and used that number to base our calculations on. Note that in the 223-benchmark, there were 597 actual changes, which is a factor 12+ more than our 48. Using the 48, we calculated that a manual pasteurizing effort would take at least 100 person-days. The fully burdened cost of a software engineer is about $1000 per day or $100,000 for the entire project. This is in the same order of magnitude as the rule of thumb. Another indication showing our factor 4 is conservative is that for Y2K and Euro repairs, Jones estimates that you can repair 15 function points per staff month (Jones, 1998b, p. 600).

And since a 100,000 LOC system is about a 1000 function points, the effort should be around the 66 staff months. A 100 person-day project for one person takes about 5 staff months. Nevertheless, we stick to our factor 4, which stems from actual measurements. This is unknown for Jones' estimate. Even with our reduction factor, the cost savings are substantial.

3.3. Reduced cost for 9210

Based on the very accurate and detailed information for the 223-benchmark, we inferred this rather conservative factor 4, so when we use the same factor for the 9210-problem, this will also save at least a factor 4 on the cost of change. For a large international bank, owning 450,000 function points of software, we already estimated the cost of manual change on $50 million, and the release costs at $50 million as well (due to the spread of the 9-digit numbers over the majority of the systems). Saving a factor 4 on the cost of change, will lead to a cost of change in the order of $12.5 million. So the total cost of a 9210-project for a large international bank amounts then to $62.5 million. This leads to a cost reduction of at least 37.5% for a 9210-project. You could lower release costs, by using partial testing: when no errors occur after a representative amount of systems, put the other systems in production without testing. We do not recommend this. To calculate *precise* costs we recommend a 9210-impact analysis plus a pricing process as we described for 223 (Klusener et al., 2004).

3.4. Silver bullet vaccination

We realize that according to Brooks' law—*there is no silver bullet*—you cannot improve productivity by an order of magnitude using a single technology (Brooks, 1995). In the case of 9210, the change effort is systematic and rule-based so that automation becomes feasible, which causes the order of magnitude productivity improvements. Our findings are in compliance with Gartner Group, who stated (Mieritz, 2002):

> These "endowed" systems can often yield a 10 percent to 50 percent efficiency improvement with the application of best practices; technology refresh (a cost in itself, but often with a rapid payback), standardization, consolidation and elimination of redundant systems.

Indeed deploying innovative technology to consolidate an IT-portfolio by solving the 9210-problem leads to efficiency improvements in the same order of magnitude as indicated by Gartner Group.

4. The cabala of numbers

Bank account numbers (BANs) are not easy, and the more you know about them, the more it looks like an esoteric doctrine or a mysterious art: the cabala of numbers. In banking databases almost only 10-digit BANs are present making many believe that there is no problem. Interpay, the responsible authority in the Netherlands for issuing

Table 1. Distribution of different types of BANs over source listings.

Type of BAN	Percentage
No BANs	50–70%
10-digit BANs only	10–20%
All kinds of BANs	15–25%
Only 9-digit BANs	2–8%

bank account numbers, released what they call the 0-series: 10-digit BANs with a leading zero. The leading zero is exploited by programmers in their code. Sometimes the leading zero is treated as a space, sometimes as a zero, sometimes it is thrown away when a 10-digit number is fetched from a 10-digit compliant database or entered by a user, sometimes the "free space" is reserved for a special token, or there are n-digit numbers from the start with $n \neq 10$.

We summarized the cabalistic situation in Table 1. An analysis of the data types used for BANs in several IT-portfolios revealed that between 50 and 70% of the source listings did not contain any kind of bank account number. In 10–20% of the listings, 10-digit BANs are used. This does not imply that the *semantics* of these 10-digit BANs is according to specification. Often the leading zero is dropped as soon as possible, or the leading zero is not taken into account in the checksum algorithm, and so on. Then between 15 and 25% is using all kinds of formats. This ranges from internal BANs that are not according to specification (starting with special prefixes like `99`), to n-digit BANs, for $n = 4, \ldots, 14$. This category also contains data type definitions containing hard-wired BANs in their Cobol value clause. In addition to numerical BANs, we also encountered alphanumerical varieties, while according to the European Committee for Banking Standards, numerical fields are only allowed (ECBS, 2002). Finally, between 2 and 8% of the source listings contain 9-digit BANs only. This percentage is relatively low, and indeed there are almost no citable quotations showing that 9-digit BANs in the Netherlands exist. An exception is a technical report of the European Committee on Banking Standards, where the specification for the Dutch BANs is specified as follows: $9/10n$, meaning minimally nine and maximally ten numeric characters (ECBS, 2002, p. 61). In addition to this standard, one Dutch bank is using another standard: BANs ranging from 1 to 7 digits. The $9/10n$ BANs use a check, which is absent for the 1–7 length BANs. Additionally, there is a multitude of data types, hard-coded BANs, numerical and alphanumerical fields all used for the single notion bank account number. This is in accordance with the 223-benchmark project, where we found in a 100,000 LOC Cobol system, every 1100 lines of code a different data type for the same two-digit product code (Klusener et al., 2004).

4.1. Standards

Our colleague Andy Tanenbaum once said that the good thing about standards is that there are so many to choose from. The occult world of bank account numbers is no exception to this truism. For a start, the Dutch banks need to implement a 10-digit BAN standard using zero and non-zero first digits, which is a difficult problem in itself,

given the many and diverse representations within a single system. Second, there is the IBAN standard. IBAN (the International Bank Account Number) is a European standard (ECBS, 2002) facilitating cross-border money transfers. The goal is that such transfers become fully traceable at minimal overhead and as cheap as domestic money transfers. The IBAN standard consists of a prefix indicating country, check digits, and bank/branch code, plus the domestic BAN. The string IE29 AIBK 9311 5212 3456 78 is an example IBAN from Ireland. IE is the ISO code for Ireland, the 2 and 9 in IE29 are check digits depending on the entire number, AIBK is short for the Allied Irish Bank, and the tail is a domestic Irish BAN. So, this standard forces the European banks to extend their domestic BAN with a variable location prefix. The IBAN standard is as nonportable as it can get: the country, bank, branch are all present in the standard. After all, its goal is traceability. These additional standards add to the complexity of the 9210-problem. For, people might want to deal with all the problems simultaneously (Koster et al., 2002). This is not a good idea, like it was unwise and hazardous to combine Y2K with the European currency conversion (Jones, 1998, p. 200). Then there is the proposal for (European) number portability, which potentially implies that all banks have to implement all domestic BAN standards (of which there are many to choose from). We will come back to this later, but first explain portability.

4.2. Number portability

Completely orthogonal to the IBAN standard, is the proposed Dutch/European policy of number portability for bank account numbers. This idea stems from regulations in the telecommunications industry: to stimulate competition between operators, number portability for telephone numbers was implemented. The idea is that a customer can switch between operators without unacceptable switching costs. The same was proposed by the Dutch government for bank account numbers to stimulate competition between banks (Working Group on Switching Costs, 2002). We display two advices (among many others) by the Dutch Committee on Switching Costs:

16. Oblige banks to implement number portability for payment accounts.
17. Prevent that European developments (like the International Bank Account number IBAN and the further integration of national payment systems) thwart the implementation of number portability.

The Dutch Government adopted bank account number portability officially (Jorritsma-Lebbink et al., 2002) and claimed to make an effort of having their policy adopted in Europe. Let's see what happens when we implement this.

4.3. Belgian–Dutch portability

Suppose you are living near the Dutch–Belgian border and you want to switch from a Belgian to a Dutch Bank. Number portability implies that the Dutch bank of your choice adopts your Belgian number. Here's an example of a Belgian BAN:

539-0075470-34 (taken from (ECBS, 2002)). The first three digits are the so-called bank-code identifying the bank (immediately showing how happy the Belgian banks will be with number portability). Then there is a 7-digit BAN, plus 2 check digits. The Belgian BAN has the following characteristic: the first ten digits modulo 97 should be equal to the last two digits; this is called the 97-check. Indeed, $5390075470 - 97 \cdot 55567788 = 34$, so the number is according to Belgian standards. Number portability implies that this number needs to be used verbatim by the Dutch bank of the customer's choice. A first small problem is that Dutch BANs are 10-digits, so this is impossible. Suppose we skip the bank-code identifying the Belgian bank; remains 7547034. This number is only a valid Dutch Postbank number: since the first 3 digits are zero it is a Postbank BAN, and there is no check on the Postbank number. So to have sort-of your old number implies you are forced to become a customer of Postbank, which is not encouraging competition. We called the Postbank and they told us that the number is in use by a business client, and it is not possible to obtain it. An alternative is to use the first nine or ten digits: (9)007547034. Except for Postbank, the Dutch domestic BANs satisfy an 11-check. The Dutch modulus 11-check algorithm with weights is used to validate the account number structure. Starting at the right, each digit is multiplied by its respective weight, ranging from 1 to 10. The sum of the resulting numbers is then divided by 11. For the account to be valid, the remainder should be zero. So, for the Belgian number, we get:

$$4 \cdot 1 + 3 \cdot 2 + 0 \cdot 3 + 7 \cdot 4 + 4 \cdot 5 + 5 \cdot 6 + 7 \cdot 7 + 0 \cdot 8 + 0 \cdot 9 + 9 \cdot 10 = 227$$

if we take ten numbers. Dividing 227 by 11 results in a remainder of 7, so this 10-digit BAN is not a valid Dutch BAN. If we remove the leading 9 we get 137, which divided by 11 gives a remainder of 2. So also this number is not a valid Dutch BAN. So, it is impossible to port this Belgian number to a Dutch bank, except if you force Postbank to remove a business client. This is not stimulating competition, but stunting it.

4.4. Dutch–Belgian portability

Now let's see what happens when we want to switch from a Dutch to a Belgian bank. An example of a Dutch BAN is: 0417164300 (taken from (ECBS, 2002)). Suppose we allow the leading zero. We need to comply with the 97-check, so we can only become a customer at those Belgian banks that have a bank-code ending in a zero: e.g., 100-4171643-00. But there is no tenfold n between 100 and 990, such that n-4171643-00 satisfies the 97-check. Suppose we ignore the leading zero as is traditional in the Netherlands: 417164300. Then there are a few more possibilities: there are about 10 valid bank-codes between 100 and 999 such that the 97-check is met. For instance, if we open an account at the bank with bank code 123, the 97-check is correct: $1234171643 - 97 \cdot 12723419 = 0$. Again, we are forced to choose instead of free to choose, disabling competition, rather than enabling it.

4.5. *Possible policies*

These examples show that there can only be one conclusion: in order to get portability working, all Dutch banks have to change all their systems to accommodate for Belgian BANs and vice versa. The used BANs were arbitrarily taken from a report by the European Committee on Banking Standards, from which we quote: "[t]he validation methods differ substantially from country to country and very often from bank to bank within the same country" (ECBS, 2002, p. 3). For instance, all 11-checks are equal, but some 11-checks are more equal than others: the BANs of the Czech Republic also have an 11-check, but use a different weight table: $2^0, 2^1, \ldots, 2^{10}$, so for the Dutch BAN 0417164300 this means:

$$0 \cdot 1 + 4 \cdot 2 + 1 \cdot 2^2 + 7 \cdot 2^3 + 1 \cdot 2^4 + 6 \cdot 2^5 + 4 \cdot 2^6 + 3 \cdot 2^7 + 0 \cdot 2^8 + 0 \cdot 2^{10} = 916$$

which has a remainder of 3 after division by 11. And this type of argument goes on forever. So, with the current systems of the banks, and the current domestic standards, European portability is not possible. Therefore, policy makers face these two possibilities to realize European bank account number portability:

- oblige all the banks operating in Europe to implement all the domestic formats and validation algorithms;
- oblige the customers and the banks to switch en bloc to a new portable European BAN system.

We expect the first possibility to be extremely costly, technically challenging, and likely to fail. For the second possibility, a single portable European standard needs to be developed, all banks need to switch to that standard, abandon the old ones, and all customers need to switch. In addition to the obstacles for the first solution, the latter possibility is also putting the cart before the horses: to prevent number switching, you first have to switch. In the end the customer will pay for the Dutch/European BAN portability, instead of being paid via better competition.

4.6. *Domestic portability*

For domestic portability the above two possibilities could be obliged at the National level. But the conclusion stays the same: mostly harmful. For instance, we already saw Belgium BANs where a bank-code identifies a certain bank; this is common in many other European countries (ECBS, 2002). In the Netherlands, there are "only" two standards, so some assume that number portability is within reach. This is illustrated by an official advice to the Dutch government stating that: while banks are solving 9210, they can just as well implement number portability—*at no additional cost* (Koster et al., 2002, p. 4). This advise has a high IT-soap caliber. Mingling the hugely complex number portability with the already complex 9210-problem does not make sense. Here's why: almost all banking software is written in Cobol. Cobol has special properties, or better: lack thereof. From our 223-benchmark project, and other projects we found that Cobol only allows for hard-wired constants and data-types (Klusener et

al., 2004). So these systems are chockful of hard-coded prefixes, up to entire BANs, hard-wired BAN-related data types, hard-coded internal booking numbers that start to interfere with new 10-digit BANs, and so on. Also it is a Herculean effort to detect all the variations and their bank-specific *hidden semantics*: how to decipher the meaning of certain prefixes, and whether it is exploited. In one system, e.g., we found that the hard-coded prefix 666 implies that the surname of the client starts with the hard-wired character X.[1] We found that other systems exploited this, for instance, to schedule batch jobs for printing bank statements. Or hard-wired bank-codes are exploited to identify a physical location. These bank-codes are used in other systems to streamline certain business processes. For instance, to print bank statements at a central shared service center, to internally ship them near the customers, and then mail them out for quick, accurate, and cheap delivery. There are many more examples, and implementing number portability can have devastating effects on the business due to the semantical mismatches with *alien* numbers. So the policy of domestic BAN portability should be banned. It is close to impossible to implement, let alone at no additional cost.

5. Lessons learned

The Dutch banks are running out of bank account numbers (the 9210-problem); new bank account number standards and number portability are proposed. We conclude that number portability is neither feasible for Europe, nor in the Netherlands. Either the bank's IT-portfolio balloons to accommodate for all the domestic standards, or the customers' cost balloon: they need to switch to prevent future switching and the bank's IT-portfolios need drastic changes as well. Number portability cannot be implemented at no additional cost, while dealing with 9210. The official advise to the Dutch government stating this cannot be taken seriously, neither can its adoption by the Dutch government. Do not mix 9210 with other portfolio-wide projects, since this adds considerable costs and unacceptable risk to a successful solution.

The change costs for 9210 are substantial, and since 9210 is spread over the majority of the IT-systems, the release costs are also high. The cost of change can be reduced by at least a factor 4 if you use automated detection and modification tools. The release costs can only be reduced by putting untested systems in production. Refraining from this still gives a net reduction of a factor 3. For a large international bank 9210 costs about a $100 million; the cost savings are in the $30+ million.

Note

1. We changed the real values for anonymity.

References

Albrecht, A.J. 1979. Measuring application development productivity, *Proceedings of the Joint SHARE/GUIDE/ IBM Application Development Symposium*, pp. 83–92.

Brooks Jr., F.P. 1995. *The Mythical Man-Month—Essays on Software Engineering*, Anniversary edition. Addison-Wesley.

ECBS, 2002. Register of European account numbers, Technical Report ECBS TR 201V2.2.18, European Committee of Banking Standards, Avenue de Tervueren 12, 1040 Brussels, Belgium, www.ecbs.org, select TR201 Report under Publications.

Faust, D. and Verhoef, C. 2003. Software product line migration and deployment, *Software: Practice & Experience* 33: 933–955. Available via: www.cs.vu.nl/~x/pl/pl.pdf.

Jones, C. 1996. *Applied Software Measurement: Assuring Productivity and Quality*, 2nd ed. McGraw-Hill.

Jones, C. 1998. *Estimating Software Costs*. McGraw-Hill.

Jones, C. 1998. *The Year 2000 Software Problem—Quantifying the Costs and Assessing the Consequences*. Addison-Wesley.

Jorritsma-Lebbink, A., de Vries, J.M. and Zalm, G. 2002. Government standpoint on the Final report of the Working Group on Switching Costs, www.ez.nl and select Documenten/Kamerbrieven, select then Juni of Kamerbrieven/2002, finally select the PDF document 00010385 02024673-vtk (in Dutch).

Klusener, A.S., Lämmel, R. and Verhoef, C. 2004. Architectural modifications to deployed software, *Science of Computer Programming*, to appear. Available via: www.cs.vu.nl/~x/am/am.pdf.

Koster, R., Ringnalda, J. and Schepers, R. 2002. The implementation of number portability for bank account numbers in the Netherlands—advise to the Working Group on Switching Costs, www.ez.nl and select Documenten/Kamerbrieven, select then Juni of Kamerbrieven/2002, finally select the PDF document 00010384 02024673-bijlage3 (in Dutch).

Mieritz, L. 2002. Performance management framework—bridging the gap between business and it value, Technical Report, GartnerGroup, Stamford, CT, USA.

Verhoef, C. 2002. Quantitative IT portfolio management, *Science of Computer Programming* 45(1): 1–96. Available via: www.cs.vu.nl/~x/ipm/ipm.pdf.

Verhoef, C. 2004. Quantifying the value of IT-investments, *Science of Computer Programming*, to appear. Available via: www.cs.vu.nl/~x/val/val.pdf.

Verhoef, C. 2004. Quantitative aspects of outsourcing deals, *Science of Computer Programming*, to appear. Available via: www.cs.vu.nl/~x/out/out.pdf.

Working Group on Switching Costs. 2002. Final report of the Working Group on Switching Costs, Technical Report 02ME20, Dutch Government, www.ez.nl/upload/docs/Kamerbrieven/PDF-Documenten/02024673-bijlage1.pdf (in Dutch).

Steven Klusener holds a Master's in computer science from the University of Amsterdam (1990) and a Ph.D. in computer science from the Technical University Eindhoven (1993). His Ph.D. thesis was titled "Models and Axioms for a Fragment of Real Time Process Algebra", the research was actually done at the Dutch Center for Mathematics and Computer Science (CWI) in Amsterdam. After his Ph.D., he was involved in the application of the Formal Methods that were developed at CWI in the industrial practice. Because of the successful cooperation with the industry he moved in 1996 to CapGemini were he got responsible for the development of tooling for their Dutch Y2K Factory. He then worked for a consultancy firm and a Y2K tooling provider. In 1999 he went back to CWI to start, with several other CWI researchers, the Software Improvement Group. In 2002 Steven accepted a 0.4 part-time detachment with the group of Chris Verhoef at the Free University of Amsterdam, from March 1 2004 he is full time employed in this group as group leader of the CaLCE project. Within the CaLCE project concrete problems, related to real-life IT-portfolios, are taken as point of departure for academic research. The project aims at development technology to better maintain and adapt existing software assets with less costs.

Steven current technical interests are grammar engineering, source code analysis and source code transformation, with the main concern that the techniques should be applicable on large software portfolios of several millions of lines of code.

His research is available via www.cs.vu.nl/~steven.

Chris Verhoef is affiliated with the Free University of Amsterdam and scientific advisor for IT innovator Info Support, both in The Netherlands. Before that he was principal external scientific advisor at Deutsche Bank AG, New York, and nonresidential affiliate at the SEI. He is Executive Board member of the IEEE Technical Council on Software Engineering. He serves in Steering Committee, General Chair and Program Chair positions for several important juried research conferences, including the IEEE Working Conference on Reverse Engineering, the European Conference on Software Maintenance and Reengineering and the Working IEEE/IFIP Conference on Software Architecture. He is a frequent speaker on international conferences. He contributed to over 50 papers in conference records and journals. He has acted as scientific advisor in several software intensive areas, notably hardware manufacturers, telecommunications companies, financial enterprises, hi-tech companies, IT-service providers, and government.