Towards a new policy for policy numbers

Historically, policy numbers have been sequential and patterned. Do these requirements provide solutions to business issues?

Or are they vestigial tail-bones and appendices, left over from evolutionary history? And perhaps counter-productive in the current era?

RANDOM THOUGHTS ON SEQUENTIAL NUMBERS

&

SEQUENTIAL THOUGHTS ON RANDOM NUMBERS

Surajit Basu, IBEXI Solutions

Surajit Basu is an IT consultant with over 20 years of experience in design, development and implementation of enterprise-wide solutions in different industries, especially insurance.
Basu received his B. Tech in Computer Science from IIT Kanpur, and his MBA from IIM Calcutta.

He can be reached at surajit_basu@ibexi.com

Table of Contents

Random thoughts on Sequential numbers....3
Really, what's in a number?....3
Statistical sign-posts? Management milestones?....3
The number speaks to me....4
Filing and finding the physical documents....4
Controlling the pre-printed stationery....4
I found the page!....5
Conflict: static id versus changes over time....5
Bottleneck: sequential numbering controllers....5
True needs....6
A static uniqueness....6
Easy to read out, hear and type....6
A summary of the real needs....6
Outside the insurance box....7
A large number is a small set of small numbers....7
Check that digit!....7
Transition from patterned sequential numbers to selection....7
Short strings....7
Sequential thoughts on Random numbers....8
Non-sequitur....8
Trial and error-handling?....8
High density can be disastrous....8
Setting the range....9
Infinite scalability and more....9

Random thoughts on Sequential numbers

Juliet:
O Two-three-o, Two-three-o! wherefore art thou Two-three-o?
What's in a number? That which we call a policy ID
By any other number would smell as sweet.
Deny thy sequence, and change thy number.

Really, what's in a number?

Or in a number of numbers, such as in the insurance business, what's in a policy number, claim number, receipt number etcetera etcetera?
Judging by the mind-numbing requirements of defining the "logic" or "pattern" of these numbers – a lot! The defenders of these patterned sequential numbers have many arguments:

- Statistical or management information: e.g. embedding branch and year is useful for MIS
- Information: e.g. business users can tell from the policy number which product it is
- Filing and finding: physical files are maintained in cupboards based on branch and year
- Control: sequential numbers ensure control of paper documents and fraud prevention
- Reconciliation of printed documents

Do the requirements of patterned sequential numbers provide solutions to business issues? Are these really useful?
Or are they vestigial tail-bones and appendices, left over from evolutionary history? And perhaps counter-productive in the current era?

Statistical sign-posts? Management milestones?

Some people claim that much statistical information such as policies issued, claims registered, receipts issued can be provided easily using sequential numbers, patterned by branch and year. True, a single fact - such as the number of policies issued by the branch in a year – can be easily found by such policy numbering. But alas, management needs are rarely satisfied by quick answers to such simple questions. Usually, the amounts (premium booked, amount collected, claims reserves) are far more important. And even for the numbers or counts, various totals and subtotals are required – such as slices and dices by zone and branch, line of business and product. So, the single logic/pattern that can be embedded into a single policy number is of little use for management information.

For several decades now systems have been able to perform statistical calculations – counts, totals and sub-totals, averages and much much more – accurately and speedily. Using the parts (or sub strings) of the policy number to identify branch or product should be avoided, because the process of identification of the branch or product is dependent on the policy

numbering logic, which may change over time. All management reporting should find the policy's branch from the policy data, not the policy number.

The number speaks to me

An argument for the patterned numbering is often the information it provides the business user instantly. We often hear: "From the policy number, I can tell the branch and line of business instantly."

Not only does it mean having to train business users with numbering logic, but also, what can the user do with the limited information embedded in the policy id? Users access a software application to get the rest of the information needed to answer or act, and hence, the limited information provided by the embedded pattern is of little value, reducing the getting of such information to the level of a clever parlour trick.

In fact, in a world of greater mobility, the information embedded in a fixed id may actually be misleading. For instance, a branch-based policy number can tell where the policy was issued, but not the current servicing branch. Including any policy characteristics which may change over time in the fixed policy number can lead to misinterpretation.

A clever parlour trick?

"What's your policy number?"
"It's 23658765876435."
"Aha! So you are calling from Latur?"
"How did you guess that?"
"The 4th and 5th characters make 58; and our 58th branch was in Latur. Simple!"
"Amazing, but last year, I moved to Hosur."

Filing and finding the physical documents

Business users could also require a patterned number for filing the physical documents. "A policy for motor issued in 2002 from the old Hyderabad branch? That's in the third cupboard in Room 4."

But how many companies now use such methods? Virtual cabinets, folders, scanned images and outsourced storage of physical documents have emerged. The policy number is rarely a reference for the physical location, and current applications and processes track the physical storage separately.

Controlling the pre-printed stationery

Yet another argument for a sequential number is control, especially for pre-printed stationery with sequential numbers. With more and more types of documents being printed as part of the business application, such stationery is increasingly hard to find. It exists, of course, in external stationery, such as cheques as well as in physical controlled documents distributed by the company such as manual cover books and notes.

The key questions for this are: is the system/process designed to meet an external sequential number constraint, such as printing on pre-printed cheques with a given start number? Is there a manual process driving the sequential numbering e.g. is someone generating and writing a number in a register before the proposal is entered? Or is the system generating the number and people note it down– if so, the system controls it: the number is the output of the control, not its centrepiece. Depending on answers to these questions, it is possible to figure out the real need for the sequential numbering controls.

I found the page!

Often we find a justification using the dreaded word: reconciliation! For outgoing documents, the sequential numbers are useful to find a missing document, rather like a bunch of pages printed with page numbers which can be easily checked to find a missing page. However, we find that almost all the time, such reconciliation is no longer done manually, visually; nowadays, reconciliation is usually done by comparing files – of sources and target lists – to identify the mismatches, and hence the sequential number is no longer relevant.

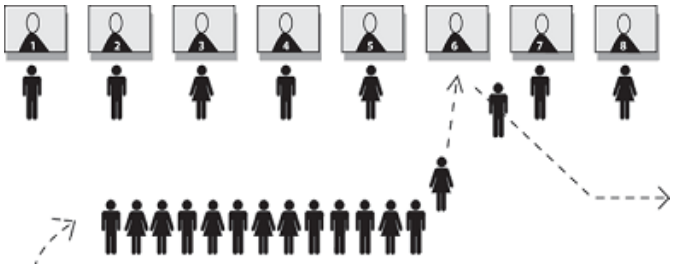
Conflict: static id versus changes over time

On the other hand, in a world of greater demands on flexibility, embedding any changeable characteristics into fixed numbers can increasingly create inflexibility. The primary need for the policy id is a static reference unchanging over time, and that conflicts with embedding any attributes which may change over time. In the telecom world, segregation of telecom numbers by regions made it impossible to have national number portability. Segregation of policies by branch or product can lead to inflexible processes – which do not allow the policyholder to change the branch or product while keeping the same policy number.

Bottleneck: sequential numbering controllers

By definition, sequential numbering requires a single controller allocating the numbers, rather like a token number allocator at a bank branch. This allocator cannot be processed in parallel, or multi-threaded.

One way for partially parallelising this is to have different numbering sequences or ranges for different policy attributes, rather like having two token number allocators at a bank branch, with a note – go to the first one if you are a new customer, else go to the other token allocator.



But this can be only a partial solution, because the number of such sub-sequences is – almost by definition – less than the number of transactions which can be executed in parallel, either because of larger and larger number of online users, or by concurrent threads of a batch process.

True needs

A static uniqueness

What's *really* needed in a number? The primary need for the policy number is a static reference unchanging over time, referring to a specific document or transaction. It can be made of numbers or characters or a combination, and as long as it is unique, associating it with a policy makes the policy unique and unambiguously identifiable.

In a Paper age, or in the age of Disconnected computer systems at branches, to maintain uniqueness, it was necessary to have separate ranges for each disconnected location. And the simplest way for humans to generate a number ensuring uniqueness was to use sequential numbering. But with centralised core systems, and system-generated ids, the need for uniqueness may not lead to the requirement for patterned sequential numbering.

Easy to read out, hear and type

The usage of the number leads to other, often unstated, needs. For instance, when a person says to the call centre operator, my policy id is 93502698768, it should be easily understood.

"Hi, I need a clarification on my policy."
"My policy number is - Nine-zero-zero-zero-zero-zero-zero-zero-zero-four-two-zero-six-six."
"Wait! I got it wrong. It is - Nine-zero-zero-zero-zero-zero-zero-zero-zero-zero-four-two-zero-six-six."
"Let's see. It has a nine, eight zeroes, then four two zero six six."
"No, not two zeroes, two zero."

Do we need to make the number easier to read out? With less typing errors? Definitely.

Do we need to make the number easy to remember? Mnemonic? Not really, we don't expect a person to remember his id numbers except those he refers to often. Fortunately or unfortunately, insurance transactions are rarely needed, and remembering policy ids even less so. So, mnemonic ids are not needed.

A summary of the real needs

The real need in policy ID - claim number, receipt number etcetera etcetera - is most likely a unique, well-presented id, with no embedded attributes. Embedded attributes providing meaning add little value, and can instead be misleading or lead to inflexible processes.. The need for uniqueness does not even necessarily require sequential numbering; in a world of disconnected systems, sequential numbering was the most convenient way to get unique numbers. In a world of connected computer systems, sequential numbering logic can instead prove to be a bottleneck, restricting parallelisation.

What then, in a centralised system, is an optimal solution for unique, well-presented id?

Outside the insurance box

The problems of uniqueness, large number of transactions, and ease of human communications are not in insurance alone. These are in many industries, such as credit cards, telecom and airlines. It is worth looking at these industries for common approaches to the same problems.

A large number is a small set of small numbers

Credit card companies print 16 digit numbers in a way that simplifies the human communication. They have found a set of small numbers easier to tell and type than a large number. So, it may be worth printing, displaying the numbers using that logic, e.g. printing 931502698768 as 931-502-698-768. Just as it is tough to read out $ 12451279832, but easier to read $ 12,451,279,832.

Check that digit!

Another technique used by credit card companies (and very few insurers) is a check digit. The last digit is computed using the previous 15 using a simple algorithm (Luhn's algorithm), and it becomes part of the credit card number. Ensuring this is entered every time the credit card number is entered reduces errors.

Transition from patterned sequential numbers to selection

The telecom industry in India is an interesting case study of change. It has moved from sequential land line numbers, with embedded sub-area codes, to landline numbers which are portable across the city. In mobile numbers, it has moved to non-sequential numbers – initially with regional ranges – to a more portable national number, even allowing the customer to choose his or her own number.

Short strings

Airlines use a different method, with flight tickets referenced by a short string of 6 or 8 characters. Character sets are easier to read out than large numbers.

Sequential thoughts on Random numbers

Non-sequitur

A different model for transactional identification is to generate a unique, but non-sequential id. This requires generating the number – often by generating a random number in a given range - and then checking that it is unique. If it isn't, then another attempt is required to generate a number.

There are three major operational questions with this approach: how many attempts should be made, how do we monitor the usage of the range, what should the range be set to. And one strategic one: why would this seemingly illogical hit-and-miss method work better than the traditional patterned sequential numbering?

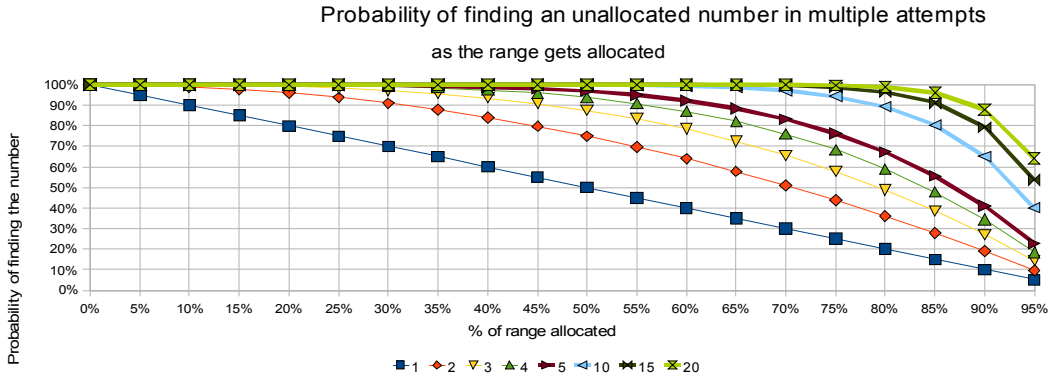
Trial and error-handling?

When the random number generated is found to be allocated, there is no real option: a new attempt has to be made. Setting a limit to the number of attempts means stopping the business process calling the number generator. But it is advisable to set a high number – perhaps 10 or 20 – as a limit. This should make failures very very unlikely – we shall see why, but in unusual situations, it may be better to set a limit and stop a single transaction, and allow a retry for the transaction later than create an infinite request loop. Of course, such errors should trigger an alert to a system administrator to review the reasons immediately.

High density can be disastrous

Initially, it may seem that any range will be adequate, if it is more than the number of the policies to be processed. But note that the logic for random number generation does not depend on the numbers allocated already. So, as the numbers get allocated to policies, the chance that the randomly generated number is already allocated increases, leading to more and more attempts to find a new unallocated random number.

As the number range gets allocated, the probability of the system requiring multiple attempts increases exponentially, and the probability of the system failing to find an unallocated number after say 10 attempts increases exponentially.



Thus, with 10 attempts, the range at 40% allocation has 1 failure per 10,000 transactions. Beyond that, the failure rate shoots up. At higher failure rates, increasing the number of attempts is not very successful; for instance, with 80% allocation, even 20 attempts will lead to 5% failures! The moral is clear: the range should be sparsely used up.

A simple rule of thumb is using 10% as a maximum allocation. This is easy to achieve simply by adding 1 more digit to the policy id than is really required. Setting the full range for the policy number generation ensures a maximum of 10% allocation, with excellent results – average number of attempts will be 1.10 approximately. Adding another digit will reduce the maximum allocation to 1%, and the average number of attempts to near 1.01. That is not a significant improvement, and there may be performance issues with longer policy ids to search for.

Unlike the sequential number model where the number of remaining numbers in the range is monitored, the random numbering model requires monitoring the sparseness or denseness of the allocation of the range. A density of more than 10% should trigger an alert to a system administrator immediately.

Setting the range

Note that like in the airlines industry, it is better to use character sets than just numbers. Short strings are easier to read out and communicate than large numbers. Character sets are also larger, e.g. A-Z has 26 characters. Excluding I, O, U and V to avoid confusion reduces the set to 22 characters; but adding 2,3,4,5,6,7,8,9 gives us a set of 29. But in all cases, the exponents grow, well, exponentially. With 6 characters, it can be used to generate combinations of over 60 million, making it equivalent to 11 digit numbers. With 7 characters, there are over 1.2 billion combinations, so it is equivalent to 13 digit numbers! Surely, that's enough for insurers.

It is also possible to generate a short string e.g. FD2EAS, and to use a simple fast algorithm to find an equivalent number, or vice versa. This duality can at the same time simplify the human communication and the internal database searching – which often works faster with numbers than with characters.

Mike Dunnion, Managing Director at MDC, Ireland says: *We just use 6 characters, using a random generator. When the random reference is generated we first check that it does not already exist in the database and if it does we just generate another.* ***The only issue over the past 13 years has been convincing insurers that 6 characters is sufficient! Old habits die hard.***

It is worth noting that equivalent approaches for non-Latin alphabets – e.g. Chinese, Japanese, Hindi – need to be investigated.

Infinite scalability and more

Why would this seemingly illogical method of trial and errror work better than the traditional guaranteed patterned sequential numbering? This method eliminates bottlenecks and provides full scalability. The random number (or string) generation is a fast, fully parallel process as is the checking if the number is already allocated.

Applications which are clustered will be able to run in parallel with dependence only on common database access. For distributed databases, separate ranges with random number generation in each range can work.

There are other advantages: the random number generation ensures an almost equal spread of numbers in sub-ranges. This allows the policy id to be used for partitioning databases and batch jobs, improving system performance.

Yes, it's time to consider the move from patterned sequential numbering to the random numbering or strings models. Perhaps that is the way ahead for insurers, leaving patterned sequential numbers behind as a relic of the Paper age.

"If YouTube can index billions of videos with a short string, why can't the insurance industry?"
Chester Gladkowski,
Sales & Marketing Officer,
GAPro System, Florida, USA

Juliet:
O Two-three-o, Two-three-o! wherefore art thou Two-three-o?
What's in a number? That which we call a policy ID
By any other number would smell as sweet.
Deny thy sequence, and change thy number.

Two-three-o:
I take thee at thy word.
Call me random, and I'll be new baptiz'd;
Henceforth I never will be Two-three-o.

With apologies to Villain ShakesSpear (after all, what's in a name?)