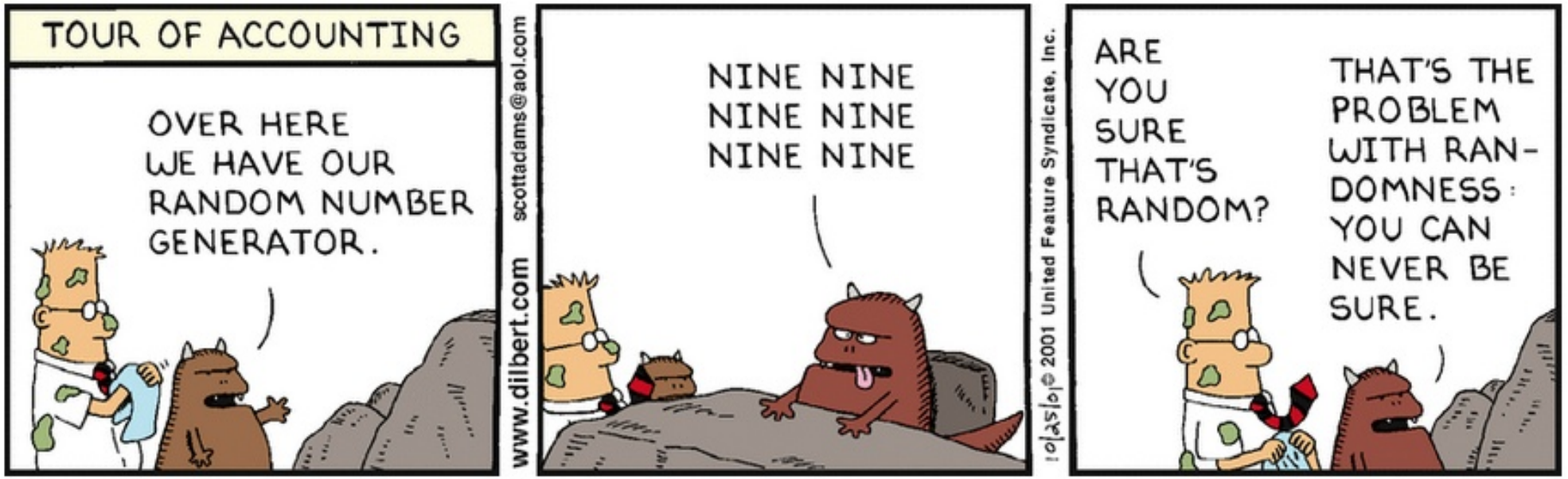
TOUR OF ACCOUNTING
OVER HERE WE HAVE OUR RANDOM NUMBER GENERATOR.
www.dilbert.com scottadams@aol.com
NINE NINE NINE NINE NINE NINE NINE NINE
ARE YOU SURE THAT'S RANDOM?
THAT'S THE PROBLEM WITH RANDOMNESS: YOU CAN NEVER BE SURE.
© 2001 United Feature Syndicate, Inc.

stdlib.h

```
void srand(unsigned int seed)

int rand()
```

rand

Return uniformly distributed integers in [0,RAND_MAX]

C standard only requires RAND_MAX to be greater than 32767 ($2^{15}-1$)

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
printf ("RAND_MAX: %d\n", RAND_MAX);
  return 0;
}
```

srand

Set the initial seed of the PRG

Default seed 1

```
srand ( time (NULL))
```

Current time in seconds from epoch date

It modifies the internal state of the PRG

The internal state is implementation dependent

Example: Linear Congruential Generator

$$I_{n+1} = a\ I_n + c \pmod{n}$$

I_0 is the initial seed

The generated sequence is periodic with period not greater than n (can be way smaller than n for bad combinations of the parameters a, c and n)

The internal PRG state is accessed and modified by `rand()` calls.

No thread safe

```
int rand_r(unsigned int *seedp)
```

How to choose a different range:

If we want to generate integers in the range [A,B] we can use:

```
A+(int)((B-A+1)*rand()/(RAND_MAX+1.0))
```

Better than

```
A+(rand()%(B-A+1))
```

In some PRG last bits have a smaller period
It skews the uniform distribution (worse if the output range is closer to the input range)