Self-Verifying Data

SSQA
November 13, 2012

Douglas Hoffman, BACS, MBA, MSEE,
ASQ-CSQE, ASQ-CMQ/OE, ASQ Fellow
Software Quality Methods, LLC. (SQM)
www.SoftwareQualityMethods.com
doug.hoffman@acm.org

After Functional Testing

Often after basic functional testing I go looking for bugs that I didn't consider. I try to create combinations and sequences that might surface interesting bugs. I like to get the computer to do my work, so I look for a way to automate the tests to do things I can't do manually.

Self-Verifying Data is one oracle technique I apply when working with automation and large data sets.

Self-Verifying Data (SVD)

Self-verifying data is self-descriptive data, i.e., the data contains the key or clues as to what the data is supposed to be.

For example, "This sentence is 52 characters in length (including spaces)."

SVD is just one of many types of oracles

Strategies For SVD

- Self-Descriptive data
- Cyclic algorithms
- Shared keys (with algorithms)
- Attached hash code ("CRC")

Self-Descriptive SVD

Describe the expected result with the data

- Name of font (e.g., Comic Sans MS)
- Color (e.g., Blue)
- Size (e.g., 36 point)
- The following line should be "xyz"
- Etc.

Self-Descriptive SVD

- Usually hand generated, but can be randomly generated
- Most often used with human oracles
- Automated oracles require semantic analysis to interpret the description
- Usually very difficult to automate recognition of attributes

Randomness and Tests

- Random number generators
  - Pseudo-Random numbers
  - Generating random seed values
  - Repeat by reusing the seed value
- Use for randomized input values
- Use for generating randomized data sets

Repeatable Random Series

```
# RUBY code
MAX_SEED = 1_000_000_000
 def initial_RNG_seed(myseed)
    if (myseed == nil) # Check if seed is provided
       # Create a random number to seed RNG
       puts "(no seed passed in, so generate one)"
       myseed = srand()
       myseed = rand(MAX_SEED)
      end
    # print the seed so that we know the seed used
    puts "myseed is #{myseed.to_s}\n"
    foo2 = srand (myseed) # initialize the RNG
    foo = rand()    # generate the [first] random number
    return foo
end
```


Random Series Output Example

```
puts ("First run: #{initial_RNG_seed(nil)} \n \n")
puts ("Second run: #{initial_RNG_seed(400)} \n \n")
puts ("Third run: #{initial_RNG_seed(nil)} \n")
→
(no seed passed in, so generate one)
myseed is 144288918
First run: 0.3705579466087263

myseed is 400
Second run: 0.6687289088341747

(no seed passed in, so generate one)
myseed is 108495905
Third run: 0.09838898989988143
```


Cyclic Algorithm SVD

- Use a repeating pattern to generate data
  - E.g., start, increment, count
  - E.g., basic string, count of iterations
- Identify the pattern in the result
- Confirm the actual pattern is the expected one
  - Extract the pattern for the comparator
  - Embed the key with the data

Cyclic Algorithm Examples

- "ab10abababababababababab"
  - The first two characters are the series value
  - The number is the number of repetitions
- {55, 10, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55}
  - The first number is the series value
  - The second number is the number of repetitions
- {5, 7, 10, 5, 12, 19, 26, 33, 40, 47, 54, 61, 68}
  - The first number is 5
  - The difference between values is 7
  - There are 10 data values

Cyclic Algorithms

- Simple repetitive patterns
- Quick and easy
- Data can be regenerated or key values computed for verification
- Well suited for random data generation
- Automated oracles are straightforward

Shared Keys SVD

1. Generate a coded identifier (e.g., random number seed)
2. Generate the test data using an algorithm
3. Attach the seed to the data
   - Embedded
   - Added field or envelope
4. Confirm by applying the algorithm using the identifier

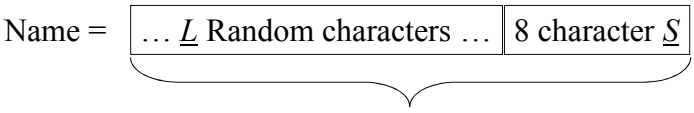
Simple Shared Keys SVD Example

Create a random name:

1. Generate and save random number seed (*S*) and convert to a string
2. Use the first random value using RAND(*S*) as the Length (*L*)
3. Generate random name (*N*) with L characters using RAND()
4. Concatenate the seed to the name

Simple SVD Example

- Assume the seed (*S*) is 8 characters and name field has a maximum of 128 characters
- Generate a random name with length of *L* characters (a maximum of 120)



Shared Keys SVD Example

Create a database record:

1. Generate a random number Seed (*S*)
2. Store the Seed value in an added field within the record
3. Generate the record using the Seed and an algorithm
4. Verify records using the Seed and algorithm

Shared Keys Approach

- Useful for generation of structured data
- Data can be regenerated for verification
- Well suited for generated random data
- Well suited for large volumes of data
- Automated oracles are straightforward
- Often used for parallel or post-test data corruption checking

Attached Hash Code SVD

- An algorithm is applied across all data to generate a [nearly] unique value
- Append the computed value to the data
- Repeat the algorithm to compare appended value with the newly computed value

Attached "CRC" Example

- CRC – cyclic redundancy check
  - Start with the first data value as the working sum
  - Circular shift the working sum by 1 bit
  - Add the next number to the working sum (ignoring overflows)
  - Repeat until end of data
  - Append this generated CRC to the data
  - Check data integrity later by regenerating the CRC and comparing with the appended one

Attached Hash Code Approach

- Useful for quick check of data
- Well suited for all kinds of data
- Well suited for large volumes of data
- Automated oracle is straightforward
- Used for post-test data corruption checking
- Vanishing small chance of false acceptance

Oracle Mechanisms for SVD

- Self-descriptive data
  - Usually human oracle
- Cyclic data
  - Regeneration
  - Pattern identification
- Random data
  - Regeneration
- Hash code check

'SVD' In Context

- May be useful:
  - High volume of inputs or referenced data
  - Key or seed can be used for data generation
  - Straightforward to incorporate key with data
- Usually avoided:
  - Outcomes don't reflect SVD data
  - Overly complex data structures
  - Data not easily generated from a key
  - Key not easy to include with data

Conclusions

- SVD embeds the answer (or its key) with the data
- Four basic types of SVD were identified:
  - Self-descriptive
  - Cyclic
  - Random
  - Attached hash code
- Sometimes SVD is very useful
  - Error detection
  - Unexpected side effects
- It is one of many oracle mechanisms