Testing on the Toilet Presents... Healthy Code on the Commode

Reduce Nesting, Reduce Complexity

Deeply nested code hurts readability and is error-prone. **Try spotting the bug** in the two versions of this code:

Code with too much nesting

```
response = server.Call(request)

if response.GetStatus() == RPC.OK:
  if response.GetAuthorizedUser():
    if response.GetEnc() == 'utf-8':
      if response.GetRows():
        vals = [ParseRow(r) for r in
                response.GetRows()]
        avg = sum(vals) / len(vals)
        return avg, vals
      else:
        raise EmptyError()
    else:
      raise AuthError('unauthorized')
  else:
    raise ValueError('wrong encoding')
else:
  raise RpcError(response.GetStatus())
```

Code with less nesting

```
response = server.Call(request)

if response.GetStatus() != RPC.OK:
  raise RpcError(response.GetStatus())

if not response.GetAuthorizedUser():
  raise ValueError('wrong encoding')

if response.GetEnc() != 'utf-8':
  raise AuthError('unauthorized')

if not response.GetRows():
  raise EmptyError()

vals = [ParseRow(r) for r in
        response.GetRows()]
avg = sum(vals) / len(vals)
return avg, vals
```

Answer: the "*wrong encoding*" and "*unauthorized*" errors are swapped. **This bug is easier to see in the refactored version, since the checks occur right as the errors are handled.**

The refactoring technique shown above is known as ***guard clauses***. A guard clause checks a criterion and fails fast if it is not met. It decouples the computational logic from the error logic. By removing the cognitive gap between error checking and handling, it frees up mental processing power. As a result, **the refactored version is much easier to read and maintain**.

Here are some rules of thumb for reducing nesting in your code:

- Keep conditional blocks short. It increases readability by keeping things local.
- Consider refactoring when your loops and branches are more than 2 levels deep.
- Think about moving nested logic into separate functions. For example, if you need to loop through a list of objects that each contain a list (such as a protocol buffer with repeated fields), you can define a function to process each object instead of using a double nested loop.

Reducing nesting results in more readable code, which leads to discoverable bugs, faster developer iteration, and increased stability. When you can, simplify!

More information, discussion, and archives:
testing.googleblog.com