This is an effort to solve the problem of writing poor software code.

*[This is a work in progress, don’t go all ape shit about some rule you don’t like or don’t understand just yet.]*

1. The examples are written to be generic, please do not complain the syntax used.
2. This is a work in progress, don’t go all ape shit about some rule you don’t like or don’t understand just yet. (do I really need to keep saying this?)

While there are many schools of how to write software correctly not many of them ever reach developers intact. Developers struggle without leadership and definite rules that can be applied to their code.

I present Vanderdecken’s Rules

1. No functions or methods or blocks with more than seven statements
   1. What are not statements
      1. Comments
      2. local variable initialization
      3. static variable initialization
      4. block brackets
      5. the single return point
2. All functions or methods must only have a single return or exit point which is the last item in the block
3. All functions that return a value must declare, initialize, assign and return only that one variable
4. Do not hide code in #defines
   1. create an inline or closure or a function
5. Do not hide values in #defines
   1. Use an enumeration
6. Do not optimize code
   1. Leave optimization to the compilers and languages
   2. Moore’s Law means that by the time you finish you 20% optimization the computers will be 100% faster
7. Elegant code works better
   1. But don’t try to nuance to the point of being unreadable garbage
8. All logic blocks and all loop block require brackets (if supported by the language
9. All if blocks need an else block
   1. func foo( x )

{

int ret = ?

if ( x == 0 )

ret = true

else

if ( x == 1 )

ret = false

return( ret )

}

So what if x == -1 ? the return could be undefined

func foo( x )

{

int ret = undefined

if ( x == 0 )

{

ret = true

}else

if ( x == 1 )

{

ret = false

}

else

{

ret = 2

}

}

return( ret )

1. Called functions cannot crash on error
   1. They need to fail gracefully
2. All class methods must leave the class instance in a stable state
   1. If the method is going to fail, then it must make sure not to leave the class instance in an unstable state.
   2. Do not modify class members until the outcome is known and safe to return.
      1. Don’t do this
         1. Do not set class methods to new values until the computations are done

class method foo( X )

{

self.y = 15

self.w = X

failure

}

Think about how rollbacks work. Do not trash the class before doing work that might fail. Back up the class if needed.

Do the work, get a valid result, then, and only then modify the class or reset it to the condition you found it

1. For the love of God, do not use a sleep() to fix a multi-threaded problem
2. All blocks are required to have brackets if the language allows
   1. The problem is this

if ( x == 1 )

y = x

else

y = -x

return

What usually happens is another line of code is sneaked in

if ( x == 1 )

y = x

else

y = -x

w = x\*y

return

Now, instead of only doing w = x \* y if x != 1, w = x\*y happens every time.