Inhale, Then Exhale

DRY

Separate Data Structures From Control Flow

Input Parameters

Output Parameters

Programming Simplicity

Further reading:

https://guitarvydas.github.io/2023/12/29/Fundamentals-of-Programming.html

https://guitarvydas.github.io/2021/04/20/Git-Could-Do-More.html

The NiCad Clone Detector, Cordy, Roy

Inspiration:

An Orthogonal Model for Code Generation J.R Cordy,

The Design and Application of a Retargetable Peephole Optimizer, Davidson and Fraser

See Also

References *https://guitarvydas.github.io/2024/01/06/References.html*

Blog *https://guitarvydas.github.io/*

Blog *https://publish.obsidian.md/programmingsimplicity*

Videos *https://www.youtube.com/@programmingsimplicity2980*

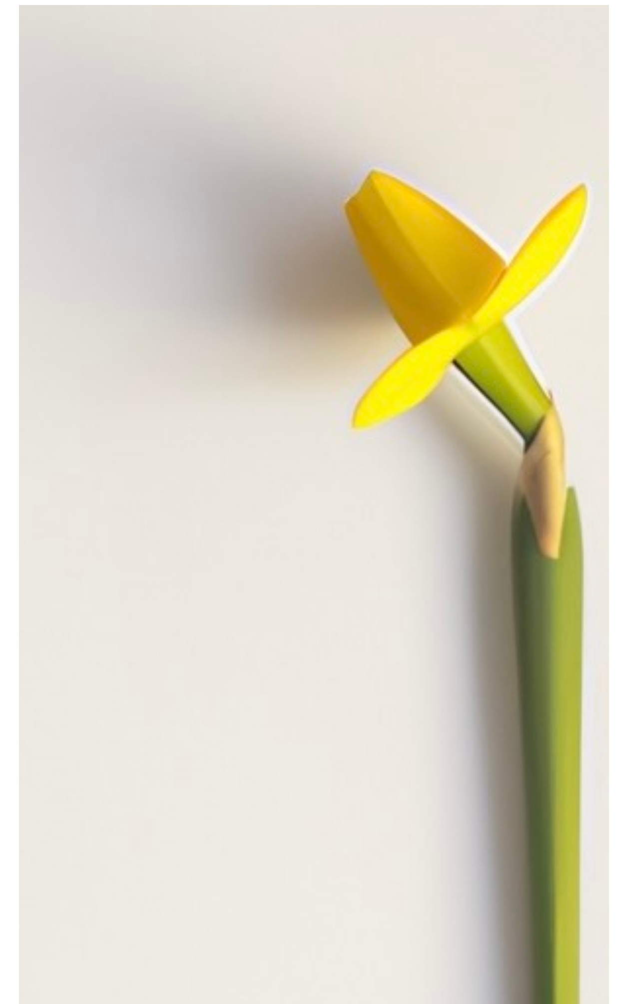
[see playlist "programming simplicity"]

Discord *https://discord.gg/Jjx62ypR*

X *(Twitter) @paul_tarvydas*

More writing *(WIP): https://leanpub.com/u/paul-tarvydas*

Laws of Software Development



Inhale, Then Exhale

1. Read, parse, infer, grok

WHAT? Figure out what is in the input, figure out what else you need to know - infer information from what you've been given.

HOW? OhmJS, relational languages, tables, dictionaries, input validation.

2. Write, rearrange, reformat

WHAT? Once all of the information has been gathered and categorized, output the data in a useful format. Useful for whom? Useful for humans? (Reports, etc.) Useful for machines? (assembler, normalized code, etc.).

HOW? String interpolation, format(), fmt(), printf(), report generator languages and DSLs, create CSVs, create JSON, text-to-text

DRY

WHAT? Make exactly one (1) version of any piece of code. The problem: making more than one version, e.g. by using Copy/Paste, can result in mysterious bugs when not all copies have the same set of edits. Many projects start out with Copy/Paste of code from other projects. While easy to do at first, this results in non-scalability later.

HOW? Abstraction,Parameterized Subroutines, Parameterized Types.

> **Danger**: Abstraction and parameterization usually obfuscate DI (Design Intent)

> **Danger**: manually-applied DRY destroys Locality of Reference, hence, DRY obfuscates DI

Rhetorical question: why do programmers have to deal with DRY, why don't machines figure this out for them? Why don't our tools automatically transform our COPY/PASTE code into DRY code, or, highlight similar sections in some way?

Separate Data Structures From Control Flow

WHAT? Don't embed knowledge of how data is structured in control flow code. For example, if a datum represents the name of someone, don't treat the datum as a low level *string,* create methods that expose useful functionality without exposing the underlying representation, like *.surname()*, etc.

HOW? OOP for data. Methods for operations on data. Syntax for control flow. Orthogonal Programming Languages.

Input Parameters

WHAT? Incoming data, decoupled from sender.

HOW? Most popular languages already have input parameters, called *function parameters*. We don't need to say much more about them, here.

Output Parameters

WHAT? Don't embed knowledge of other functions in code. Calling other functions bakes the names of the other functions into code at the call-point, creating strong coupling and poor scalability.

HOW? Don't call functions directly, except built-ins. If you need to call other functions, pass the functions in as parameters (aka "dependency injection"). Calling a function causes ad-hoc *blocking.* Ideally, don't call functions, just leave data in an output queue (output parameters) for further processing at some other time.

A function call is actually four (4) operations:

1. create parameter bundle
2. invoke callee
3. determine if callee has terminated
4. extract output data from result bundle.

Current programming languages (Python, Rust, etc.) wrap all 4 operations into one line, which is OK for single-threaded apps, but results in accidental complexity for distributed apps. Such languages cause the caller to *block* until the callee has terminated, then require the caller to unpack the results immediately.