Minimalism

A Practical Guide to Writing Less Code

Kevlin Henney

kevlin@curbralan.com

Minimalism

> *It isn't necessary for a work to have a lot of things to look at, to compare, to analyse one by one, to contemplate. The thing as a whole, its quality as a whole, is what is interesting. The main things are alone and are more intense, clear and powerful.*
>
> **Donald Judd**

- A movement in art, architecture, music and interior design
- More generally, a tendency towards less rather than more
  - Seeking the minimum, the lowest point on a curve

Less Code, More Software

> *The difference between a good and a poor architect is that the poor architect succumbs to every temptation and the good one resists it.*
>
> **Ludwig Wittgenstein**

- Creeping featurism and code verbosity do not add to the used functionality of the software
  - Functionality is weakly correlated with the number of lines of code written
- On the other hand, minimalism is not the reduction of source code to line noise
  - Code should be sufficient and fit for purpose

Remove to Improve

> *The minimum could be defined as the perfection that an artefact achieves when it is no longer possible to improve it by subtraction. This is the quality that an object has when every component, every detail, and every junction has been reduced or condensed to the essentials. It is the result of the omission of the inessentials.*
>
> **John Pawson**

- Yes, it's OK to delete code
  - Controlled and considered code implosion shows that a project is maturing
- But remember, minimalism is not nihilism
  - Refactoring is a disciplined practice

A Programmer's Dozen
Refactoring, Repairing and Regaining Control

> ***programmer*** *a person who writes computer programs.*
> ***dozen*** *a group or set of twelve.*
>
> **The New Oxford Dictionary of English**
>
> *Asymmetric bounds are most convenient to program in a language like C in which arrays start from zero: the exclusive upper bound of such an array is equal to the number of elements! Thus when we define a C array with 12 elements, 0 is inclusive lower bound and 12 the exclusive upper bound for the subscripts of the array.*
>
> **Andrew Koenig**
>
> *A **baker's dozen** contains thirteen items as opposed to the familiar twelve. This dates from the time when bakers were subject to heavy fines if they served under-weight bread. To avoid this danger bakers provided a surplus number of loaves, the thirteenth loaf in the dozen being called the vantage loaf.*
>
> **Lock, Stock & Barrel**

Recommendation 0
Follow Form

> *This principle, that of parallel construction, requires that expressions similar in context and function be outwardly similar. The likeness of form enables the reader to recognize more readily the likeness of content and function.*
>
> **Strunk and White**

- Suitable idioms in code should be followed to improve communication
  - Ideolects reduce general comprehensibility
- But idioms should not be followed slavishly if they make little sense or detract from quality

Recommendation 1
Don't Break Contracts

> *Politicians are the same everywhere. They promise to build a bridge even when there's no river.*
>
> **Nikita Khrushchev**

- Methods have a contract for their use
  - Assertions may or may not be enough to enforce the contract fully
- Subclassing and implemented interfaces should follow substitutability
  - Unsupported operations or operations that do a little less or expect a little extra add complexity

Recommendation 2
Include Only What You Need

> *"I wish you wouldn't keep appearing and vanishing so suddenly: you make one quite giddy!"*
>
> *"All right," said the Cat; and this time it vanished quite slowly, beginning with the end of the tail, and ending with the grin, which remained some time after the rest of it had gone.*
>
> *"Well! I've often seen a cat without a grin," thought Alice; "but a grin without a cat! It's the most curious thing I ever saw in all my life!"*
>
> **Lewis Carroll**

- A class typically defines an interface backed by an implementation
  - Separate the usage type from the creation type

Recommendation 3
Express Independent Ideas Independently

> *What do you get when you cross a mosquito with a rock climber?*
> *Nothing, you can't cross a vector and a scalar.*

- Normalise dependencies
  - Consolidate common argument lists and decouple roles within interfaces
- Watch out for porous layers and noosely coupled code
  - Weak wrapping and cyclic package dependencies propagate dependencies unnecessarily

Recommendation 4
Parameterize from Above

> *While moon sets*
> *atop the trees,*
> *leaves cling to rain.*
>
> **Bashō**

- Global packages, Singletons and bundles of constants should be rationalised or eliminated
  - They introduce coincidental coupling, so code is less adaptable and harder to test
- Use arguments and interfaces to invert dependencies

Recommendation 5
Manage Resources Symmetrically

> *As the sunrise to the night,*
> *As the north wind to the clouds.*
>
> **Percy Bysshe Shelley**

- Garbage collection is insufficient for any resource other than memory
  - And even then, it can be less than perfect
- Resource management should be explicitly balanced or fully encapsulated
  - Consider using block objects to wrap resource use

Recommendation 6
Encapsulate

> ***encapsulate*** *enclose (something) in or as if in a capsule.*
> - *express the essential feature of (someone or something) succinctly.*
> - *enclose (a message or signal) in a set of codes which allow use by or transfer through different computer systems or networks.*
> - *provide an interface for (a piece of software or hardware) to allow or simplify access for the user.*
>
> **The New Oxford Dictionary of English**

- Encapsulation is more than just data hiding
  - Self-containedness includes exception safety, thread safety, ease of use, transparency, etc.
- Encapsulation reduces affordances

Recommendation 7
Flow Don't Jump

> *Still glides the Stream, and shall for ever glide;*
> *The Form remains, the Function never dies.*
>
> **William Wordsworth**

- Jumpy control flow should be used sparingly
  - It is easy to get addicted to *break* and early *return*s — and *continue* and *goto* (where it exists)
  - *throw* is an exception
- Control flow that doesn't flow can be hard to comprehend and refactor
  - Structured programming is not completely dead

Recommendation 8
Sharpen Fuzzy Logic

> *This above all: to thine own self be true,*
> *And it must follow, as the night the day,*
> *Thou canst not then be false to any man.*
>
> **William Shakespeare**

- Many verbose coding practices are justified because they simplify debugging
  - This is solving the wrong problem
- Boolean algebra is not rocket science
  - Although you may use it for that purpose

Recommendation 9
Hand Redundant Code its Notice

> *Omit needless words.*
>
> **Strunk and White**

- Code may be less than the sum of its parts
  - Redundant code is code that has no genuine effect
- Redundant code acts as a speed bump to understanding — remove it
  - Unreachable code, synchronising non-thread affected code, unused code, etc.

Recommendation 10
Let the Code Make the Decisions

> *His had been an intellectual decision founded on his conviction that if a little knowledge was a dangerous thing, a lot was lethal.*
>
> **Tom Sharpe**

- There is no need to spell everything out in minute detail
  - Control flow becomes a patchwork of special cases
- Code can also appear to do more than the sum of its parts
  - Polymorphism, lookup tables, grouped actions on collections, double dispatch, etc.

Recommendation 11
Consider Duplicate Code to be a Mistake

> *You cannot step twice into the same river.*
>
> **Heraclitus**

- A lot of programs suffer from a persistent case of the common code
  - Duplicate code is long grass in which bugs thrive
- Visible duplication is a problem best solved by refactoring the commonality
  - Into a class, a method or a block surrounded by clearer logic

Recommendation 12
Prefer Code to Comments

> 1. *If a program is incorrect, it matters little what the documentation says.*
> 2. *If documentation does not agree with the code, it is not worth much.*
> 3. *Consequently, code must largely document itself. If it cannot, rewrite the code rather than increase the supplementary documentation. Good code needs fewer comments than bad code does.*
> 4. *Comments should provide additional information that is not readily obtainable from the code itself. They should never parrot the code.*
> 5. *Mnemonic variable names and labels, and a layout that emphasizes logical structure, help make a program self-documenting.*
>
> **Kernighan and Plauger**

- Code that is not code needs a clear role
  - Most comments do not qualify

Summary

> *Remember that there is no code faster than no code.*
>
> **Taligent's Guide to Designing Programs**

- Baroquecratic practices and just-in-case code have a br(e)aking effect
  - The measure of code is in software behaviour
- Developmental requirements are as important as functional and operational requirements
  - Attention to developmental requirements makes other requirements easier to address