How To Write Unmaintainable Code

Roedy Green, 1997
http://mindprod.com/unmain.html

Outline

- General Principles
- Naming
- Camouflage
- Documentation
- Program Design
- Coding Obfuscation
- Testing
- Philosophy: Roll Your Own
- The Shoemaker Has No Shoes

General Principles

- Programmer wants to rapidly find the place of change.
- Programmer never sees the whole program but a small part visible on screen.
- Any local change made should introduce side effects.
- Force him to read and understand all of the source if he insisting on making a safe change.
- Misinterpret the concept “Information Hiding”

Naming

- Art of naming the identifiers
- Buy a copy of a name book like “Names for Baby”
- If you want easy to type names type several keys gently with your fingers. It will be optimum:
  `asd poi ef oin`
- Use single letter variable names:
  meaningless enough, unguessable.
- Misspell anything meaningful:
  `fuond complated raedy`

Naming (cntd.)

- Use abstract words like, *it, everything, handle, stuff, things, do, calculate, perform*
- use A.C.R.O.N.Y.M.S
- CapiTaliZe aRbItRarilY
- Reuse Names. Give same names to variables, functions, methods, class names as your programming language permit.
- Exploit Compiler Name Length Limits `my_project_counter` and `my_project_tmp` will be interpreted same if there is a limit with 8.

Naming (cntd.)

- Use mixed combinations of underscores: `_`, `__`, `___` as identifiers
- extended ASCII and accents
  `typedef struct {int x,y} ?nt;`
- Rename and Reuse.
  You do not have to respect people named the class libraries, library functions etc. Recycling is good, save the earth.
- When to use `i`
  `i` as the loop variable is old fashion and boring. Use it for anywhere else.

Naming (cntd.)

- Use visual similarities:
  `L0NG , LONG, 1ong, long`
- Misleading names
  `isValid(x)` does not have to check if `x` is valid, and always modify side-effects.
- Use obscure references
  use `MyGirlFriendsFavoriteColor` instead of `blue`.

Camouflage

- Make comments look like code and Vice Versa

```
y += x ;/* y keeps the sum
y += 2 ; * calculated in the loop body
          *Resulting something useless */
printf("%f\n",y);
```

- Overuse "`#define`". It is a gold mine. It also hides compiler errors.
- Overload operators
  Choose unrelated operators for overload.
- Enrich your code with unused variables, functions, methods etc.

Documentation

- Avoid correct, up to date and meaningful documentation.
- Document the obvious
  ```
  i++; // increment the i
  ```
- Do not document the purpose. Leave it to maintainer for fun.
- Do not document the unimportant details. How to add a new syntax, handler, where to make related changes etc.

Documentation (cntd.)

- Giving much detail will also help. Write a design report with >5 levels of titles and hundreds of pages full of useless details.
- Do not document possible troubles and found bugs. Comments like “Fix here” will make you look incompetent.
- Do not document variables. What they represents, what are the units, limits.

Program Design

- Abstraction rule “declare once, use many times” is useless. Guarantee that if some change is to be made, you have to make it in several places, files, directories.
- Types are annoying. Make everything `(void *)` or `Object` and cast later when necessary.
- Never use exceptions, asserts and proper use of error codes. They cut the joy of debugging shorter.
- Do not encapsulate. Be transparent to calling functions, they are also yours .

Program Design (cntd.)

- Copy/paste. Otherwise bugs cannot reproduce.
- Use static arrays and forget to check the index boundaries.
- Make too much of the good things: over-abstraction, over-encapsulation.
- Make use of everything violating well structured OO programming: global variables, side effects, friend declarations, public member variables.

Program Design (cntd.)

- Use many overloaded variants of the same function. Let people try to find out what the difference is.
- Permute everything. Never use the same parameter order for similar functions/methods.
- Avoid configurability. Do not use external configuration files, resource managers, environment variables etc. Recompiling for a simple change is fun.

Coding Obfuscation

- You can always a more strange looking code that makes the same thing. Increment operators, assignment values, pointer arithmetic for arrays etc.

```
for (i=j=t=0;i*j*t;i+=3+j++,scanf("%d",&t)) ;
```

- Use syntactic properties of your language. 040 is more surprising than 32.
- Implicit type conversion is handy.
- Semicolons will fake people:

```
for (i=0;i<10;i++); { ... }
```

Coding Obfuscation (cntd.)

- Nesting. Go as deeper as you can go. No one can continue after 10 open parenthesis or nested blocks.
- Depend on the items described as "compiler dependent" in your manual.
  `a = ++a - ++a;`
- Proper indentation is time and space consuming.
- Use macros (again)
- The longer it is, the harder is to read it. Single page function looks too simple.

Testing

- Avoid it. Don't you have self confidence?
- Avoid performance testing. If it is not fast enough, your customer can buy a faster machine.
- Never write test cases. Several screens and clicks are always sufficient.

Philosophy: Roll Your Own

- Do not share any information with others.
- Create your own standards, own mechanisms, own syntax etc.
- Your job is to make your functions work. Others should fit into your choices.
- You want to make it easy for yourself, neither for users nor for other maintainers.
- You are skillful enough to do it all.

Shoemaker Has No Shoes

- You do not have to use technology.
- Pretty printers, syntax checkers, revision control systems, CASE tools are all for newbies.