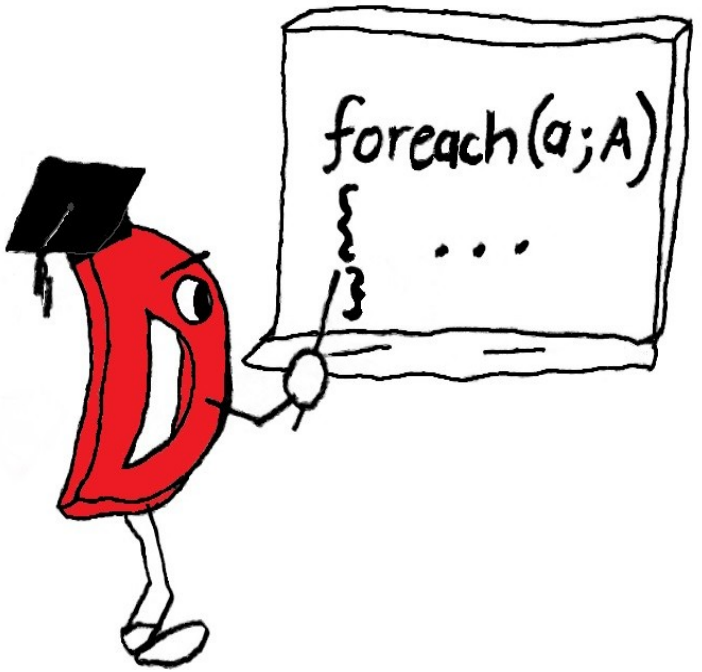
Eliminating Code Smells

by Walter Bright
dlang.org



You Know What I'm Talking About

Pick up unfamiliar code that isn't working. You can smell where the bug is. But what is it that actually smells?

Writing code for 40 years. Seen an awful lot of code. Been finding bugs in other peoples' code. Start to see patterns to bugs, i.e. smells.

Highly Personal

- Based on my experience
- You'll recognize many of these
- I hope you'll see some new ones

Let's Get On With It

```
#include <windows.h>
```

#include <windows.h>

- Should only appear in files that must interoperate with the operating system
- Other files, such as the engine, should not be dependent on the operating system
- Besides eliminating a vast amount of irrelevant information, it makes the program more portable.
- Only the files that include windows.h need to be ported

Nice Formatting

Sloppy formatting is like misspelled words. Who reads text seriously when it has misspellings in it? If the author has taken care to format it nicely, he's likely to have taken care writing it.

Short Global Names

```
extern char c;
```

- Global names should stand out
- Global names should be greppable

Long Local Names

```
for (loopIndex = 0; loopIndex < 10; ++loopIndex)
```

- short names suggest they are local
- long names are useless clutter
- short names don't need to be greppable
- do formulas in books ever use long names?

More Local Names

- Integer variables: i, j, k, l, m, n
- Unsigned: u, v
- Floating point: x, y, z
- Pointers: p
- Arrays: a

Put public interface above the fold

Single Assignment

- Don't reuse variables

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

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

More Single Assignment

```
e = e.semantic(sc);
e = resolve(sc, e);
```

```
auto e2 = e.semantic(sc);
auto e3 = resolve(e2);
```

Even More

```
int i = 8;
if (condition)
    i = 9;
```

```
const i = condition ? 9 : 8;
```

Minimize Variable Scope

```
int i;
...
if (c) {
    i = 3;
    ...
}
```

```
...
if (c) {
    int i = 3;
    ...
}
```

Don't Mix Queries With Actions

- is-a
  - A Mustang is a car
- has-a
  - A Mustang has a cup holder
- can
  - A Mustang can drive

Side Channel Globals

```
int x;
...
int sum(int a, int b) {
    return a + b + x;
}
```

Paper Over the Problem

```
int x;
int getX() { return x; }

int sum(int a, int b) {
    return a + b + getX();
}
```

Correct Fix

```
int x;
...
int sum(int a, int b, int x) {
    return a + b + x;
}
```

Leaky Abstractions

```
struct S {
    S* next;
    ...
}

for (S* s = start; s; s = s.next)
    ...
```

Let's Encapsulate!

```
struct S {
    S* _next;
    S* next() { return _next; }
    ...
}

for (S* s = start; s; s = s.next())
    ...
```

```
 struct SRange {
    this(S* s) { this.s = s; }
    S* front() return  { return s; }
    void popFront() { s = s.next; }
    bool empty()    { return !s; }
    private S* s;
 }

for (sr = SRange(sl); !sr.empty(); sr.popFront())
{
    s = sr.front();
    ...
}
```

More Succintly

```
foreach (s; SRange(sl)) {
    ...
}
```

Random Globals

```
uint changes;
type_t mfoptim;
vec_t defkill;
...
```

Aggregate Globals

```
struct MyGlobals {
   uint changes;
   type_t mfoptim;
   vec_t defkill;
   ...
}

MyGlobals myGlobals;
...

   if (myGlobals.changes) ...
```

const For Variables

```
size_t length = strlen(s);
```

```
const length = strlen(s);
```

No Preprocessor

What Operating System Is “Not Windows” ?

```
#if !Windows
    ...
#endif
```

Better

```
#if Windows
   ...
#elif Linux
   ...
#else
   assert(0);
#endif
```

Overloading Functions That Do Something Completely Different

- hard to grep
- there's no shortage of unique identifiers
- it's just lazy

Returning Allocated Strings

```
char* cat(char* s1, char* s2) {
   const len1 = strlen(s1);
   const len2 = strlen(s2);
   auto s = cast(char*)malloc(len1 + len2 + 1);
   assert(s);
   memcpy(s, s1, len1);
   memcpy(s + len1, s2, len2 + 1);
   return s;
}
```

Push Allocation Up Call Stack

```
void cat(void delegate(char*) put,
                    char* s1,char* s2) {
   put(s1);
   put(s2);
}
```

No I/O In Low Level Functions

```
int sum(int i, int j) {
   if (i > 100)
       fprintf(stderr, "i is out of range\n");
   return i + j;
}
```

Push I/O Up Call Stack

```
int sum(void delegate(char*) put, int i, int j) {
   if (i > 100)
       put("i is out of range");
   return i + j;
}
```

Pull Request Smells

- Too many lines
- Touches too many files
  - encapsulation problem?
- mixing together
  - fixes
  - refactors
  - code motion
  - reformats

git bisect !

Translating Code

- prepare code for translation
- translate with minimal diffs
  - no reformatting
  - no refactoring
  - no bug fixes
  - no enhancements

Conclusion

- My code suffers from all these defects
- I constantly work at fixing them

Trick or Treat!