

B. SPAGHETTI CODE

The following is an excerpt of the textual description of the spaghetti code from Brown’s book [6, page 119–120]. Underlined text highlights key concepts extracted during the Step 1 (Analysis) of our method.

AntiPattern Name: Spaghetti Code

...

Anecdotal Evidence: “Ugh! What a mess!” “You do realize that the language supports more than one function, right?” “It’s easier to rewrite this code than to attempt to modify it.” “Software engineers don’t write spaghetti code.” “The quality of your software structure is an investment for future modification and extension.”

...

General Form

Spaghetti Code appears as a program or system that contains very little software structure. Coding and progressive extensions compromise the software structure to such an extent that the structure lacks clarity, even to the original developer, if he or she is away from the software for any length of time. If developed using an object-oriented language, the software may include a small number of objects that contain methods with very large implementations that invoke a single, multistage process flow. Furthermore, the object methods are invoked in a very predictable manner, and there is a negligible degree of dynamic interaction between the objects in the system. The system is very difficult to maintain and extend, and there is no opportunity to reuse the objects and modules in other similar systems.

Symptoms and Consequences

- After code mining, only parts of object and methods seem suitable for reuse. Mining Spaghetti Code can often be a poor return on investment; this should be taken into account before a decision to mine is made.
- Methods are very process-oriented; frequently, in fact, objects are named as processes.
- The flow of execution is dictated by object implementation, not by the clients of the objects.
- Minimal relationships exist between objects.
- Many object methods have no parameters, and utilize class or global variables for processing.
- The pattern of use of objects is very predictable.
- Code is difficult to reuse, and when it is, it is often through cloning. In many cases, however, code is never considered for reuse.
- Object-oriented talent from industry is difficult to retain.
- Benefits of object orientation are lost; inheritance is not used to extend the system; polymorphism is not used.
- Follow-on maintenance efforts contribute to the problem.
- Software quickly reaches a point of diminishing returns; the effort involved in maintaining an existing code base is greater than the cost of developing a new solution from the ground up.

...

C. EXAMPLE OF SPAGHETTI CODE

We provide an excerpt of the method `startElement` of class `org.apache.xerces.validators.dtd.DTDValidator`, a typical example of spaghetti code, as explained in the introduction.

```
1 public boolean startElement(int, XMLAttrList)
2 throws Exception {
3     ...
4     if (... && !fValidating && !fNamespacesEnabled) {
5         return false;
6     }
7     ...
8     if (contentSpecType == -1 && fValidating) {
9         ...
10    }
11    if (... && elementIndex != -1) {
12        ...
13    }
14    if (DEBUG_PRINT_ATTRIBUTES) {
15        ...
16    }
17
18    if (fNamespacesEnabled) {
19        fNamespacesScope.increaseDepth();
20        if (attrIndex != -1) {
21            int index = attrList.getFirstAttr(attrIndex);
22            while (index != -1) {
23                ...
24                if (fStringPool.equalNames(...)) {
25                    ...
26                } else {...}
27            }
28            index = attrList.getNextAttr(index);
29        }
30    }
31    int prefix = fStringPool.getPrefixForQName(elementType);
32    int elementURI;
33    if (prefix == -1) {
34        ...
35        if (elementURI != -1) {
36            fStringPool.setURIForQName(...);
37        }
38    } else {
39        ...
40        if (elementURI == -1) {
41            ...
42        }
43        fStringPool.setURIForQName(..., elementURI);
44    }
45    if (attrIndex != -1) {
46        int index = attrList.getFirstAttr(attrIndex);
47        while (index != -1) {
48            int attName = attrList.getAttrName(index);
49            if (!fStringPool.equalNames(...)) {
50                ...
51                if (attPrefix != fNamespacesPrefix) {
52                    if (attPrefix == -1) {
53                        ...
54                    } else {
55                        if (uri == -1) {
56                            ...
57                        }
58                        fStringPool.setURIForQName(attName, uri);
59                    }
60                }
61            }
62            if (fElementDepth >= 0) {
63                ...
64            }
65            fElementDepth++;
66            if (fElementDepth == fElementTypeStack.length) {
67                ...
68            }
69            return contentSpecType == fCHILDRENSymbol;
70        }
71    }
```