Locality of Behaviour (LoB)

Carson Gross
May 29, 2020

> "The primary feature for easy maintenance is locality: Locality is that characteristic of source code that enables a programmer to understand that source by looking at only a small portion of it." – Richard Gabriel

The LoB Principle

Locality of Behaviour is the principle that:

> The behaviour of a unit of code should be as obvious as possible by looking only at that unit of code

Discussion

The LoB principle is a simple prescriptive formulation of the quoted statement from Richard Gabriel. In as much as it is possible, and in balance with other concerns, developers should strive to make the behaviour of a code element obvious on inspection.

Consider two different implementations of an AJAX request in HTML, the first in htmx:

```
<button hx-get="/clicked">Click Me</button>
```

and the second in jQuery:

```
$("#d1").on("click", function(){
  $.ajax({
       /* AJAX options... */
  });
});
```

```
<button id="d1">Click Me</button>
```

In the former, the behaviour of the `button` element is obvious on inspection, satisfying the LoB principle.

In the latter, the behaviour of the `button` element is spread out amongst multiple files. It is difficult to know exactly what the button does without a total knowledge of the code base. This "spooky action at a distance" is a source of maintenance issues and stands in the way of developers understanding of the code base.

The htmx example demonstrates good Locality of Behaviour, while the jQuery example has poor Locality of Behaviour.

Surfacing Behaviour vs. Inlining Implementation

A common objection to Locality of Behaviour is that it is inlining implementation details within a code unit, making the code unit less abstract and more brittle.

However, it is important to make the distinction between inlining the *implementation* of some behaviour and inlining the invocation (or declaration) of some behaviour.

Consider functions in most programming languages: there is a distinction between the declaration of function and its use at call sites. A good function abstracts away its implementation details, but is also invoked in an obvious manner, without any spooky action at a distance.

Increasing the obviousness of the behaviour of an element is, ceteris paribus, a good thing, but it falls to both end-developers and especially framework developers to make LoB both as easy and as conceptually clean as possible.

Conflict With Other Development Principles

The LoB will often conflict with other software development principles. Two important ones are:

- DRY - Don't Repeat Yourself

  Software developers typically strive to avoid redundancy in their code or data. This has come to be called "Staying DRY", i.e. Don't Repeat Yourself. Like other software design principles this, on its own, is a good thing. htmx, for example, allows you to place many attributes on parent elements in a DOM and avoid repeating these attributes on children. This is a violation of LoB, in favor of DRY, and such tradeoffs need to be made judiciously by developers.

  Note that the further behaviour gets from the code unit it effects, the more severe the violation of LoB. If it is within a few lines of the code unit, this is less serious than if it is a page away, which is less serious than if it is in a separate file entirely.

  There is no hard and fast rule, but rather subjective tradeoffs that must be made as software developers.

- SoC - Separation Of Concerns

  Separation of concerns a design principle for separating a computer program into distinct sections such that each section addresses a separate concern. A canonical example of this is splitting HTML, CSS, and Javascript. Again, on its own and in isolation this may, indeed, be a good thing. Inlining styles has become more prevalent lately, but there are still strong arguments in favor of SoC in this regard.

  Note that SoC is, however, in conflict with LoB. By tweaking a CSS file the look and, to an extent, behaviour of an element can change dramatically, and it is not obvious where this dramatic change came from. Tools can help to an extent here, but there is still "spooky action at a distance" going on.

  Again, this isn't to condemn SoC wholesale, just to say that there are subjective tradeoffs that must be made when considering how to structure your code. The fact that inline styles have become more prevalent as of late is an indication that SoC is losing some support amongst developers.

Conclusion

LoB is a subjective software design principle that can help make a code base more humane and maintainable. It must be traded off against other design principles and be considered in terms of the limitations of the system a code unit is written in, but, as much as is it is practical, adherence to this principle will increase your software maintainability, quality and sustainability.

</>