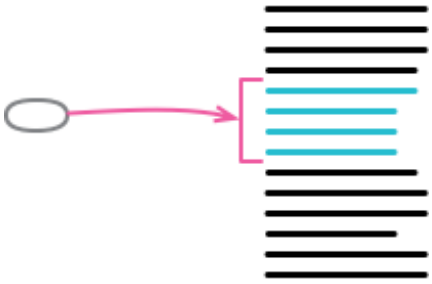
Inline Function

previous: Extract Function

next: Extract Variable



```
function getRating(driver) {
  return moreThanFiveLateDeliveries(driver) ? 2 : 1;
}

function moreThanFiveLateDeliveries(driver) {
  return driver.numberOfLateDeliveries > 5;
}
```

```
function getRating(driver) {
  return (driver.numberOfLateDeliveries > 5) ? 2 : 1;
}
```

inverse of: Extract Function

formerly: Inline Method

Motivation

One of the themes of this book is using short functions named to show their intent, because these functions lead to clearer and easier to read code. But sometimes, I do come across a function in which the body is as clear as the name. Or, I refactor the body

of the code into something that is just as clear as the name. When this happens, I get rid of the function. Indirection can be helpful, but needless indirection is irritating.

I also use Inline Function is when I have a group of functions that seem badly factored. I can inline them all into one big function and then reextract the functions the way I prefer.

I commonly use Inline Function when I see code that's using too much indirection—when it seems that every function does simple delegation to another function, and I get lost in all the delegation. Some of this indirection may be worthwhile, but not all of it. By inlining, I can flush out the useful ones and eliminate the rest.

Mechanics

- Check that this isn't a polymorphic method.
  If this is a method in a class, and has subclasses that override it, then I can't inline it.
- Find all the callers of the function.
- Replace each call with the function's body.
- Test after each replacement.
  The entire inlining doesn't have to be done all at once. If some parts of the inline are tricky, they can be done gradually as opportunity permits.
- Remove the function definition.

Written this way, Inline Function is simple. In general, it isn’t. I could write pages on how to handle recursion, multiple return points, inlining a method into another object when you don’t have accessors, and the like. The reason I don’t is that if you encounter these complexities, you shouldn’t do this refactoring.

Example

In the simplest case, this refactoring is so easy it's trivial. I start with

```
function rating(aDriver) {
  return moreThanFiveLateDeliveries(aDriver) ? 2 : 1;
}
function moreThanFiveLateDeliveries(aDriver) {
  return aDriver.numberOfLateDeliveries > 5;
}
```

I can just take the return expression of the called function and paste it into the caller to replace the call.

```
function rating(aDriver) {
  return aDriver.numberOfLateDeliveries > 5 ? 2 : 1;
}
```

But it can be a little more involved than that, requiring me to do more work to fit the code into its new home. Consider the case where I start with this slight variation on the earlier initial code.

```
function rating(aDriver) {
```

```
  return moreThanFiveLateDeliveries(aDriver) ? 2 : 1;
}

function moreThanFiveLateDeliveries(dvr) {
  return dvr.numberOfLateDeliveries > 5;
}
```

Almost the same, but now the declared argument on `moreThanFiveLateDeliveries` is different to the name of the passed-in argument. So I have to fit the code a little when I do the inline.

```
function rating(aDriver) {
  return aDriver.numberOfLateDeliveries > 5 ? 2 : 1;
}
```

It can be even more involved than this. Consider this code:

```
function reportLines(aCustomer) {
  const lines = [];
  gatherCustomerData(lines, aCustomer);
  return lines;
}
function gatherCustomerData(out, aCustomer) {
  out.push(["name", aCustomer.name]);
  out.push(["location", aCustomer.location]);
}
```

Inlining `gatherCustomerData` into `reportLines` isn't a simple cut and paste. It's not too complicated, and most times I would still do this in one go, with a bit of fitting. But to be cautious, it may make sense to move one line at a time. So I'd start with using Move Statements to Callers on the first line (I'd do it the simple way with a cut, paste, and fit).

```
function reportLines(aCustomer) {
  const lines = [];
  lines.push(["name", aCustomer.name]);
  gatherCustomerData(lines, aCustomer);
  return lines;
}
function gatherCustomerData(out, aCustomer) {
  out.push(["name", aCustomer.name]);
  out.push(["location", aCustomer.location]);
}
```

I then continue with the other lines until I'm done.

```
function reportLines(aCustomer) {
  const lines = [];
  lines.push(["name", aCustomer.name]);
  lines.push(["location", aCustomer.location]);
  return lines;
}
```

The point here is to always be ready to take smaller steps. Most of the time, with the small functions I normally write, I can do Inline Function in one go, even if there is a bit of refitting to do. But if I run into complications, I go one line at a time. Even with one line,

things can get a bit awkward; then, I'll use the more elaborate mechanics for Move Statements to Callers to break things down even more. And if, feeling confident, I do something the quick way and the tests break, I prefer to revert back to my last green code and repeat the refactoring with smaller steps and a touch of chagrin.

previous:
Extract Function

next:
Extract Variable