A Real-Life Example of Refactoring

The development team at Valiant Technology wrote a wrapper for Yelp's Fusion API in a study last summer. The wrapper works nicely and they have implemented it in a "lunch suggestions" bot for Microsoft Teams. After running the codebase via CodeClimate, they found that some of the conditional logic was not as well (or simply) written as it could have been. This is just a small block of code, but it serves as an excellent example of how refactoring can improve code performance, readability, and help reduce technical debt.

The code snippet is from a function that sends authentication details to Yelp's API so we can start making requests for local restaurant information:

Before Refactoring

```
public static function bearerRequest($id, $secret) {
    $uri = self::$apiUri . "/oauth2/token";
    $body = self::urlEncoded([ "grant_type" => "client_credentials", "client_id" => $id, "client_secret" => $secret ], false);
    $response = self::doRequest($uri, $body, "post");
    if($response->code == 200){
        return $response->body;
        } else {
        throw new \Exception("API responded with a HTTP status of {$response->code}.");
    }
}
```

This works fine, but a few minor changes could be made to make the code easier to understand and to remove some conditional logic that is not strictly necessary.

After Refactoring

```
public static function bearerRequest($id, $secret) {
    $uri = self::$apiUri . "/oauth2/token";
    $body = self::urlEncoded([ "grant_type" => "client_credentials", "client_id" => $id, "client_secret" => $secret ], false);
    $response = self::doRequest($uri, $body, "post");
    if($response->code !== 200) throw new \Exception("API responded with a HTTP status of {$response->code}.");
    return $response->body;
}
```

The difference between the code snippet before and after the refactoring is small. We took 5 lines of conditional logic and reduced it to 2 without changing how it works. This makes it easier to read and understand and removes if/then/else conditional logic that wasn't really needed in the first place.

Now, think about this change in terms of scale. It doesn't look like much in the example above, but it can make a big difference in a monolithic application.

There is the potential to remove hundreds or even thousands of rows from a large application - making maintenance much easier and helping reduce technical debt.

The time you can spend debugging or working on a feature is reduced and energy can be spent on more productive, revenue-generating efforts.