```
    {
        return;
    }

    user.name()[0] = std::toupper(user.name()[0]);
}
```

This function was then incorrectly refactored to avoid repetition of the `user.name()` invocation. However, a missing reference qualification leads not only to an unnecessary copy of the string, but also to the function failing to perform its job:

```
void capitalizeName1(User& user)
{
    auto name = user.name();  // Bug, unintended copy

    if (name.empty())
    {
        return;
    }

    name[0] = std::toupper(name[0]);  // Bug, changes the copy
}
```

Furthermore, even a fully cv-ref-qualified **auto** might still prove inadequate in cases as simple as introducing a variable for a returned-temporary value. As an example, consider refactoring the contents of this simple function:

```
void testExpression()
{
    useValue(getValue());
}
```

For debugging or readablity, it can help to use an intermediate variable to store the results of `getValue()`:

```
void testRefactoredExpression()
{
    auto&& tempValue = getValue();
    useValue(tempValue);
}
```

The above invocation of `useValue` is not equivalent to the original expression; the semantics of the program might have changed because `tempValue` is an *lvalue* expression. To get close to the original semantics, `std::forward` and **decltype** must be used to propagate the original value category of `getValue()` to the invocation of `useValue`; see Section 2.1. "Forwarding References" on page 377: