- "**noexcept** Specifier" (§3.1, p. 1085) describes the specifier commonly applied to move operations indicating that they do not throw exceptions.

- "Ref-Qualifiers" (§3.1, p. 1153) explains a feature that allows for overloading member functions on the value category of the object on which they are invoked.

Further Reading

- For the definitive retrospective on value category naming in C++11 by Stroustrup himself, see **stroustrup**.

- For the trail of papers that introduced move semantics, rvalue references, and the refined C++11 value categories, start with N1377 (**hinnant02**) and continue to N3055 (**miller10**). Produced in 2006 during the evolution of the feature, N2027 (**hinnant06**) gives an overview of the basics and cites many of the papers that contributed to how the feature took shape.

- For a solid treatment of the theory value semantics along with its practical applications, see **lakos15a** and **lakos15b**.

- *Effective Modern C++* (**meyers15b**) contains an excellent discussion of value categories, rvalue references, move semantics, and perfect forwarding.

- *C++ Move Semantics — The Complete Guide* is a recent attempt by a world-renowned author to capture all things related to move semantics, including value categories, rvalue references, and perfect forwarding; see **josuttis20b**.

Appendix

The evolution of value categories

What is a value category? In C++, we use declaration statements to introduce named objects and functions into a scope:

```
const int i = 5;  // variable i of type const int having the value 5
double d = 3.14;  // variable d of type double having the value 3.14
double* p = &d;   // variable p of type double* holding the address of d
char f();         // function f returning a value of type char
enum E { A } e;   // variable e of type E enumerating A
```

We can then combine these functions and objects along with literals to form expressions. Some of these expressions might identify an object, and these expressions are all collectively known as *lvalues*: