Chapter 2

Conditionally Safe Features

With great power comes great responsibility. Several modern C++ language features afford immense opportunities for expressiveness, performance, and maintainability; however, they come at a cost. For these features to add value without introducing unacceptable risk, they require a comprehensive understanding of their behavior, effective uses, and consequential pitfalls. This chapter introduces those C++11 and C++14 features that can yield significant, sometimes prodigious, benefits, yet are dangerous when used naively by the uninitiated; the sometimes-latent defects that result might not be discovered until much later in the software development life cycle. As these features offer a moderately high degree of systemic risk when introduced widely into a predominantly C++03 codebase, some form of organized training is an essential prerequisite. An organization's leadership — to feel reasonably comfortable supporting its experienced engineers' use of these features — is well advised to ensure each engineer has been suitably trained in their respective behaviors, effective use cases, and known pitfalls.

Conditionally safe features are characterized by being of moderate to high risk but with rewards that could justify the education and training costs needed to effectively mitigate those risks. Recall from "A *Conditionally Safe* Feature" in Chapter 0 that the default member initializers feature (p. 318) was chosen as representative of *conditionally safe* features. This feature is of moderately low complexity and, though not immune to misuse, addresses a need that often occurs in practice. At the high end of the complexity spectrum, we find the variadic templates feature (p. 873). Though this feature is not especially error prone, the effort needed to master its effective use is substantial for implementers and maintainers alike, yet it offers a degree of power and flexibility that is simply unavailable in C++03. Of moderately high complexity is the flagship feature of modern C++, *rvalue* references (p. 710). This feature fairly demands a substantial training investment, yet its ubiquitous applicability and widespread use typically justify the up-front costs. Although the features presented in this chapter differ widely in their complexities, risks, and general applicability, all provide a plausible value proposition and thus are considered *conditionally safe*.

In short, widespread adoption of *conditionally safe* features offers a sensible risk–reward ratio. All of these features, to be used safely, require some amount of training, hence this book. An organization considering incorporating *conditionally safe* features into a predominantly C++03 codebase would be well advised to ensure its engineers are fully apprised of the behaviors, effective uses, and known pitfalls of these features. Even if you're familiar with the features of modern C++, you will be rewarded by a thorough reading of any section corresponding to a *conditionally safe* feature you might want to employ.