During the development of C++17, there was an attempt to address these concerns.[48] The revised definition is that a trivially copyable class has a *non*deleted trivial destructor, that each of its declared copy and move operations is trivial, and that at least one copy or move operation is both trivial and *non*deleted (note that a **deleted function** is still trivial). Hence, there's now at least one *potentially* (e.g., it could be private or ambiguous) callable copy/move operation to better justify the validity of bitwise copying indicated by the `std::is_trivially_copyable` trait; see *Potential Pitfalls — Using the wrong type trait* on page 482. A similar contemporaneous change also required the destructor of both a trivially copyable class and, hence, a trivial class also be *non*deleted.[49] Applying these changes to the classes above, `S11` is no longer trivially copyable but `C11` remains so. Note, however, that an object of trivially copyable class type having **`private`** copy operations can be copied only by member and **`friend`** functions, unless exploiting the special permission to perform bitwise copies (e.g., using `std::memcpy`) granted for trivially copyable types:

```
C11 c1;      // OK, invokes public default constructor
C11 c2(c1);  // Error, invokes inaccessible private copy constructor

void f11()  // friend of C11
{
    C11 c3(c1);  // OK, invokes private copy constructor as a friend
    // ...
}

void g11()  // nonfriend of C11
{
    C11 c4;
    std::memcpy(&c4, &c1, sizeof c1);  // OK, C11 is still trivially copyable.
}
```

Another issue that evolved over multiple C++ versions was a concern that **`volatile`** data members may not have trivial semantics and therefore might require special copying semantics on certain platforms. This concern was addressed in C++14 by modifying the definitions of trivial copy/move constructors and assignment operators, in turn adding a further restriction to trivially copyable types that they do not have a **`volatile`**-qualified *non***`static`** data member.[50] This change, however, was found to break compatibility with important platform ABIs, so the change was reverted via a defect report against C++14.[51]

```
class V  // trivially copyable in C++11, but might not be in C++14
{
    volatile int i;  // volatile-qualified nonstatic data member
};
```

[48] See **izvekov14**.
[49] See CWG issue 1734; **widman13**.
[50] See CWG issue 496; **maddock04**.
[51] See CWG issue 2094; **vandevoorde15**.