2. string — In this case, string.h and string.cpp would instead be modified so as to depend on vector. Clients wanting to use a string would also be forced to depend physically on vector *at compile time*.

Another possibility might be to create a third component, called stringvector, that itself depends on both vector and string. By **escalating**[8] the mutual dependency to a higher level in the physical hierarchy, we avoid forcing any client to depend on more than what is actually needed. The practical drawback to this approach is that only those clients that proactively include the composite stringvector.h header would realize any benefit; fortunately, in this case, there is no **one-definition rule (ODR)** violation if they don't.

Finally, complex machinery could be added to both string.h and vector.h to conditionally include stringvector.h whenever both of the other headers are included; such heroic efforts would, nonetheless, involve a **cyclic physical dependency** among all three of these components. Circular intercomponent collaborations are best avoided.[9]

All members of an explicitly defined template class must be valid

In general, when using a class template, only those members that are actually used get implicitly instantiated. This hallmark allows class templates to provide functionality for **parameter** types having certain capabilities, e.g., default constructible, while also providing partial support for types lacking those same capabilities. When providing an explicit-instantiation definition, however, *all* members of a class template are instantiated.

Consider a simple class template having a **data member** that can be either **default-initialized** via the template's **default constructor** or initialized with an instance of the member's type supplied at construction:

```
template <typename T>
class W
{
    T d_t;  // a data member of type T

public:
    W() : d_t() {}
        // Create an instance of W with a default-constructed T member.

    W(const T& t) : d_t(t) {}
        // Create an instance of W with a copy of the specified t.

    void doStuff() { /* do stuff */ }
};
```

This class template can be used successfully with a type, such as U in the following code snippet, that is not default constructible:

[8] **lakos20**, section 3.5.2, "Escalation," pp. 604–614

[9] **lakos20**, section 3.4, "Avoiding Cyclic Link-Time Dependencies," pp. 592–601