- Easily adapted to accommodate new or changed requirements
- Based on data.

In addition to these characteristics, the software construct should also have:

- encapsulated data
- a well-defined public interface.

This type of unit would be easier to test thoroughly both in isolation and as part of an integrated test plan. It would more likely be reusable and could be included in a software library.

The object-oriented community proposed the *object* as a software construct which had the characteristics listed above. Objects, although they provide functionality, are based on the data in the system. This provides a structure that is less subject to change. As we shall see later in this chapter, an object protects (or encapsulates) its data by making the data accessible only by using the object's publicly declared operations. Well-designed objects are independent and autonomous, making them easy to test in isolation before being integrated into the system. A good object is cohesive, i.e. concerned with a single idea, which makes it suitable for inclusion in a software library for possible reuse.

Use case decomposition and object-oriented decomposition

Although objects are based on the data in the system, this does not mean that object-oriented software ignores the required functionality. We have seen in Chapter 3 that the use case model specifies, from the user's point of view, what the system must do. The objects we define must be capable of delivering that functionality. The use case model provides one decomposition of the system, based on the required functionality. However, the software structure of an object-oriented system is based on the object. In other words when we come to build the software and divide it up into separate parts we don't split it up according to use cases but by objects and groups of objects.

Another advantage of the object-oriented approach is that it provides a seamless development process. The objects we identify as part of the analysis stage persist through the development stages to the code. This thread of continuity means the software objects have a certain predictability – we have expectations, for example, of what an object representing a customer will look like and what it will do. In well-designed object-oriented code our expectations will not be disappointed. This makes the code easier to understand and therefore easier to maintain.