Chapter 8 - Boundaries

We seldom control all the software in our systems. Sometimes we buy third-party pack- ages or use open source. Other times we depend on teams in our own company to produce components or subsystems for us. Somehow we must cleanly integrate this foreign code with our own.

Using Third-Party Code

There is a natural tension between the provider of an interface and the user of an interface. Providers of third-party packages and frameworks strive for broad applicability so they can work in many environments and appeal to a wide audience. Users, on the other hand, want an interface that is focused on their particular needs. This tension can cause problems at the boundaries of our systems. Example:

```
Map sensors = new HashMap();
Sensor s = (Sensor)sensors.get(sensorId);
```

VS

```
public class Sensors {
  private Map sensors = new HashMap();

  public Sensor getById(String id) {
    return (Sensor) sensors.get(id);
  }
  //snip
}
```

The first code exposes the casting in the Map, while the second is able to evolve with very little impact on the rest of the application. The casting and type management is handled inside the Sensors class.

The interface is also tailored and constrained to meet the needs of the application. It results in code that is easier to understand and harder to misuse. The Sensors class can enforce design and business rules.

Exploring and Learning Boundaries

Third-party code helps us get more functionality delivered in less time. Where do we start when we want to utilize some third-party package? It’s not our job to test the third-party code, but it may be in our best interest to write tests for the third-party code we use.

It's a good idea write some test for learn and understand how to use a third-party code. Newkirk calls such tests learning tests.

Learning Tests Are Better Than Free

The learning tests end up costing nothing. We had to learn the API anyway, and writing those tests was an easy and isolated way to get that knowledge. The learning tests were precise experiments that helped increase our understanding.

Not only are learning tests free, they have a positive return on investment. When there are new releases of the third-party package, we run the learning tests to see whether there are behavioral differences.

Using Code That Does Not Yet Exist

Some times it's necessary work in a module that will be connected to another module under develop, and we have no idea about how to send the information, because the API had not been designed yet. In this cases it's recommendable create an interface for encapsulate the communication with the pending module. In this way we maintain the control over our module and we can test although the second module isn't available yet.

Clean Boundaries

Interesting things happen at boundaries. Change is one of those things. Good software designs accommodate change without huge investments and rework. When we use code that is out of our control, special care must be taken to protect our investment and make sure future change is not too costly.