How Should Teams Organize their Applications?

Juanita J. Ewing
Instantiations, Inc.

In my previous column I began to address some important issues for teams of Smalltalk programmers. Teams of programmers are important because large complex applications cannot be built by a single programmer. Continuing with issues relating to teams, this column will present heuristics for organizing applications. Organizational units can be the basis of work assignments for team members and the basis for distributing completed portions of an application. An additional benefit is that organizational units tend to represent reusable units.

Should teams organize their application? When a team of programmers implements an application, the development work needs to be structured and distributed among team members. Without some kind of organization, development would be a free-for-all, and no schedule would be possible. The most obvious organizational technique is to partition an application along class lines. In this organizational scheme, each member of the team would be responsible for implementing and maintaining a group of classes. In Smalltalk, a class is a unit that encapsulates behavior and the data specification for a particular kind of object. An organization based on classes has the advantage of being built on an existing supported Smalltalk unit, and is able to use many of the existing Smalltalk tools. But, classes don't exist in isolation.

Should hierarchically–related classes be organized together? Classes are usually part of a hierarchy in which superclasses also specify data and behavior. The behavior of an object is defined by the behavior in its own class and the behavior of its superclasses. Since a class requires its superclass in order to function, it is desirable to organize both classes together. This desire is the basis of our first heuristic.

This kind of grouping usually involves several classes, since an inheritance tree is frequently larger than just two classes. Entire trees of hierarchically–related classes might be grouped together to satisfy this heuristic, but it cannot be followed blindly. If it were, most of the classes in an image would be grouped together.

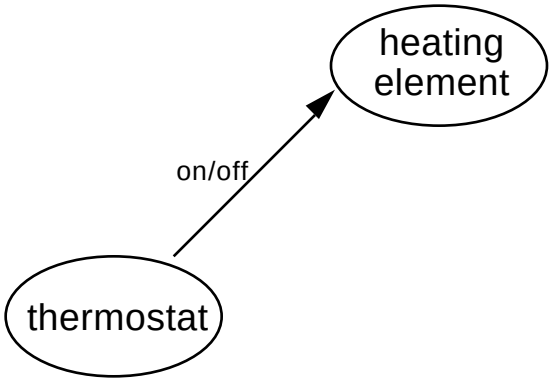
How do you limit the hierarchical groups? If classes were grouped strictly by inheritance, the size of groups would not be reasonable. Use the additional heuristic that hierarchically–related classes performing a similar function should be grouped together.

For example, suppose you are developing an application that has a plumbing system. The plumbing system is composed of plumbing components such as valves, spigots and pipes. All of the plumbing components are subclasses of an abstract class, PlumbingComponent. PlumbingComponent is a subclass of Object. A group based on function would contain PlumbingComponent and all its subclasses, but would not contain the superclass Object because it does not fulfill the same function as a plumbing component.

Should collaborating classes be organized together? Frequently an application contains several classes that send messages back and forth. These classes collaborate. Collaborating classes require each other to function. Because these individual classes don't stand alone it is desirable to organize these classes together.

The degree of collaboration affects this organizational heuristic. If two classes collaborate with just one message, then the degree of collaboration is small. Many messages indicate a large degree of collaboration, and a stronger reason to organize the classes together.

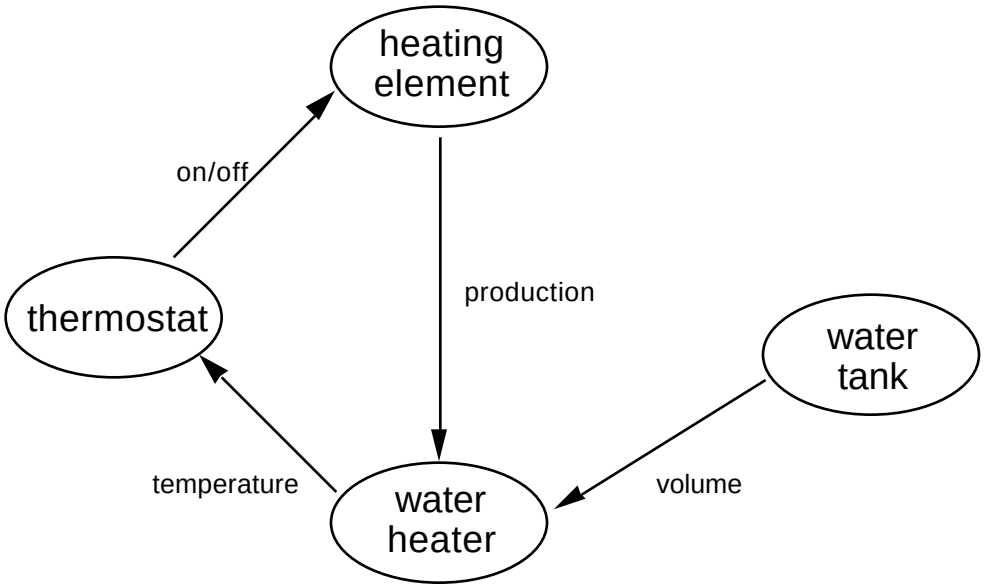
Suppose our plumbing system contains a water heater. A water heater has a water tank, a heating element and a thermostat. The heating element must be turned on and off when the water temperature, as sensed by the thermostat, reaches upper and lower limits. The thermostat sends messages such as turnOn and turnOff to the heating element. These two classes collaborate and therefore should be grouped together.



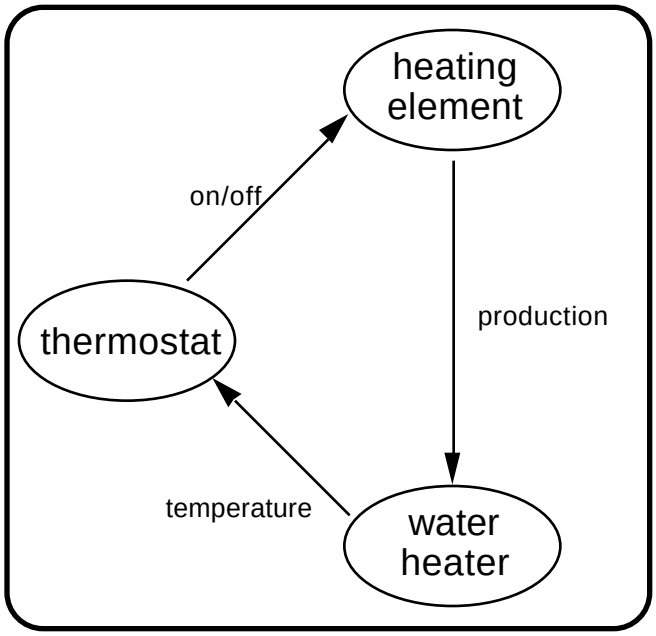
We haven't addressed the issue of how to organize the water tank class in our example. The heating element would send messages indicating how much heat it has produced, and based on the volume of water, a temperature rise could be calculated. Does the water tank perform the temperature rise calculation? No. The water tank is responsible for knowing its volume of water, but nothing about a generic water tank suggests that it be able to calculate temperature rises. (We will ignore volume fluctuations based on temperature variations.)

Our system also needs to include a water heater object that performs operations specific to a water heater such as calculating the temperature rise. The heating element communicates with the water heater object to pass on heat production, and the water heater tells the thermostat the current temperature.

Because of the collaboration between the water heater and both the heating element and the thermostat, the water heater should also be included in the organization based on collaboration.



The water tank collaborates with only one of the other classes in this example. Because of the small degree of collaboration and also because the information doesn't take an active role in the primary calculation, we leave it out of the water heater group.



Our group contains three classes: heating element, thermostat and water heater. This organization is based solely on collaboration.

What if your classes are in a hierarchy and collaborating? It is likely that your application contains classes that belong to a hierarchy and also collaborate with other unrelated classes. Both hierarchical and non-hierarchical relationships should be taken into account. Classes in the hierarchy should be organized together, and tightly coupled classes in the application should be organized together.

Let's examine the water heater example. Some of the objects in this system are hierarchically related. The thermostat and the heating element are part of an electrical component hierarchy, and the tank is part of the plumbing component hierarchy. Yet we also want to capture the relationships based on collaboration, as depicted in the diagram. There is a desire to associate the heating element with other electrical components, and a desire to associate it with the other classes comprising the water heater.

How do you organize a class in more than one way? We have discussed a unit that captures a single organization. Let's call this unit the *primary organizational unit*. In order to represent multiple overlapping associations, we need another type of organization. *Configurations* are another type of organization that is used to represent secondary relationships. Configurations refer to other organizational units. As such, they are another level of organization. They can be nested, so that one configuration may refer to another configuration or simply to a group of primary organizational units.

In most cases a hierarchical relationship forms the basis for the primary organizational unit. This unit can then be combined with other units via configurations. This tactic reflects the point of view that the hierarchical relationship is tighter and more stable than collaboration–based relationships.

Another way to think about different organizations is to imagine scenarios for reuse and maintenance. If developers are more likely to reuse a hierarchy of classes than a group of collaborating classes, then the primary organization should be based on inheritance. If a group of classes will be maintained as a unit, this means that they are closely related and should be grouped together.

In the plumbing example, all the plumbing components can be organized into a primary unit. This unit contains classes related by inheritance. The same should be done for the electrical components. A configuration representing the water heater would contain three primary units:

- the plumbing components unit
  (for the water tank)
- the electrical components unit
  (for the thermostat and heating element)
- the water heater unit consisting of only the water heater class.

With inheritance as the organizational basis for primary units, collaboration–based relationships can be represented by configurations. In our example, this organization is useful because the plumbing components can exist in different systems. You can imagine the configurations and primary organizational units needed to represent a well and pump, or a solar hot water heater.

What about related code in other classes? All parts of an application might be neatly contained in classes. Frequently, though, methods will be sprinkled throughout the class library.

Suppose a way to distinguish between other objects and plumbing components is needed. It is reasonable for a developer to define a method in Object that answers whether the receiver is a plumbing component (isPlumbingComponent). This method returns false. A similar method implemented in PlumbingComponent returns true. All classes inheriting the

method from Object will answer false when asked if they are a plumbing component. Subclasses are free to override the method.

Should code organization be based on classes? In our example, a single method in an unrelated class has functionality that relates to another class. How should this method be organized? Should the method in Object be associated with the class Object, or should it be part of the plumbing component unit? Obviously, this method relates to plumbing components, and not to the generalized behavior of objects in a Smalltalk system. It should be associated with the plumbing component classes.

Organizing strictly along class boundaries is not flexible enough. I advocate a flexible grouping scheme in which classes and methods can be organized together into primary units. (We will refer to classes and methods as definitions.) Use the heuristic that functionally–related definitions should be organized together without regard to class boundaries.

In the plumbing system example, the classes composing the plumbing component hierarchy and the method Object>isPlumbingComponent should be bundled into one primary unit. Both implementations of isPlumbingComponent would be contained by the same primary unit. It is likely that many of the definitions would be used together and maintained together. In particular, if the meaning of Object>isPlumbingComponent is changed, then PlumbingComponent>isPlumbingComponent is also likely to change.

Use these heuristics to organize your application:

- organize hierarchically–related classes together.
- use functionality to limit the size of hierarchically–based groups.
- organize collaborating classes together.
- put functionally–related definitions together.

Two types of organization are required to represent different kinds of relationships, and retain the flexibility required in a highly productive environment like Smalltalk. With the primary organization units, a developer can bundle definitions together that are closely related—either through inheritance or collaboration. These definitions are maintained together and reused together. Primary organizational units contain logical groups of definitions that cannot stand alone.

Configurations are another level of organizational structure that represents secondary relationships. The components of a configuration stand alone and are more likely to be used in other situations. Developers should be encouraged to mix and match different organizational units to extend the usability of a set of definitions. Secondary relationships, represented by configurations, are the basis of mixing and matching.

Primary organizational units are suitable for organizing the development of an application. Each team member should be assigned to implement one or more primary organizational units, and the implemented units should be distributed to other team members. Configurations are used to build up the various subsystems in an application and ultimately specify the application itself.