**What is the difference between a view and a view point?**

An architectural view is a way to portray those aspects or elements of the architecture that are relevant to the concerns the view intends to address—and, by implication, the stakeholders to whom those concerns are important.

This idea is not new, going back at least as far as the work of David Parnas in the 1970s and more recently Dewayne Perry and Alexander Wolf in the early 1990s. However, it wasn’t until 1995 that Philippe Kruchten of the Rational Corporation published his widely accepted written description of views, Architectural Blueprints—The “4 + 1” View Model of Software Architecture. This suggested four different views of a system and the use of a set of scenarios (use cases) to elucidate its behavior. Kruchten’s approach has since evolved to form an important part of the Rational Unified Process (RUP).

A view is a representation of one or more structural aspects of an architecture that illustrates how the architecture addresses one or more concerns held by one or more of its stakeholders.

A viewpoint is a collection of patterns, templates, and conventions for constructing one type of view. It defines the stakeholders whose concerns are reflected in the viewpoint and the guidelines, principles, and template models for constructing its views.

Architectural viewpoints provide a framework for capturing reusable architectural knowledge that can be used to guide the creation of a particular type of (partial) AD. You may find it helpful to compare the relationship between viewpoints and views to the relationship between classes and objects in object-oriented development.

A class definition provides a template for the construction of an object. An object-oriented system will include at runtime a number of objects, each of a specified class.

A viewpoint provides a template for the construction of a view. A viewpoints-and-views-based architecture definition will include a number of views, each conforming to a specificviewpoint.

Viewpoints are an important way of bringing much-needed structure and consistency to what was in the past a fairly unstructured activity. By defining a standard approach, a standard language, and even a standard metamodel for describing different aspects of a system, stakeholders can understand any AD that conforms to these standards once familiar with them.

In practice, of course, we haven’t fully achieved this goal yet. There are no universally accepted ways to model software architectures, and many ADs use their own homegrown conventions (or even worse, no particular conventions at all). However, the widespread acceptance of techniques such as entity-relationship models and of modeling languages such as UML takes us some way toward this goal.

In any case, it is extremely useful to be able to categorize views according to the types of concerns and architectural elements they present.