Integrating Architecture-Centric Methods into Object-Oriented Analysis and Design

Raghvinder S. Sangwan
The Pennsylvania State University
rsangwan@psu.edu

It has long been known that software systems are complex. Brooks asserted that much of this complexity was inherent to the domain: "the systems we produce are themselves the solutions to complex problems that are constantly changing; the intangible nature of software defies our abilities to represent or visualize it without eliminating essential detail; and it provides the connective tissue between a vast array of other systems, both man and machine." [1] In the twenty years that have passed since Brooks made those assertions the scale of systems has grown enormously, making the problem even more daunting. Fortunately this period has also seen significant advances in our understanding of what constitutes complexity in those systems. We now have catalogs of heuristics, principles, and patterns that document how to construct software designs that are extensible, robust, and comprehendible along with tools and techniques that identify and highlight when these guidelines are transgressed. Despite these advances, however, the problem remains – we continue to develop overly complex systems.

Hoare states: "there are two ways of constructing a software design: one way is to make it so simple that there are obviously no deficiencies; the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult." [2] Hoare implies, therefore, that while creating simple designs is anything but easy, it is far easier to create complicated designs. From our perspective, a design becomes complicated not because it needs to contend with the essential complexity inherent in a system but introduction of excessive complexity that can be an incidental byproduct of a chosen development methodology.

Complexity, in its many forms, is a systemic property of a system, indeed it is an emergent property that arises from the interdependence and interconnectivity of the system, both structurally and dynamically. Consequently, an appropriate approach to system development likely to keep excessive complexity in check must be systemic. Mainstream design methodologies such as Object-Oriented Analysis and Design (OOAD), however, treat systemic properties only indirectly or implicitly. The quality of systems developed using such methodologies, thus, depends largely on the skill level and experience of its architect. It has been suggested, therefore, that augmenting these methodologies with software architecture-centric methods such as the Quality Attribute Workshop (QAW) and Attribute Driven Design (ADD) can provide explicit and methodical guidance to an architect in creating systems with desirable qualities [3].

In this tutorial, we will first go through the exercise of applying OOAD techniques (use case analysis, domain modeling, and component-based design) for creating the architecture for a system from the building automation domain. We then use the techniques prescribed by the architecture-centric methods (quality attribute workshop and attribute driven design) for creating the architecture for the same system. These two exercises are used to clearly demonstrate the shortcomings of OOAD. We finally demonstrate how OOAD can be augmented with architecture-centric methods to overcome these shortcomings.

References

[1] F.P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley, Boston, MA, 1995.

[2] C.A.R. Hoare, "The Emperor's Old Clothes," *Communications of the ACM*, Vol. 24, No. 2, February 1981, pp. 75 – 83.

[3] Sangwan, R. Neill, C., El Houda, Z. and Bass, M. "Integrating Software Architecture-Centric Methods into Object-Oriented Analysis and Design," *Journal of Systems and Software*, Volume 81, Issue 5, May 2008, Pages 727-746.