Using Pattern Languages for Object-Oriented Programs

Kent Beck, Apple Computer, Inc.
Ward Cunningham, Tektronix, Inc.

Technical Report No. CR-87-43
September 17, 1987

Submitted to the OOPSLA-87 workshop on the
Specification and Design for Object-Oriented Programming.

Abstract

We outline our adaptation of Pattern Language to object-oriented programming. We sumarize a system of five patterns we have successfuly used for designing window-based user interfaces and present in slightly more detail a single pattern drawn from our current effort to record a complete pattern language for object-oriented programs.

The search for an appropriate methodology for object-oriented programming has seen the usual rehash of tired old ideas, but the fact is that OOP is so different that no mere force-fit of structured analysis or entity-relationship methods will provide access to the potential inherent in OOP. In particular, neither of these methods address the user interface design issues that have obviously become of paramount importance. In addition, while E-R seems to be "object-oriented" it is not suited to the dynamic nature of objects as in Smalltalk and encourages the use of a global perspective while designing, a sure lose in object-oriented programming.

We propose a radical shift in the burden of design and implementation, using concepts adapted from the work of Christopher Alexander, an architect and founder of the Center for Environmental Structures. Alexander proposes homes and offices be designed and built by their eventual occupants. These people, he reasons, know best their requirements for a particular structure. We agree, and make the same argument for computer programs. Computer users should write their own programs. The idea sounds foolish when one considers the size and complexity of both buildings and programs, and the years of training for the design professions. Yet Alexander offers a convincing scenario. It revolves around a concept called a "pattern language."

A pattern language guides a designer by providing workable solutions to all of the problems known to arise in the course of design. It is a sequence of bits of knowledge written in a style and arranged in an order which leads a designer to ask (and answer) the right questions at the right time. Alexander encodes these bits of knowledge in written patterns, each sharing the same structure. Each has a statement of a problem, a summary of circumstances creating the problem and, most important, a solution that works in those circumstances. A pattern language collects the patterns for a complete structure, a residential building for example, or an interactive computer program. Within a pattern language, patterns connect to other patterns where decisions made in one influence the others. A written pattern includes these connections as prologue and epilogue. Alexander has shown that nontrivial languages can be organized without cycles in their influence and that this allows the design process to proceed without any need for reversing prior decisions [Alex77].

Consider a very small pattern language for designing Smalltalk windows. We suggest the following patterns:

1. Window Per Task
2. Few Panes Per Window
3. Standard Panes
4. Short Menus
5. Nouns and Verbs

We presented these patterns to a team of application specialists writing a specification for a special purpose programming environment. Without detailed understanding of any of Smalltalk's interface mechanisms (MVC for example) they were able to specify very reasonable interfaces after one day of practice [Cunn87]. Note that we sorted and numbered the patterns. Pattern 1 must be addressed first. It decides what windows will be available and what will be done in them. Next patterns 2 and 3 divide each window into panes. Finally patterns 4 and 5 determine what selections and actions will do within each pane. This order was derived from the topology of influences between each pattern.

We have begun writing a complete pattern language for object-oriented programming. An example from this language is the pattern entitled Collect Low-level Protocol. Here it is in abbreviated form:

> Once you have initially decomposed a system into objects [Objects from the User's World] and refined the objects [Engines and Holders] you need to begin collecting useful functionality that doesn't particularly fit into any single object. Often many objects need to communicate with low-level (bit- or byte-oriented) parts of the system. For example, external files can have complex or highly encoded formats that require substantial byte or even bit manipulation to interpret. Collect all necessary protocol for decoding file formats or any other particular low-level task into an object specifically designed for the purpose. Do so even if you might otherwise spread it around several other objects. Once you have done this you are ready to begin testing and refining your objects [Elegance through Debugging].

We have completed approximately ten patterns, have sketched out 20-30 more, and expect our finished pattern language to contain about 100-150 patterns. Our initial success using a pattern language for user interface design has left us quite enthusiastic about the possibilities for computer users designing and programming their own applications.

References

[Alex77] Christopher Alexander et. al., *A Pattern Language*, Oxford University Press, New York, 1977.

[Cunn87] Ward Cunningham and Kent Beck, *Constructing Abstractions for Object-Oriented Applications*, CR-87-25, Computer Research Laboratory, Tektronix, Inc.

---

ward@c2.com