The Road, Christopher Alexander, and Good Software Design

by
Linda Rising

I started up the mountainside while I was recovering from a broken collarbone, the result of a biking accident. I found myself, after three weeks of sitting, shoulder full of sharp pain, resigned to forgetting about my bike for awhile, but still yearning for some kind of activity. With only one working arm, I realized the answer to my limited choices lay in a near-by mountain preserve. It was a short climb and asphalt paved all the way but as I started out, head high, arm in sling, reveling in the sun, I saw the paved road as a friend, allowing me to travel up the mountain without fear of tripping on a rough surface or slipping on loose rocks. My fellow climbers were of all ages and abilities. There were young, fit specimens with heavy loads on their backs, no doubt preparing for some arduous trip. Others were of lesser talents though surely equally determined. We all struggled together to meet the same goal, each at his own pace.

It was that road I thought back to as I sat mesmerized by the presentation of building architect Christopher Alexander at OOPSLA '96. Like many others around me, I was curious, hopeful, excited, and incredibly alert, for the early morning keynote speech. Many of us had read some or all of Alexander's works. Many of us were involved to some degree in the "patterns movement," which had sprung from Alexander's insights. We wondered what he would say to this football-field-sized room full of software people.

The Quality Mystique

By way of introduction, Jim Coplien said that Alexander has had more influence on the object-oriented software development world than any other person outside the community. Alexander has an impressive list of credentials and has written several books, the most famous being A Pattern Language [Alexander77] and The Timeless Way of Building [Alexander 79]. These books are about patterns, yes, but they are also about something else he calls, the "quality without a name." This unnamed quality is the heart of what patterns are all about. As Alexander describes it:

"There is a central quality which is the root criterion of life and spirit in [all things]. ...The search which we make for this quality, in our own lives, is the central search of any person, ... It is the search for those moments and situations when we are most alive."

In Doug Lea's excellent critique of Alexander's work [Lea94]:

"Alexander's central premise, driving over thirty years of thoughts, actions, and writings, is that there is something fundamentally wrong with twentieth century architectural design methods and practices. ... Alexander illustrates failures in the sensitivity of contemporary methods to the actual requirements and conditions surrounding their development. He argues that contemporary methods fail to generate products that satisfy the true requirements placed upon them by

individuals and society, and fail to meet the real demands of real users, and ultimately fail in the basic requirement that design and engineering improve the human condition."

At OOPSLA, Alexander kept referring to this notion or quality using the expressions "living structure," and "wholeness." He said he wondered if all the patterns activity in the field of software development had really made people's lives better. Did they feel more whole, more alive, when they were using patterns. Do software patterns really make programs more alive, more whole, or is the software community just using the pattern form for documentation?

Certainly all this philosophy seems far removed from our daily struggles with deadlines and requirements changes but Alexander truly is addressing these issues. He is describing living structure as being able to grow in the face of unpredictable events. That sounds like changing requirements to me! His search for wholeness is obviously a meaningful goal for software.

As Alexander talked of wholeness, describing the difficulty of measuring it, I recalled my discovery of the road. How good it felt to be among those moving, active, living souls! How we all shared something - a smile, a nod, a greeting, pleasantries we never would have shared had we passed one another on the street. How clean the path was, while below, the litter piled up. Somehow that road made us all feel happier, more human, more alive. Could that be measured?

Measuring Information Hiding

While working on my Ph.D. research a few years ago, I tried to measure something equally elusive, information hiding. My fascination with this topic began when I was trying to learn Ada, some years earlier, by doing maintenance, a common learning technique for programmers in any language. My assignment was to take a program of about 15,000 lines of code and modify it for a different environment. I began by printing the modules, each in a different file, and sitting on the floor of my cubicle, surrounded by the files. It never occurred to me that, being new on the job, others might walk by and wonder why I was already on the floor! In fact, my new boss did just that, saying he wished he could sit on the floor with any assurance that he would be able to get up!

I studied the program, with a list near-by of file names and module names. To my surprise, I was always able to find exactly the file I needed whenever I would ask a question, for example, "I wonder where this particular activity takes place?" Determining the kind of change I would have to make for each activity was also surprisingly easy. Two answers came to mind to explain the easy, familiar way I was able to adopt with this non-trivial program: (1) I'm pretty smart! and (2) maybe it's Ada. I've been trying to solve hard problems too long to go with the first and after I had tried maintenance assignments on other Ada programs, I also discarded the second. I was truly intent on finding the answer, since my work as a maintainer and Jill-of-many-trades was only a part-time consulting activity for Fridays and vacations. My "real" job was teaching in a near-by university. I wanted to teach the secrets of this amazing program to my students. What could I carry back to the classroom? I never did answer that question directly. Instead I

physically carried the program back and used it as an example in a software engineering class [Rising89].

I started to learn more about design. I attended the new offerings at the Software Engineering Institute for university staff. I paid my own way to attend one of the last classes at the Wang Institute taught by David Parnas. Parnas had written about information hiding. To most software folks, this means "hiding information," a natural assumption! Really, there is much more to it. Parnas was talking to programmers who worked in languages like FORTRAN, COBOL, C, Pascal. He kept referring to a "module" and to most, this conjured up an indistinct picture. They wanted to use a procedure or function to clarify the examples, but obviously this wouldn't work. I think most gave up and settled on the phrase "information hiding" in its most basic interpretation. If object-based or object-oriented languages had been more in vogue in the '70s things would have been easier. Module would have translated to "package" or "class" and everyone would have said "Aha!" Unfortunately, by the time these OO constructs became popular, the over-simplification of information hiding had already taken place.

The Thing Being Hidden

I wish I could say that I made great strides in my Ph.D. research to help clear up this problem. I feel that I improved my own understanding of information hiding by measuring, in a rather weak way, what Parnas really wanted. The key, as I discovered, in my research and in my experiments, was the thing being hidden, a single design decision or "secret." Information hiding is hiding a single design decision behind a minimal interface. The minimal interface or the "hiding" part is a worthless facade without the intelligent contribution of the single design decision. In addition to the statistics and the requisite reams of difficult to decipher prose, I wanted to write the following stories about what I learned about information hiding.

The first story came from an experiment where I used a class of graduate students and a couple of versions of an Ada program [Rising93]. The students were divided into two groups and given a maintenance assignment, to be performed first on one version of the program and then on the other version. The two versions represented "good" or "bad" design as far as my information hiding metric was concerned. The two groups performed these in different orders. After the experiment had been completed, one of the graduate students said, "Now I really understand what information hiding is all about!" It is my deeply felt belief that the entire group could have said this. They really learned something. They "saw" the power of information hiding when they tried to do the maintenance assignments on the two versions of the program. It's such a simple thing, really. These students could all define information hiding. They had all read the classic paper [Parnas72], but until they met it face to face in an environment where someone pointed out the road to travel on and said, "Here, look at this," they really didn't grasp the concept.

Considering the Whole Environment

I thought of this while Alexander was talking about objectively measuring the wholeness of a street. He suggested that we look to see the effect the street has on people and ask them whether they feel "more whole." When someone points in a direction, when they walk down that street, they respond, they learn, they feel more alive. I thought of the road and my experiments with

information hiding. What I felt and what I learned, and what Alexander was describing - are we talking about the same things?

One more story from my research, again, an experiment, with another group of students. Again, they were divided into two groups. Again, two versions of the program, one "good" and one "bad." This time, they only performed the modification on one version but they had to complete the assignment in one session and couldn't leave until they had finished it. Once handed in, the good programs showed that the modification was performed but nothing else was touched. In the bad versions, many other changes were made, in addition to the required modification. I thought this was remarkable considering the time constraint. In some cases, the bad program had been completely redesigned.

I related this to my earlier experience - now they understand what information hiding is all about but I wanted to add something else. I felt that the students were reacting to something in the good program. They saw that it was good -- and they left it alone. They saw the bad program and felt compelled to improve it. All of us who have written software have this in us, some residue from an engineering background: we try to make the world a better place. Could I call whatever they were reacting to that singular quality without a name?

I agree with Richard Gabriel when he talks about information hiding and software "habitability," saying:

"Habitability is one of the most important characteristics of software. It enables developers to live comfortably in and repair or modify code and design."

I think those of us who have spent any time walking around in masses of code, can hold this up as a concrete, worthwhile, objective goal. If Alexander were working as a programmer, he would probably respond to this goal as a close relative of the quality without a name or wholeness. As Gabriel describes it:

"What is important is that it be easy for programmers to come up to speed with the code, to be able to navigate through it effectively, to be able to understand what changes to make, and to be able to make them safely and correctly."

That's exactly what happened. The students working with good programs, got in, performed the required modification, and got out. It was easy and I think they felt good about it. The students working with the bad programs, wandered around, performed modifications of all sorts, trying to improve the bad design, and in many cases, made things worse. It can't have been very satisfying.

After the experiments, I applied my information hiding metric to the Ada software I described at the beginning of this article. You're probably thinking that it was a good program and that the modules had perfect scores for information hiding. The answer is, interestingly enough, they didn't. Some of the modules or packages were good and some were not so good. But when the behavior of the entire program was examined, each module was treated by the rest as though it were perfect. Even though some interfaces were not minimal, for instance, the program never

took advantage of that and instead played by the rules, refusing to touch global data or violate information hiding. This isn't a good way to design, of course, it leaves the program open to maintainers who might decide it's easier to break the rules -- since it is possible to do so -- and use global data directly instead of calling an access operation. But, even though I was just learning the language and just learning about design, I didn't do those things, I left the good intentions intact, instinctively, I suppose. I just left it alone.

On Not Being Afraid

Here are some things Richard Gabriel says about software that possesses Alexander's quality without a name [Gabriel96].

"Its modules and abstractions are not too big if they were too big, their size and inflexibility would have created forces that would overgovern the overall structure of the software; every module, function, class, and abstraction is small and named so that I know what it is without looking at its implementation.

If I look at any small part of it, I can see what is going on, I don't need to refer to other parts to understand what something is doing. This tells me that the abstractions make sense for themselves -- they are whole.

If I look at any large part in overview, I can see what is going on, I don't need to know all the details to get it.

Everything about it seems familiar.

I can imagine changing it, adding some functionality.

I am not afraid of it, I will remember it."

As I read through this list, I thought back to my maintenance assignment, where I sat on the floor, wondering why things were so easy. You are right, Dick Gabriel. The things in your list hold true - I was not afraid of it and I will always remember it.

I kept wondering, throughout Alexander's speech, how people around me were reacting. I realized that Alexander was a building architect addressing nearly 3000 software developers. These people were typically from some kind of engineering background, what were they thinking about all this talk of wholeness and living structure? At the end of the speech, I was surprised and thrilled to see that people were standing all around me, applauding. I can't recall another standing ovation at OOPSLA! No wonder I enjoy working with software people! They are smart; they work hard; they care about what they do. They care about how their software affects the lives of others, whether they are end users or maintainers. They care about making products that have the quality without a name.

We walked thoughtfully out of the ballroom, down the hall to the book displays. I stood in line for an autograph. When I finally reached the head of the line, I told Alexander my story and

asked him if the students who left the good programs unchanged were perhaps responding to a feeling of wholeness in the design. He smiled broadly and said, "I like it!" He could understand both the engineering compulsion to make improvements and the reaction of a designer to something good, the developer's decision to leave it alone. Maybe building architecture and software architecture do share some measurable qualities of wholeness. Maybe they can both have the quality without a name. I'd like to think so.

References

1. [Alexander77] C.A. Alexander, et al, A Pattern Language. Oxford University Press, New York: 1977.
2. [Alexander79] C.A. Alexander, The Timeless Way of Building. Oxford University Press, New York: 1979.
3. [Gabriel96] R.P. Gabriel, Patterns of Software. Oxford University Press, New York: 1996.
4. [Lea94] Doug Lea, "Christopher Alexander: An Introduction for Object-Oriented Designers," Software Engineering Notes, Vol. 19, No. 1, pp. 39-46.
5. [Parnas72] D.L. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules," CACM, Vol. 15, No. 12, December 1972, pp. 1053-1058.
6. [Rising89] L. Rising, "Removing the Emphasis on Coding in a Course on Software Engineering," SIGCSE Bulletin, Vol. 21, No. 1, 1989, pp. 185-189.
7. [Rising93] L.S. Rising and F.W. Calliss, "An Experiment Investigating the Effect of Information Hiding on Maintainability," 12th Ann. Int. Phoenix Conference on Computers and Communication, March 1993, pp. 510-516.

"The Road, Christopher Alexander, and Good Software Design" appeared in Object magazine, March 1997, pp. 47-50.