Foreword

A year or two ago, I was astonished to get several letters from different people in the computer science field, telling me that my name was a household word in the software engineering community: specifically in the field of object-oriented technology. I had never even heard of object-oriented programming, and I had absolutely no idea that computer scientists knew my work, or found it useful or interesting; all this was a revelation to me. It was a pleasant revelation, but one that I did not really grasp at first; nor did I think much about it. I assumed the people who wrote to me were exaggerating anyway, out of politeness.

Then, one of these people, Marc Sewell from IBM in Atlanta, came to see me and told me much the same, to my face, in a discussion over coffee. Naturally, I assumed he too was exaggerating. When I expressed my surprise and doubts about the depth of this "alexandrian patterns movement," he told me that in any given issue of *The Journal of Object-Oriented Programming*, there was almost certain to be some mention of my name. To prove it the next day he came with the current issue of The Journal of Object-Oriented Programming. There was in it, an article by Richard Gabriel, the essay that appears in this book as the chapter entitled "The Bead Game, Rugs, and Beauty."

I sat down to read the article; and for the first time, became truly interested in this connection. What was fascinating to me, indeed quite astonishing, was that in his essay I found out that a computer scientist, not known to me, and whom I had never met, seemed to understand more about what I had done and was trying to do in my own field than my own colleagues who are architects.

Indeed, a cool factual appraisal or summary of my lifelong struggle with the problems of what to do in the field of architecture, has rarely been written objectively in the architectural literature. Architects, many of them agonizingly tied to a field which does not work, are mentally and emotionally tied up in the problems of the discipline, are often shocked by what I have said (either because it makes

them angry or because it makes them ecstatic), and have therefore rarely given large-scale cool appraisals of what I have written. Principally, I think this is because what I have to say, once taken seriously, has such enormous basement-shaking results for architecture that it irrevocably changes the field.

Yet here in Richard Gabriel's essay, far away from the internecine struggles of architecture, and without the sense of panic that so often accompanies reviews of my work in the architectural press, was sober stuff, written by a person who clearly understood it profoundly, and had taken the trouble to make himself familiar with a great deal of what I have done and written.

I found that the scientific and artistic problems I have described in my work, are being assessed, reported, without bias or prejudice, just as a matter of fact, with failures and successes given equal weight, and with the same feelings that I myself have about the task in hand, the experiments, the attempts, what works, what doesn't work—with discussion of what works and what doesn't written in rather plain English. It was tremendously refreshing, and stimulating. I was by now astonished and delighted.

That was about a year ago. Then, out of the blue, Bill Zobrist, an editor at Oxford University Press, sent me this book and asked me to read it and comment on it.

Now, I am on different ground. Suddenly the tables are turned, and I have to struggle to make out what is actually being *said* in the field of object technology and software engineering. Can I understand it? Within my limited understanding, does it make sense? Is the analogy, metaphor, or extension, from architecture to programming legitimate? Suddenly, from being reviewed, I became the reviewer.

In architecture, the question, the question I have been asking is very simple: "Can we do better? Does all this talk help to make better buildings?"

Are the questions being raised by Richard Gabriel equally straightforward? I think they are not. His questions, though in simple words, are not only about this kind of programming. He seems, to me, to be trying to jolt the software engineering community into an entirely new state of awareness, trying to create the possibility of a new field, more elevated, more marvelous, without knowing whether this is possible, because it has never been done before.

In this sense, as he describes himself, he is a Dr. Johnson, a "critick," not necessarily a practical man, but a goad, a spiritual leader, a man who sees possibilities and the glimpse of some distant promised land in software engineering.

But still a fundamental question of practicality must lie at the forefront. Does all this thought, philosophy, help people to write better programs? For the instigators of this approach to programming too, as in architecture, I suppose a critical question is simply this: Do the people who write these programs, using alexandrian patterns, or any other methods, *do they do better work?* Are the programs better? Do they get better results, more efficiently, more speedily, more

profoundly? Do people actually feel more alive when using them? Is what is accomplished by these programs, and by the people who run these programs and by the people who are affected by them, better, more elevated, more insightful, better by ordinary spiritual standards?

Here I am at a grave disadvantage. I am not a programmer, and I do not know how to judge programs. But, speaking only about what appears in this book, I must confess to a slight— reluctant—skepticism. I have not yet seen evidence of this improvement in an actual program. Of course my ignorance is such that I would not have good instincts, at first anyway, about a given bit of code, not even enough to be able to say "This is a beautiful program, this one less so." I do not therefore ask these probing questions in a negative or hostile spirit at all. I ask them, because I hope, and believe it may propel readers of this book, programmers themselves, into trying to do better. But I cannot tell, as yet, whether the probing questions asked in this book, will actually lead to better programs, nor even what a better program *is*.

In my life as an architect, I find that the single thing which inhibits young professionals, new students most severely, *is their acceptance of standards that are too low*. If I ask a student whether her design is as good as Chartres, she often smiles tolerantly at me as if to say, "Of course not, that isn't what I am trying to do. . . . I could never do that."

Then, I express my disagreement, and tell her: "That standard *must* be our standard. If you are going to be a builder, no other standard is worthwhile. That is what I expect of myself in my own buildings, and it is what I expect of my students." Gradually, I show the students that they have a *right* to ask this of themselves, and *must* ask this of themselves. Once that level of standard is in their minds, they will be able to figure out, for themselves, how to do better, how to make something that is as profound as that.

Two things emanate from this changed standard. First, the work becomes more fun. It is deeper, it never gets tiresome or boring, because one can never really attain this standard. One's work becomes a lifelong work, and one keeps trying and trying. So it becomes very fulfilling, to live in the light of a goal like this.

But secondly, it does change what people are trying to do. It takes away from them the everyday, lower-level aspiration that is purely technical in nature, (and which we have come to accept) and replaces it with something deep, which will make a real difference to all of us that inhabit the earth.

I would like, in the spirit of Richard Gabriel's searching questions, to ask the same of the software people who read this book. But at once I run into a problem. For a programmer, what is a comparable goal? What is the Chartres of programming? What task is at a high enough level to inspire people writing programs, to reach for the stars? Can you write a computer program on the same level as Fer-

mat's last theorem? Can you write a program which has the enabling power of Dr. Johnson's dictionary? Can you write a program which has the productive power of Watt's steam engine? Can you write a program which overcomes the gulf between the technical culture of our civilization, and which inserts itself into our human life as deeply as Eliot's poems of the wasteland or Virginia Woolf's *The Waves*?

I know Richard Gabriel opens these kinds of doors. I feel, blowing through these pages, a breeze, an inspiration which could begin to make a programmer ask herself these kinds of questions, ones that reach for the stars.

But so far, I do not yet see the programs themselves to fulfill this promise. So far, there is still the danger that all this paraphernalia, all this beautiful work, thought, and inspiration is only marginal comment on an activity which is still static, still not actually, *as a program*, really better.

In Richard Gabriel's next book, I would hope to see examples of programs which make you gasp because of their beauty. And I would hope for a growing knowledge, in the field of software engineering, of what this means.

Perhaps too, a knowledge more widespread in our culture, so that people outside the field, lay people like me, could also begin to grasp the beauty of programs, could have some idea of what it might mean . . . and would above all, feel helped in their lives, on the same level that they are helped by horses, and roses, and a crackling fire.

That—I think—has not happened yet.

Will this book make it happen? This is the critical issue above all, our ability to make real improvement in the geometry of buildings, in the geometry of code, in the quality of buildings, and in the quality of programs.

As I reached the end of *Patterns of Software*, I realized that my story as told by Richard Gabriel—was incomplete in a number of important ways, which may have direct bearing on the computer scientists struggling with just these questions.

Richard Gabriel focuses, very much, on *unsolved* problems, on the struggle and the path to almost ineluctable difficulties in architecture. He does not comment, perhaps enough, on the fact that these problems are solvable in practice, in fact are *being* solved right now. The geometry of life, in buildings, which I wrote about for 25 years, in order to attain it, is finally being attained, just now.

That is of crucial importance, because if the analogy, or parallel, between architecture and software engineering that he has drawn in this book, has validity, then the fact that it is solvable, must have a parallel too. If the parallel exists, then the questions are not only inspiring, but there really are programs, code, etc., which have these nearly magical qualities that breath life. And programs and code with these qualities are attainable, *now*, in our lifetime, as a practical matter in the

world of programming. This is a stunning conclusion, one which Richard Gabriel has not sufficiently emphasized.

In order to better understand how these problems might be solved in software engineering, we might look at where Richard Gabriel's examination of my work stops short and at the remainder of my work, particularly, the progress my colleagues and I have made since 1985. It is in this time period that the goal of our thirty-year program has been achieved for the first time. We have begun to make buildings which really do have the quality I sought for all those years. It may seem immodest, to presuppose such success, but I have been accurate, painfully accurate in my criticism of my own work, for thirty years, so I must also be accurate about our success. This has come about in large part because, since 1983, our group has worked as architects and general contractors. Combining these two aspects of construction in a single office, we have achieved what was impossible when one accepts the split between design and construction. But it has come about, too, because theoretical discoveries, considerably more potent than the pattern language have supplemented the power of the patterns, and the way they work, and their effectiveness.

In 1992 a pair of articles appeared that show, in short summary form, what can be achieved by these ideas when you bring together the roles of architect and builder, within the framework of the ideas of *A Pattern Language* and *The Timeless Way of Building*.*

The articles describe a number of my building projects that have indeed succeeded; they are both large and small, and include both private and public buildings. The first article gives concrete glimpses of material beauty, achieved in our time. Here the life, dreamed about, experienced in ancient buildings, has been arrived at by powerful new ways of unfolding space. These methods have their origin in pattern languages, but rely on new ways of creating order, in space, by methods that are more similar to biological models, than they are to extant theories of construction. Above all, they reach the life of buildings, by a continuous unfolding process in which structure evolves almost continuously, under the criterion of emerging life, and does not stop until life is actually achieved. The trick is, that this is accomplished with finite means, and without back-tracking. The second article describes the nature of the social process I believe is needed in the design-construction business to get these results; it is a kind of Hippocratic oath for the future. The second shows what kind of social and professional program may be needed to change things effectively in the world. If anything similar is

* Ziva Freiman and Thomas Fisher, "The Real Meaning of Architecture," *Progressive Architecture*, July 1991, pp. 100–107, and Christopher Alexander, "Manifesto 1991," *Progressive Architecture*, July 1991, pp. 108–112.

needed for computer programmers, it would be fascinating. Both these articles may have a bearing on the way software people understand this material.

A full description of all these new developments, together with a radical new theoretical underpinning, will appear shortly in *The Nature of Order*, the book on geometry and process which has taken more than 20 years to write, and is just now being published. The book, being published by Oxford, will appear in three volumes: *Book 1: The Phenomenon of Life*, *Book 2: The Process of Creating Life*, and *Book 3: The Luminous Ground*. These three books show in copious detail, with illustrations from many recently-built projects all over the world, how, precisely how, these profound results can be achieved. What is perhaps surprising, is that in these books I have shown, too, that a radical new cosmology is needed to achieve the right results. In architecture, at least, the ideas of *A Pattern Language* cannot be applied mechanically. Instead, these ideas—patterns—are hardly more than glimpses of a much deeper level of structure, and is ultimately within this deeper level of structure, that the origin of life occurs. The quality without a name, first mentioned in *The Timeless Way of Building*, finally appears explicitly, at this level of structure.

With the publication of *The Nature of Order* and with the mature development of my work in construction and design, the problems that I began to pose 35 years ago are finally being solved. There are immense difficulties, naturally, in implementing this program throughout the field of architecture and building. But the feasibility of the whole matter, and the extent to which it is well-defined, can, I think, no longer be in doubt. What is most important is that all this can actually be *done*. Buildings with these qualities, can be made, in our time, within the context of the modern age, using modern and hypermodern techniques. That is the prototype fact, which must, perhaps, appeal to those in software engineering, who hope to arrive at similar results within their field.

I am very sorry that Richard Gabriel and I did not meet, and that this material was not available to him, because I believe that he wants our quest to have succeeded, at least succeeded better than it seemed to him it had as of 1985 when we were just beginning to see the last part of the way. Because I get the impression that road seems harder to software people than maybe it did to me, that the quality software engineers might want to strive for is more elusive because the artifacts—the programs, the code—are more abstract, more intellectual, more soulless than the places we live in every day.

Once again, for the readers of *this* book, the question remains, whether this—the solution of the architectural problem—like anything else in architecture, has a true parallel in the field of software engineering.

I do find the vision which Gabriel summons up, the possibility of a world of computer programs which really do meet the Zen-like conditions that I have brought to light, quite fascinating in their implications for the world.

Although, by now, we all experience computer programs—indirectly at the very least— and benefit from them, the vision of a technical world out of control, soulless, in which we are merely digits, still looms large, and for some is getting larger. It has frightened the core of modern man. Thoughtful people wonder, no doubt, whether humanity can be regained or maintained.

If the heart of human existence, what matters most deeply to man, woman, child, really can find its way into computer programming, and into the programs, and into the meanings of those programs, and into the actual code and substance of those programs, and into their effects—then the future world will be changed immeasurably.

And if that happens, Richard Gabriel must take enormous credit for his courage in writing such a crazy and inspiring book, based on the work of a visionary drunk in God, outside his field and outside the field of his readers. I should like to take my leave of him, and you, and salute my friend, whom I have never met, and hope that his wish is fulfilled, and that looking back from the year 2100 he may be known, in some new fashion, as a Dr. Johnson of the twenty-first century.

Berkeley, Calif.
May 1996

Christopher Alexander