Ian Simmonds
41 Cedar Street - First Floor
Dobbs Ferry
NY 10522
USA

Home: +1-914-674 0699
Work: +1-914-784 7987

Christopher Alexander
Center for Environmental Structure
University of California at Berkeley
232 Wurster
Berkeley
CA 94720-1801

1 January 1997

Dear Professor Alexander

After listening to your keynote address at the OOPSLA conference in San Jose in October, three of us decided that we should think about what you had said rather than listen to a day full of the typical software conference *dross*, to borrow one of your words. Almost immediately we decided that we should formulate a response to your speech, although our collective motivation dissipated steadily as I took them on a tour of my "favorite places" in San Francisco.

I'm not sure whether my two colleagues or anyone else has made any kind of meaningful response to your speech. I know that I'm still not able to respond as well as I would like. However, I wanted to let you know about a couple of things that may be of interest, and may eventually even be of use to you in your own work. It is offered in the same way that you gave your talk, hoping that it may be of interest and that you might know what to do with it, and expecting nothing in return.

I also want to convey my initial thoughts in response to your suspicion that our fields ought to merge, and mention some other fields that probably also have to be included if we are to succeed.

You'll find these thoughts in the attached document: *Notes for Christopher Alexander*. They did not take more than a day to write, although it took three months of occasional thoughts and background reading to figure out what to say, and nine years of frustration in the computer industry, plus you as an example of someone saying "what's going on is appalling, and if I don't try to fix it, nobody else will" to make me consider adopting a similar outlook.

I hope that you find this useful, or at least evidence that *someone* in Computing is starting to think about issues that you might sympathize with.

Yours

Notes for Christopher Alexander

Who am I?

I don't think that it makes sense to respond to your talk without letting you know a little bit about who I am. Hopefully these initial, brief notes will give you some feeling for why I felt a need to respond to your talk.

I'm British, and consider my roots to be in Sheffield and Derbyshire. I like to think of that area as the center of the Industrial Revolution, which created the modern world, for good or bad.

Like you, I went to Cambridge and studied maths, but graduated in 1987. However, I had no idea what I wanted to do with my life. I had been good at maths in school, so I took a degree in that and was fortunate enough to end up with a first. I never felt that I tried especially hard, but I was at least diligent, and tried to understand what was going on.

However, at the end of my degree my director of studies suggested -- for whatever reason -- that I should *not* stay on to do a doctorate, and I am now exceedingly glad that I didn't. I would have found it exceedingly dull and inhuman.

Instead I started working -- incredibly reluctantly at first -- in computing. My assumption on entering computing was that it was a career for nerds. I worked in Covent Garden for a small subsidiary of GEC whose "mission" was to develop CASE tools: software to support people whose goal is to produce software; and often called Computer-Aided Software Engineering tools.

For two years in London, four in Paris, one in Toronto, and now two in New York (with IBM Research), I have been involved in various activities related to that same goal: what are the right tools to help the developer of software. These projects ranged from the small and personal, to EEC funded, multi-company, multi-country and multi-million dollar extravaganzas.

However, until very recently I felt that the search for tools to help in software development was hopeless. Moreover, I felt an increasing urge to dissassociate myself from an industry full of "Guns For Hire" (how much your phrase resonates with so many of us!) who mostly seemed to be producing systems that would put people out of work. Immediately after hearing your talk, I even thought that Architecture sounded like a better alternative!

Outside work, for the past 7 years (that surprises me, I didn't realize it was that long!) my main interests have been in looking at art, reading about art history, and getting to *know* the places that I live in. My Parisian friends often comment that I know Paris far better than they do.

I don't know how unique my concerns and experiences are for a member of the software industry, but I still feel a little out of place. However, I've recently concluded that this industry needs to go through as fundamental a paradigm shift as you are attempting -- consciously or unconsciously -- in architecture, and that I'm going to stay around and try to make that happen.

Frankly, so much remains to be done in computing that bringing end-user aesthetic considerations (small or large) into the picture is going to require a considerable amount of subtlety and

slight of hand. Nonetheless, it absolutely needs to be done, and the total lack of sophistication on nearly all matters in this industry, as well as the lack of equivalents to Chartres, probably still remains more of an opportunity than a problem.

By the way, my favorite places in Paris include rue de Furstenburg, St Thomas Aquinus, Place St Sulpice, Place Dauphine, le Marais and, perhaps above all, St Nicolas des Champs, with its many rows of columns, and its incredible patterns of light.

Also, I went to Caius College in Cambridge, and my room in my third year was on the (English) first floor overlooking the market square. I spent many hours just staring out of those windows.

Next come the points that I wish to share, in an order that seems more or less appropriate.

Please let me know if you want to hear more about any of them.

Concern: Constructs that are in the wrong frames of reference

It's finally dawned on me just how important it is to distinguish three aspects of computing:

- developing the technologies of computing, both software and hardware
- using these technologies, and excellently
- solving the problems in an application domain (which for me is now "business", and the financial industry in particular).

The concern is that -- to a very large extent -- all three sets of issues are reasoned about in terms of a single set of very much technology-related constructs. This means that people are only encouraged to be rigorous when they use constructs that are only really adapted to the physical laws of computing. There are very few constructs -- if any -- that are rigorous enough and well enough adapted to support reasoning *in terms of the "natural" laws* of problem domains such as insurance, banking, manufacturing, distribution, the travel industry or, more generally, "the work of individuals and corporations".

As for good old Object Oriented programming. I'm concluding that this seems to be an example of taking a few good, basic intuitions, and forcing them to conform to the physical laws of computing, rather than to the "natural" laws of any non-computing application domain.

I'll return to what the right constructs seem to be below. It *seems* that we might at last be making some progress.

Concern: Excessively narrow focus of most people in Computing

I hope that this title makes sense in its own right.

For five or more years I worked on projects whose goal was to make technologies that would allow the various CASE tools necessary to form a complete "software engineer's workbench" to be

used together. One of the major problems in computing is connecting together components that have been developed independently, or even to get one tool to access data generated by another tool. The storage format of any data must take into account different approaches to: understanding the application domain (ie, what the tool does and needs to store), a choice of technology (eg, which database technology), and choice of technology usage strategy (eg, how to design database schemas; different opinions on how to make space/time tradeoffs, or whatever).

We were sure that if only we supplied the right technology, then the problems would be solved, and everyone would get on with getting their tools to work together. Of course, *we* ended up being the ones trying to get *other* people's tools working together using *our* "new-and-better glue".

Most worryingly, we never spent any significant amount of time thinking about *what good tools might look like*, or *what is it that makes tools work well together*. We were far too narrowly focused, as were the developers of each of the tools.

In Toronto, I worked in a software development organization in which there were perhaps 60 separate projects, each of which was developing a piece of software technology. I won't go into any details, but needless-to-say, few of these teams had the slightest intuition of how their technology might be used until (after years of struggle -- if ever!) they had customers, and these customers started complaining about "those idiots who dreamt up this technology -- it doesn't fit with the *other* things that we use".

In the past couple of years I have been trying to understand *all aspects* of the development of software applications for the insurance industry. Finally, a holistic approach! I was involved in *requirements capture*, selection of technologies, development of technology usage strategies, application design, training of staff, and *building common understanding amongst all participants*. I have found the experience utterly incredible, and the insights I have been able to observe from a close (but nonetheless real) distance to be amazing.

Working across the *full breadth* -- I'm also involved in management and (almost) contract discussions -- of a project with a very narrow business scope has been totally eye-opening. Frankly, most of the various strands of thought of nine years in computing have woven to form a really quite beautiful -- and simple -- picture.

On *most* academics

Most academics in computing have an incredibly narrow focus, and almost no feel for what software is *really* about. And they are the teachers of the next generation!

However, I will enclose a copy of a paper by Tony (C.A.R.) Hoare of Oxford University, entitled "Unification of Theories: A Challenge for Computer Science" (attached document #1). He is concerned enough to write such philosophical papers, and we should applaud him for doing so. The most important issues for me are only touched on in passing, *but they are mentioned.*

On *many* computer practitioners

I was really pleased with something you said at OOPSLA, and equally disappointed by Richard Gabriel's attempt to apply your work to computing.

Gabriel said that your work was about "making architecture better for its users". In applying your work to software -- Gabriel continues -- we have to be careful (and here I'm paraphrasing my recollections) not to be *misled* into applying your work to those users that are end users of the systems that we develop. Rather, we should consider ourselves -- software developers -- as users since we do, after all, *live in* the code as we develop and maintain it. We should strive to make our code *habitable*, and that is where we need inspiration from Chris Alexander.

I was utterly appalled. How arrogant. Yet how typical of a member of an incredibly nervous and uncertain discipline that is long overdue for a paradigm shift.

In comparison, you talked about your amazement at how ignorant most software people are of the incredible impact that software is having on all aspects of civilization.

To that I whispered my agreement.

Making code habitable, indeed. Yes ... But ...

Concern: *BEWARE OF TOOLS THAT OFFER GRATIFICATION TO NON-THINKERS*

Bulletin boards on the Internet are a well known example. They seem to cry out:

> post a reply immediately! there's a constant, immediate and large audience for your ego! no quality controls! no reviews to pass! almost complete anonymity! it's better to respond quick and dirty! and now!

Reasoned argument can only occur when people are obliged to think. Many of the tools of the Internet encourage people to act instantly, to reply without thinking. Consequently, the signal to noise ratio is tiny ...

Be very careful about the use of tools. Often the tool that allows the Master to immediately and confidently take perfect actions based upon fully developed intuition is placed in the hands of thousands (millions) of novices and frauds will act *with even greater speed and dexterity*, but with *absolutely no quality*.

Sometimes I think that *utterly atrocious response times* support clear thinking more than any other feature of a CASE tool.

Concern: Absolutely no clue about how to make tools (software applications) that are good

I don't suppose you stayed around to hear it. I wouldn't have, in your shoes. However, the final keynote at OOPSLA -- by Larry Constantine -- really amused me.

Constantine is pretty important in the field. He played at least a minor role in bringing a first round of discipline into software. More recently he has been concerned with how to develop better user interfaces.

His goal was to present the *latest* major advance in *making users happy* (also known as *user interface design*), but he started by reviewing the *"major paradigm shifts"* in user interface design in the past several years:

- first, there was a focus only on *the code of the system*
- second, a focus on *what functionality and features the system should have*
- third, let's include a system component called a *user interface!*
- fourth, let's try to make this interface *"user-friendly"*, and publicly proclaim just how important that is to us
- fifth, let's be *user-centered* -- get the user to review the interface at least once
- sixth, let's be *user-centric* -- insist on extremely high levels of user participation
- seventh -- and *today's great innovation* -- let's be *usage centered.*

They've finally noticed that if something exists primarily to be used, then *first understand the usage, then design to support that usage*. Get people to tell you about how they will *use* the system, and *what it means for the system to be usable* and *before* you produce a design. Don't just ask the users to evaluate a design inspired by (who exactly?), and expressed in terms for which they have no developed sense of quality.

Well, I was amazed -- after all these years they were doing what my intuition has *always* led me to do.

However, it did encourage me to put my intuition into words: "practicality every thing that you have to think about when designing software is about using something: *use of the application* by real people, *use of technology* by the application, and by specialists in technology usage".

Solution?: Emerging tools and constructs that *seem* to be in a better frame of reference

In the past year I've had the pleasure of working with a guy called Haim Kilov. He has spent his own career attempting to produce better systems. He and his dissident colleagues were responsible for developing the first ever extended relational database in the Soviet Union, while officially "shunned" and "exiled" to the Latvian Leather Research Institute.

Before joining IBM, Haim worked at Bellcore -- one of the Telecom industry's leading corporate research centers. Their they struggled with structuring incredibly complex sets of requirements, including those governing telephone "switches". A switch is effectively a fully fledged computer, and so has an enormous number of extremely complex requirements.

The results are presented in the book "Information Modeling an Object-Oriented Approach", published by Prentice-Hall, which Haim jointly wrote with James Ross. Haim and I summarized our first attempts to apply the ideas with an IBM customer in attached document #2.

Haim's most recent and -- I feel -- most lucid explanation of these concepts is in attached document #3. However, in my opinion and words, the key breakthroughs are the following:

- favor understanding by all, rigor, and completeness over everything else

- when considering external (business) requirements, completely ignore *all* technological factors and constructs. Assume that the constructs useful for *designing* a software application will be *absolutely inappropriate* for formulating the requirements
- concentrate on things that are *absolutely essential to the business*, and worry about more *tactical* things later, such as what will be automated, outsourced, done in a cubicle, or whatever
- capture the essentials *rigorously* (rather than *formally*, using mathematical symbols that are frightening to business people and, especially, to software people)
- capture the essentials *declaratively*, in terms of operation contracts (what happens expressed in terms of pre- and postconditions, and *no suggestion of* how *it might happen*) and invariants (what remains true no matter what operations are performed, except, perhaps, temporarily as part of completing an operation)

You should be at home with an emphasis on invariants. Haim certainly jumped up and down when he saw *you* using that word.

Now, this is where it gets interesting, and may bare *some* relationship to your Nature of Order:

- express all rules (pre- and postconditions and invariants) in terms of *ways in which things may be classified* rather than *intrinsic* "properties" or "types" of thing

By analogy, although an alcove has a certain number of intrinsic properties (a certain size, a certain relationship to other centers, a certain level of intimacy), people *think* and formulate rules more generally than this (such as: a good place to quietly reprimand a child; a good place for an intimate proposal; a good place to sell life insurance; and so on). You can express a lot more information a lot more succinctly if you express it in terms of the classifications into which (perhaps very different kinds of) things fall at different points in time.

Incidentally, I suspect that an architectural "center" feels good if your unconscious recognizes that it can be "used" in a wide variety of ways.

A few other interesting points:

- the properties of one thing may be (at least) partially determined by the properties of another thing. In such cases, emphasize that such a relationship exists and that it is about property determination, and worry about the details of what is determined and how at a later stage.
- some things act as abstractions for collections of other things. In such cases, some of the properties of the abstraction are determined (in whole or in part) be some of the properties of its parts

Finally:

- one classification may be a subclassification of another classification
- moreover, a set of subclassifications may *partition* a parent classification (for example, if we concluded that a telephone booth could *only* be used for making 'phone calls, or an intimate conversation, *but not both at once*)
- there may be several orthogonal subclassification hierarchies for a single classification. People may be: male or female; democrat, republican or independent; managers,

technical or both; married, divorced, widowed or single; and so on. They may therefore fall into many classifications at once, and may also be reclassified by certain business operations (such as, *record marriage*).

And so on.

We call these reusable fragments of invariant *elementary specification patterns*, or *generic relationships*. These constructs are fully in the public domain -- as part of an ISO standard, no less. Others -- those more strongly related to separating an application's business content from its usage of technology -- are also emerging from this work, but are as yet inadequately presented in written form.

Some *larger* fragments of business specification -- what we call *business patterns* -- are mentioned below, and discussed in attached paper #4.

The bottom line.

Learn what the right time is to make a certain decision. Split larger decisions into smaller pieces, and choose an appropriate moment for making each fragment. Even when *you think that you know how a design decision should be made*, only make that decision when you are ready.

Learning from failures

While reading Stephen Grabow's book on you -- found last weekend in a second hand bookstore on 18th Street in New York -- I enjoyed the reference to Zen in the Art of Archery. Yesterday I came across that very book, and read it earlier today. (I still haven't finished Grabow's book -- I'm over half way through -- but I think that I got the point)

I was struck by several things in these two books, as well as things in The Timeless of Way of Building, and in your OOPSLA talk.

Firstly, all of the potential for progress that I see in computing is related to the elimination of ego from the tasks at hand. You simply can't think in terms of the application domain (eg insurance) at the same time as you think about how to exploit computer technologies to solve these problems.

Somehow my colleague Haim has stumbled on (more or less) Zen approaches to these problems -- "completely illiminate all thoughts and desires related to *implementing* a computer system while you are understanding the business that it will/might support". (I'm making this up as I go along) Perhaps he's a Zen Master in disguise? He's kind of short, and very much balding, although his addiction to extremely strong coffee seems somewhat anomalous.

Secondly, you can only truly understand why certain steps in software development are so important by understanding *and experiencing* why failure occurs if you don't follow these steps.

Thirdly, I was intrigued by Grabow's references to eastern philosophies that embrace elements of what I think of as mathematical rigor. Nameless qualities. Generative structures. Elimination of the self. And so on.

I always thought of (is it?) Godel's theorem when I heard about your experiments related to the quality without a name: every mathematical system allows correct hypotheses to be formulated which cannot be proven wholly within that mathematical system. So it seems natural to go even further and conclude that you can only learn about qualities of environmental structure that are evidentally *beyond* the reaches of current approaches by asking questions that explicitly banish all references to qualities that *are addressed* -- however partially -- by current approaches.

Fourthly, it seems bloody useful to have a clear understanding of the nature of paradigm shifts (a la Kuhn) if you feel obliged to occupy yourself with issues that may lead to such a shift.

Fifthly, I think I now realize *just how serious you are* in all of your professional activities. Serious problems require serious analysis and serious solutions. And serious propositions from a leader in one field to members (anyone!) in another field *really ought to be taken seriously*, at least by somebody.

Learning from other disciplines

Yesterday I was lucky enough to buy about 40 books -- for $120 -- from the library of a man who had recently died. The books were mostly about the philosophy of art and literature. They included titles that discuss the role in literature of:

- semiotics
- what it means to make reference to things in the real world
- narrative
- on Foucault's "Uses of Uncertainty"
- and so on.

I feel that I have built more intuition about what is wrong in computing by reading recent works on art history than books in any other field, especially books on computing. Similarly, I suspect that we can learn more about how to make people comfortable in doing tasks using computers by studying the importance of narrative structure in literature than we will ever learn by working on our own.

There are two often cited "oddities" in computing, both of which are seen to relate more to the Humanities than to the rest of computing. The first is the Apple Macintosh -- an early attempt to *thoroughly* and *holistically* apply graphic and ergonomic theory to the design of a computer *and its software*. The results has been longer lasting -- and provoked more passionate loyalty -- than practically anything else in computing, before or since.

The second is the computer game industry, where the most successful games have an extremely strong sense of *narrative*. I think that I heard someone (on Public Radio) say that a little more narrative content easily wins over a little more graphical "fizz". It is no wonder that there are successful computer games built around Norse epics, and so on.

It seems to me that many of the patterns in A Pattern Language represent "centers" corresponding to elements of narrative (entrance transitions, meeting places, communal dining, and so on), as well as those that provide enough texture to allow rich descriptions (for the places in a journey, a context where people will meet for a short time, or a long time, or for a particular kind of activity, and so on).

We are generally happier with a place that would be a good setting for a novel; and are happier with a novel set in a place that is worth describing. We are unsettled when we are able to describe a place *completely in a few words*, or when we are told that *there's nothing else there* than has been described in a few words.

Business patterns: Where *some* of your work might meet with *some* of our work

In our work to find larger, more meaningful *patterns of business* (formulated in terms of generic operation contracts and corresponding generic invariants), we have concluded that most business is constructed of (admittedly many special cases of) a relatively small number of business patterns:

- perform an assessment
- make a decision
- gather a specific piece of information
- collect "relevant information", determining what you need as you go
- determine how something should be classified
- act on something because of the way that it has been classified
- receive communications
- classify communications as solicited or unsolicited
- and so on.

Orthogonally, each of these things may be accomplished by an individual or by a group (for example, by negotiation or collaboration). In some cultures (environments), certain things just aren't sensibly accomplished through negotiation, ...

Orthogonally again, the work may require some kind of tool support, be it paper and pen, a software application, a hammer, a jet spray for quick drying concrete, or whatever. In some cultures (environments), certain things just aren't sensibly accomplished using a jet spray, ...

Orthogonally again, the work may be best carried out formally, casually, incidentally, in isolation (from interruptions), and so on. In some cultures (environments), certain things just aren't easily accomplished unless they happen casually (eg, telemarketing), ...

Orthogonally again, the work may be eased in an architectural context satisfying some specific Alexandrian pattern language. In some cultures (environments), certain things just aren't sensibly accomplished anywhere but in an alcove, ...

And so on.

How about a *whole design* for an environment that supports the settlement of parking tickets: the surroundings must be reasonably private, intimate, yet austere. While negotiation is an option (in a court), normally you just send a communication: by letter, enclosing a check, or via a credit card payment made through a computerized kiosk in suitable surroundings (for example, in a shopping mall). And so on, including much more profound considerations than that. (For example, we ought to include narrative generating properties as we design a center's relationships with sibling, enclosed and enclosing centers)

As mentioned above, our first presentation of business patterns was in attached paper #4.

Where *more* of your work might meet with *more* of my personal knowledge

If you are serious about developing software packages to allow people to "design their own houses, communities, and corporate headquarters" following your approach, then I may have some uncommon insights into the nature of tools.

These insights relate to the dangers of a tool rendering decision making *too easy*, and ensuring that a tool supports *decision making* and *refinement* (*unfolding*) rather than symbol shuffling. These are *precisely* the concerns for any kind of software tool, in your industry, mine, or for any other human endeavor.

Please be *very wary* of anyone who approaches you with the goal of developing such a software tool. If their prime motives -- either proclaimed, or apparent to you -- are based upon generating revenue or reputation, then there is a large probability that the results will not please you. The development of really good software tools is an as-yet-undeveloped art.

Apply that caution to everyone who comes knocking, including me ...

Role of philosophy, visual art and literature in the future of computing

I play a game when I come across art that is unfamiliar or uncomfortable. I assume that I must be able to have some rationale thought about it; make some value judgment.

Over time I have become more comfortable with a wider range of works of art. When I am comfortable with a work of art, I can intuitively determine *something* about how interesting it might be, and almost purely by instinct. This is an orthogonal issue to whether or not I like the works.

A few weeks ago -- after several years of encouragement from May -- I started writing short essays on (what I perhaps mistakenly call) computing and aesthetics. Each one talks about a single work of art and an analogy based upon *one or more of its properties* that enables me to take a tangential swipe at computing.

Topics, works and artists that I am considering include:

- many early portraits in America were painted by itinerant, mediocre portraitists who had failed in the London art scene. This is comparable to the mediocrity of many computer consultants and architects.

- the early work of the most gifted of early American born artists -- and especially John Singleton Copley -- was stilted, weak and mediocre in international standards, simply because they had never encountered an accomplished work of art. However, once exposed they rapidly achieved the highest international standard, of which they were more than capable. In the absence of good examples and fair, honest criticism by Masters, even the gifted in computing will produce dross. Most enterprise computing systems are *never* subject to public criticism.
- Richard Long's sculptures take the form of large, essentially two dimensional geometrical forms, such as disks and rectangles. However, they are constructed from somewhat loosely packed pieces of rock. One disk, made of chalk, shows how it is relatively simple to construct the circle with a large number of pieces of this soft rock, which is easily broken down into (very roughly) spherical pieces. Another disk, made of slate, shows how it is more challenging to construct the circle with what must be long, thin slabs. However, the slate circle -- once constructed -- is much more interesting, and has many readily visible intermediate sizes of structure: just try arranging different length sticks so that they fill a circle *without introducing* some clusters of parallel sticks. Conclusion: some constructs encourage you to produce intermediate levels of structure, others do not; when you need that intermediate structure for some purpose -- aesthetic or practical -- then you are better off using constructs that foster it, even if they take slightly longer to master.
- Toronto artist Michael Snow's art is the most intellectually stimulating that I know, amongst modern artists. He has thoroughly explored the nature of the medium of photography, and also that of film. He is internationally recognized in film theory, with only Andy Warhol considered to have achieved anything comparable related to the *nature* of the medium of film. I am unaware of any other artist who has so fully explored a medium, let alone mediums so diverse as still photography, film, holography, jazz, poetry and public monuments. In the latter case, he was responsible for a sculpture of a flock of geese landing in a shopping mall (Eaton Center, Toronto), and a statue of an audience, which somehow dominates an entire baseball park (Skydome, Toronto), by exploiting a thorough analysis of the nature of the *approaches* to the ball park. But it is his works on the nature of photography and holography that I enjoy the most.
- and many, many more.

I would be more than happy to share these (embryonic and faltering) first steps in (non-computing) essay writing with you.