Conversations with Alexander

David West, Transcendence Corporation

This is an essay about the author's interaction with ideas advanced by Christopher Alexander. The essay has three inter-related themes: Alexander's ideas and how they influenced the author's own work; Alexander's impact (or lack of it) on software development in general; and the Patterns Movement that Alexander is credited for inspiring.

ACM Reference Format:

West, D.. 2012. Conversations With Alexander. ACM 21 pages.

Our first, meeting was in 1987. The meeting place, *Notes on the Synthesis of Form* (*Notes*) - a small book with a stark black cover and white text; an edited version of Alexander's doctoral thesis twenty years earlier.

I was a graduate student at the University of Wisconsin – Madison. I had returned to school (leaving Macalester the second semester of my senior year) two years earlier (1985) after seventeen years working as a software development professional. By 1987 I had finished my undergraduate credits and completed two masters degrees, one in computer science and the other in cultural anthropology and was working on my Ph.D., ostensibly in cognitive anthropology but really interdisciplinary. I had three anthropologists, two computer scientists, a linguist and a psychologist as my dissertation committee.

Although my CS Masters was in the area of artificial intelligence, I was convinced that the computational model of cognition was, not only wrong, but silly. It seemed that the model willfully ignored 99% of cognition to focus only on the simplest elements, those that a computer could simulate.

My dissertation research was focused on devising a model of cognition that at least incorporated the role of culture. So it was that one day, while browsing in the University of Wisconsin-Madison bookstore, I picked up *Notes* and was intrigued with the chapter on "The Unselfconscious Process."

Alexander seemed to be saying that it was possible for a body of knowledge to exist outside of any individual mind – threaded throughout a culture; and, the results of using the 'unselfconsious process' were superior to results originating in professional expertise.

So I took the book home.

Alexander's description of how a body of knowledge (architectural form and function) could be dispersed into culture, via stories, myths and rituals, did indeed influence one of the ideas behind the model of cognition in my dissertation. That model, a virtual Topographic Adaptive Organism (vTAO), based on a circular, reciprocal, relationship between culture and cognition.

DOI10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

That dialog with Alexander was relatively minor in the scheme of things as it turned out. It seemed I was fated (doomed perhaps) to put my interests in cognition, anthropology, cognitive science, and altered states of consciousness aside and continue life as a software professional.

Incidentally, I studied anthropology because, in my entire professional career, I had never had a system fail for technical reasons – it was always the human, the political, or the cultural that caused problems. The last two of the three years I spent getting my graduate degrees, I was also a full-time member of the academic staff in the School of Engineering leading a software team. We built some cool stuff, ranging from a simulation of an X-ray diffractometer to software that helped measure the lenses of a cow's eye in order to fit it with contact lenses. Post graduation my career remained firmly embedded in software development – teaching it, coaching it, consulting in it, and doing it.

So, I would argue that the most important influences from *Notes* are those applicable to software development. Beginning with the problem of design.

> *"Let us look again at just what sort of difficulty the designer faces. Take, for example, the design of a simple kettle. … I have deliberately filled a page with the list of these twenty-one detailed requirements or misfit variables so as to bring home the amorphous nature of design problems as they present themselves to the designer. Naturally the design of a complex object like a car is much more difficult and requires a much longer list."* [Notes, p. 60]

Software developers, even software engineers, are designers. No matter how much we would like to pretend that we are scientists and not artists. Moreover we confront, as a matter of course, problems with hundreds if not thousands of 'detailed requirements' – uber-amorphous problems. What to do?

> *"Since we cannot refer to the list in full each time we think about the problem, we invent a shorthand notation. We classify the items, and then think about the names of the classes … Each of these concepts [name of a class] is a general name for a number of the specific requirements."* [Notes p. 61]

Given that we are all, "bears of very little brain," like Pooh we are forced to engage in aggregating, categorizing, and labeling as concepts, in order to make the problem sufficiently tractable. At issue, however, is where the concepts (names of the aggregates) come from. Alexander, in *Notes,* describes two different ways that such concepts come into existence.

In the "unselfconscious process" the concepts arise from collective experience – essentially making mistakes and having them corrected until a consensus is reached as to which requirements belong in an aggregate and how to satisfy or resolve that set of requirements. *"The right way is the residue when all the wrong ways are eradicated."* [Notes p. 50]

Each "residue," each aggregate of requirements and means of resolution, is then integrated with all other similar aggregate-resolution concepts by being embodied in myth and ritual.

The concepts arrived at in this fashion reflect "natural" aggregates. Natural meaning two things: consistent with cultural worldview, and separated at the 'natural joints'. Alexander's focus is on a collective, cultural process requiring lots of trial and error and lots of time. Plato parallels Alexander but speaks to the actions of individuals.

> *"[First,] perceiving and bringing together under an Idea the scattered particulars, so that one makes clear the thing which he wished to do … [Second,] the separation of the Idea into classes, by dividing it where the natural joints are, and not trying to break any part, after the manner of a bad carver … I love these processes of division and bringing together; and if I think any other man*

is able to see things that can naturally be collected into one and divided into many, him I will follow as if he were a god."

Both Alexander and Plato are asserting that it is possible to partition (decompose, as that word is used by software professionals) a complex, amorphous, and large-scale system into tractable subsystems in a "natural" way. The unselfconscious process yields natural decomposition.

In contrast, in what Alexander calls the "self conscious process," the concepts derive from "theory;" from an attempt to find and define general principles and formal relationships that then define the concepts. This process is biased towards the creation of concepts that are arbitrary in nature and/or the result of historical accident. And this presents a problem.

A small clarification is useful here. When Alexander used the term 'theory' he was referring to the set of ideas behind various schools of architecture – not 'scientific theories'. Although he was completely dismissive of 'architectural theory, as pure ego-driven abstraction; Alexander would be less critical of 'scientific theory'. After all, he considered himself to be a 'scientist'.

Observation, experiment, induction, deduction, and abduction – applied in a systematic way – aka, 'scientific method' – might, in specific realms yield the same 'natural joints' and therefore establish theory with 'correct' concepts.

Scientific theory is not, however, immune to being driven by arbitrary abstraction any more than architecture. This is especially true for wannabe sciences, like CS. The "cognition and computation are both instances of physical symbol systems" would be a prime example of selfconscious theory.

Given our focus on software development, more specifically application software, it is appropriate to treat the theories driving software engineering as exactly the kind of selfconscious theory derided by Alexander. And this presents a problem.

Remember that a concept is a name used to label a set of requirements. It is critical to get the 'correct' set of requirements in each defined set.

"Suppose we picture the system crudely by drawing a link between every pair of interdependent requirements … A subsystem is one of the obvious components of the system … If we try to adjust a set of variables which does not constitute a subsystem, the repercussions of the adjustment affect others outside the set, because the set [subsystem] is not sufficiently independent. [Notes p. 64]

Alexander is arguing that "good" decomposition is dependent on achieving appropriate cohesion (the right requirements collected as a concept) and minimal coupling – manipulation of variables within a concept affecting interdependent requirements in another concept.

This is interesting and somewhat remarkable as he discovered these ideas four (based on 1964 copyright of *Notes*) to six (when the dissertation that *Notes* is based on was written) years before the term "software engineering" was invented. The "structured programming" and "structured design" mainstream of software development came even later, in the Seventies. Also, this was done in a field totally independent of computer science / software engineering. It is obvious why Alexander's ideas had such an influence at the 1968 NATO conference.

"And one more thing …"

Alexander argues that qualitatively different results – concepts / aggregations of requirements into independent subsystems – will accrue from the "unselfconscious process" than from the "selfconscious process."

> *" … the procedure of the unselfconscious system is so organized that adjustment can take place in each on of these subsystems independently. This is the reason for its success. In the selfconscious situation, on the other hand, the designer … [must] decide which subsets of requirements to deal with independently.*
>
> *My contention is this, These concepts will not help the designer in finding a well-adapted solution unless they happen to correspond to the system's subsystems. But since the concepts are on the whole the result of arbitrary historical accidents, there is no reason to expect that they will in fact correspond to these subsystems.*
>
> *… as a rule, concepts are not generated or defined in extension; they are generated in intension. That is, we fit new concepts into the pattern of everyday language by relating their meanings to those of other words at present available in English.*
>
> *Yet this part played by language in the invention of new concepts, though very important from the point of view of communication and understanding, is almost entirely irrelevant from the point of view of a problem's structure."* [Notes pp. 64-67]

Alexander is asserting that problems (and by extension problem domains) are systems that have naturally occurring, and discoverable – by finding the "natural joints." Further, the unselfconscious process is organized in such a way that natural divisions between / among subsystems are revealed. And, the selfconscious process is so organized that it is difficult if not impossible to find these natural subsystems. Even worse, if by chance a natural subsystem (concept) is found, the selfconscious process mitigates against its survival because it will be inconsistent with the body of existing concepts and the theory that birthed them.

There are supporting examples in the world of software. In the "real world" you have: customers, accounts, transactions, rules, processes, and organizations. In the software world you have: algorithms, data structures, control structures, ANDs / NANDs, tuples, and relations – all of which derive from theories of computation, of data, of logic, etc.

While good programmers – for readability purposes – will employ the real world labels as metaphoric identifiers of program constructs – the 'goodness' of the program depends entirely on its consistent adherence to the theory-derived concepts.

And when a new concept, like "Object" is introduced to better reflect the natural disjunctions in the problem space, it is distorted to agree with prevailing selfconscious concepts until it becomes an 'animated data structure' with little or no resemblance to the original concept.

The problem is not limited to one word, e.g. 'object'. Tackling large amorphous systems with lots of requirement-aggregates yields a large vocabulary of concepts.

The unselfconscious process provides us with a vocabulary where each concept has meaning because it is not separated from the culture within which it was formed.

Again, a brief aside is useful here. Alexander asserts that the concepts arising from the unselfconscious process are embedded in culture and that they take a long time to become established. This would imply that anytime we faced a totally novel situation, it will take a long time for the concepts to arise and standardize.

This, in turn, suggests that software development, in novel application domains, would take a really long time. But Alexander was only partially correct about culture. Culture embedded vocabulary can arise, seemingly, instantaneously – think 'google' as a verb.

In a different essay, or at another time, I would argue that the real problem here is metaphor abuse. We have little or no choice, whenever we encounter something novel, than to employ metaphor as a bridge linking the known to the unknown. In a 'cultural' context, metaphors arise (epiphor), are found useful (diaphor) and maybe truthful (lexical term) or are not helpful and are discarded (dead metaphor). In the selfconscious process there is a third outcome – paraphor (a neologism of mine) – where the metaphor is clearly dead, but persists because it is consistent with a dearly held theory. Pylyshyn's insistence that the metaphor – mind is a computer – is a lexical term for example.

The very first work I published, in *AI Magazine*, [Vol 12, No. 1 and No. 2, 1991] was a two-part article on metaphors of mind. A fuller discussion of metaphors and metaphor abuse can be found there.

The selfconscious process provides a vocabulary where the concepts merely have definition and a syntax for manipulating the defined tokens.

There is a way that a selfconscious process can "*help the designer in finding a well-adapted solution [by] correspond [ing] to the system's subsystems."* All that is required is for the theory to be congruent with the system as a whole.

Computer science offers an example. If the system of interest is the deterministic universe of classical physics or the deterministic system of a machine; it is likely that the language, the theory, of computer science does, indeed, map to the systems subsystems. The Theory yields a vocabulary that has meaning and can be used by a programmer to create excellent software – device drivers, the software portions of RISC architectures, TCP/IP stacks, compilers, etc.

This works because the Theory is of a machine and the vocabulary it generates is of the machine and machine parts and therefore allows programmers to write excellent software, as long as the system at issue is a machine.

Alexander's problem was two-fold. First, he wanted more than buildings that did not fall down, had correct geometry, and were ergonomic; he wanted buildings to have meaning, to have 'fit'. To have what he years later would call QWAN, or the Quality Without A Name. Second, his system of interest was Nature, and nature is most definitely not a machine. (Unless of course your are follower of Mach and Einstein.)

Software developers, anyone tasked with writing anything from an operating system to domain specific applications share Alexander's problem. No machine grounded theory is capable of providing the concepts, and vocabulary, that would lead one to discover QWAN, nor would they help if the system at hand is qualitatively different – complex instead of deterministic.

The application developer looks at her problem domain, a natural world, and fails to see any functions, algorithms, linked lists, or tuples. The theory and vocabulary of computer science does not map to natural joints or independent subsystems in the real world. The theory does not fit the reality.

You could try to force reality to fit your theory. You could abstract away every aspect of a human customer that is not consistent with the 'record structure' concept in your computer science theory. You could even name the record structure Customer, metaphorically relating the two. But doing so perverts metaphor and creates artificial (in the most negative sense of that word) solutions.

The solutions arrived at in this manner are really bad. Despite all its imperfections the annual "Chaos Report" is not wrong in its documentation of how bad. But instead of recognizing failure as a misfit of theory, the theorist has the temerity to assert, usually indignantly, that the theory is obviously correct and the stupid developers were insufficiently trained in the theory and failed to use it properly.

Another example. Most of the AI that I was studying was based on a computer inspired theory that cognition was nothing more than the formal, (according to mathematical and logical rules), manipulation of symbols, (precisely and unambiguously defined). Words are simply symbols - ripped from the mouths of people, stripped of their context and meaning, and warped into mere data.

LDA, Linear Discriminant Analysis is an obscenely mathematical exemplar of this kind of theory. Natural language uses grammar in order to form meaningful sentences – relationships between the words. But users of language often contravene 'official grammar' and even 'official definition' of words to create meaning where grammatical associations cannot. LDA substitutes statistical algorithms for natural grammar and common usage to form an artificial set of relationships among words.

I first encountered LDA in marketing, but it has been used in many other areas, including machine learning. A marketing company wanted to better target customers for their products – a pretty common desire. A data set was created and the algorithms applied and a model of the sort, 'people who respond to these words in the promotional literature for these cruises will be interested in these products.'

A, somewhat naïve, young woman responded to the words but was dismayed to suddenly be inundated with email offering various homosexual products and services. It seems that the words in the data set were euphemisms, clearly understood by those taking the cruises, but totally misinterpreted, both by our young woman and by the LDA software.

As mentioned earlier, at the time I was digesting *Notes*, I was simultaneously pursuing my graduate degrees and working as head of a software development team in the School of Engineering.

My day job involved coding in C and using version 0.9 of the Windows SDK. Aside – we had to compile over 10,000 lines of code, all but 150 or so from libraries, in order to display 'Hello World" in a window – end aside. We also followed formally defined procedures – structured method – to guide our development efforts.

Object-oriented programming was rapidly becoming mainstream and I obtained my first copy of Methods (for $99 on a 5 ¼ inch floppy disc) and the two "Goodies Packs" from Dave Thomas' team at Carleton (for $19 each and also on floppies) giving me access to the Smalltalk programming language on my IBM PC.

I was not and never will be an ace programmer, but I did program and, as part of my anthropology work I wrote a simulation, (including 2D graphics), of the entwined economic systems of a Tibetan Buddhist temple and its surrounding village in Methods (Smalltalk).

The contrast between developing in C and Smalltalk was illuminating. Not at the language level – all languages are Turing Machines and therefore essentially equivalent – but in the areas of design and decomposition / modularization.

Alexander spoke to me. I found his ideas about decomposition, systems, and subsystems, coupling and cohesion indispensable in how I approached the Smalltalk and Object projects in which I was engaged – but almost irrelevant to the C projects.

An Object, for me, was Alexander's 'concept', reflective of a natural occurring set of expectations distinguished from other objects at natural joints.

A C module, and later a C++ object, had nothing to do with the natural world. A C module was nothing more than an aggregate of low level instructions, which were, in turn, aggregates of primitive machine operations. Although it might be argued that C / C++ modules / classes represent 'natural joints' it must be recognized that they are natural joints of the machine – not of the real world.

Coupling and cohesion were principles by which C modules modules were discovered and defined. Alexander never used the terms, coupling and cohesion, but provided a far better explanation, in my opinion, of them than found in any of the CS textbooks I was studying.

Notes provided a touchstone with which I evaluated the various approaches to doing object design. This had the effect of pushing me into the minority camp of "behavioral objectivists" like Cunningham, Beck, Wirfs-Brock, and the 1990 version of Grady Booch.

This allegiance pushed me to the fringes of Objects where I watched with dismay as the mainstream adopted the Schlaer-Mellor, Rumbaugh, and post "three amigos" / UL / UML/ UP version of object design. The object mainstream rapidly became an example of the selfconscious process and objects lost their power and utility as they were twisted to fit the theoretical precepts of software engineering and computer science.

Abundant evidence of this descent into selfconscious-ness is seen in the arcane and obfuscating arguments like: multiple-inheritance or no; compilation or interpretation; strong or dynamic typing; do getters and setters violate encapsulation; method-X is better than method-Y because it has a formal (mathematically provable) foundation. Such discussions certainly did not lead or contribute to any of the design goals articulated in notes.

"And one more thing …"

Alexander liked pictures – diagrams, actually. A diagram to capture *'each set of requirements,"* i.e. concept.

"The invention [which we now call] diagrams is familiar to every designer.

The famous stroboscopic photograph of the splash of a milk drop is, for certain purposes, a diagram of the way the forces go at the moment of impact.

Le Corbusier's ville radieuse *is a diagram, which expresses the physical consequences of … the need for people to be housed at as high overall density, and have equal and maximum access to sunlight and air.*

The sphere is a diagram …

The texture of bathers on a crowded beach is a diagram.

… these diagrams may have either or both of two distinct qualities … they may summarize aspects of a physical structure … We shall call such a diagram a form diagram. … On the other hand, the diagram may be intended to summarize a set of functional properties or constraints … We shall call such a diagram a requirement diagram.

… in the case [where] the formal description and the functional descriptions are just different ways of saying the same things; we can say, if we like, that we have a unified description … the abstract equivalent of a constructive diagram.

The solution of a design problem is really only another effort to find a unified description. The search for realization through constructive diagrams is an effort to understand the required form so fully that there is no longer a rift between its functional specification and the shape it takes."

A worked example at the end of *Notes* aggregates all of the requirements of an Indian village into a hierarchy of naturally disjoint subsystems (concepts, requirement aggregates), each of which is illustrated with a constructive diagram. In, "A City is Not a Tree," Alexander argued that the 'hierarchy' or tree structure suggested in Notes should be replaced with a 'semi-lattice." Other than noting this fact, I leave it to the reader to follow up and see why this matters.

In the preface to the paperback edition of *Notes*, Alexander said the most important idea in the book was the diagrams. *"These diagrams, which, in my more recent work, I have been calling patterns … the key to the process of creating form."* Taking him at his word, it was inevitable - *The Timeless Way of Building* (*TW*) and *A Pattern Language* (*APL*) soon joined my library.

But before going there, a caveat is in order. Actually two: first, Alexander is prone to promising more than he delivers; and two, he is Janus faced. The latter is more important here because the first might be nothing more than an example of Mencius' aphorism, *"It is the teachers responsibility to hold up one side of the square, the student's to find the other three."*

The stated intent of *Notes* was to find a science, a mathematics, of design - to reduce the design process to formula. He was sufficiently successful in this effort that his work became a major part of the discussion, in 1968, in Belgium, to spur the invention of the discipline of Software Engineering.

Everything Alexander wrote about the selfconscious and unselfconscious processes as well as the diagrams, contradicted his stated intent. Alexander was clearly a man in conflict with himself. Later I learned more biographical information about the man and that knowledge confirmed his Janus nature. He fervently desired to be a scientist, he was trained as a mathematician, but could not escape his commitment to the mystical, the ineffable, and to his faith convictions.

I did not actually read *TW* and *APL* until I left graduate school and began my academic career at the University of St. Thomas. The Graduate Programs in Software that I joined became the largest graduate software development program in the world – over 900 FTE students at its peak.

When I joined the program, prospective students had to have had two years of real world experience in software development to gain admission. This meant that the students often new more about how things were really done and what was really important in terms of both theory and practice than the faculty. This made for a very invigorating and challenging environment for a newly minted professor.

St. Thomas is just down the street – less than a mile, directly west on Summit Avenue, from Macalester College where I was an undergraduate. When I hied myself off to Macalester College, I intended to become a quantum chromodynamicist.

Physics 101 was so dull and a happenstance encounter with Asian Philosophy via a freshman seminar was so exciting I changed majors my first semester. My specific interests centered on mysticism and enlightenment – along with intense exploration into altered states of consciousness.

The interest in altered states led to an experiment where I took LSD, still legal at the time, while immersed in a sensory deprivation tank. Based on my experience, I found the first half of the movie *Altered States*, starring William Hurt, to be very realistic.

As an academic, I was just as conflicted as Alexander – a mystical anthropologist wolf in software engineer sheep's clothing. In addition the object curriculum I created and the Object Lab that I founded

were semi-heretical, being firmly grounded in behaviorism and simulation based on natural decomposition a' la Alexander and Plato.

The stage is set to continue my conversation with Alexander via *TW* and *APL* but there is one more element in *Notes* to deal with first. The graduate program at St. Thomas was software engineering in all but name. Because of this, I was obliged to teach courses in software development process and method, both structured and later object oriented.

As software engineering grew in popularity and gained space in computer science curricula a lot was written on the topic of method. The early nineties witnessed an efflorescence of object development methods. Which method to use, which to teach was a central question.

Eventually the chaos, at least in terms of objects, was resolved when the "Three Amigos" consolidated their distinct approaches into what became UML and Unified Method. They were then able to use market forces to effectively impose a single, albeit wrong, method for object design, depiction, and programming.

Alexander had some very interesting things say about method. From the 1971 "Preface to the Paperback Edition" of Notes:

> *"… so many readers have focused on the method which leads to the creation of the diagrams, not on the diagrams themselves, and have even made a cult of following this method.*
>
> *Indeed … a whole academic field has grown up around this idea of 'design methods' – and I have been hailed as one of the leading exponents of these so-called design methods. I am very sorry that this has happened, and want to state, publically, that I reject the whole idea of design methods as a subject of study … In fact, people who study design methods without also practicing design are almost always frustrated designers who have no sap in them, who have lost, or never had, the urge to shape things. Such a person will never be able to say anything sensible about how to shape things either."*

Strong words! And, by analogy:

Design method :: design
Software method :: software

all those words written about software and object method are pointless.

Method is a field of study, methodology. As such it is subject to selfconscious-ness and Theory, just as any other domain.

If I don my anthropologist hat and look at the community of methodologists, I immediately recognize that they are indeed engaged in a selfconscious process. Further, their Theory and the worldview it is grounded upon (based on the methodologists' behaviors and language) clearly assumes that humans are not to be trusted, are unreliable, and are venal.

Methodology is premised on a lack of trust in human beings, a preference for formality and the supposed lack of ambiguity that comes from it. There is a lust for a world as definite and predictable as a machine. Art is denigrated and hierarchical control is extolled.

I detest method. To me it is nothing more than an attempt to remove the human and the creative from a developmental course of action. Method is a tool for coercing human beings into executing

precisely described actions in an equally precisely described sequence in order to generate standardized (and also precisely defined) results.

If the world around us were deterministic, mechanical, devoid of ambiguity, and absent any exceptions to invariant formulaic rules – method might be extraordinarily powerful.

Fortunately, the world is fundamentally unpredictable, deeply ambiguous, constantly changing, and intensely interesting.

We live in this world, interact with, and alter it. Simply existing alters the World in many different ways. As software developers we are deliberately, albeit not always consciously, altering the world about us – in ways that profoundly affect everyone else living in that world. Software folk are quite literally, Reality Constructors. It might be nice if we had some direction, some counsel, some advice, on how to alter reality in a positive, human enhancing, and respectful way.

This leads us back to *TW* and *APL*, Alexander's two volume manifesto asserting that there is a *"Timeless Way"* and following it will lead to human, humane, and results that exhibit QWAN. *TW* is not a method – it is the antithesis of method.

Alexander seldom uses the term, 'method'. When he does, it is shorthand for *"the discipline that teaches us the true relationship between ourselves and our surroundings."* Discipline is a way of thinking about something. This way of thinking, this perspective, guides our actions but it is merely a means to an end and must be transcended.

> *"Once this discipline has done its work, and pricked the bubbles of illusion which we cling to now, we will be ready to give up the discipline, and act as nature does.*
>
> *This is the timeless way: learning the discipline – and shedding it."*

Of course it would be difficult to develop a methodical way of achieving something as mystical as:

> *"A building or town will be alive to the extent that it is governed by the timeless way. …To seek the timeless way we must first know the quality without a name. … To reach the quality without a name we must then build a living pattern language as a gate. … Once we have built the gate, we can pass through it to the practice of the timeless way. … And yet the timeless way is not complete, and will not fully generate the quality without a name, until we leave the gate behind."*

But, imagine my pleasure encountering *TW* and finding it filled with this kind of language – language that could have been taken directly from the Taoist/Zen Bull Herding pictures.

Although *TW* was about buildings and environments and cities, Alexander stated that the timeless way also applied to life in general – to the way in which we might 'design' our lives by making conscious decisions about the patterns of activity we choose. The key point of TW is that the humans, the patterns of activity of those humans, and the spaces that support those activity patterns, and the patterns that comprise the components from which those spaces are constructed - all are "living" things.

TW promises an approach for building worlds that exhibit QWAN. If this promise is kept then the connection to using software instead of wood and stone seems obvious. Indeed, Alexander, speaking to software developers at OOPSLA made this exact connection. Architects are responsible for a very small portion of the "built environment," but software developers affect every aspect of the constructed, both physical and virtual, world we inhabit.

However, neither *TW* nor APL has anything to say about the composition of 'dead' things. Computers and programming languages (virtual machines) are necessarily 'dead' things. It would seem an inescapable conclusion, therefore, that software patterns are oxymoronic. A disturbing thought, but before addressing the negative, a bit more must be said about the promise of *TW* and *APL*.

The opening sentences of *APL*:

"Volume I, The Timeless Way of Building and Volume 2, A Pattern Language, are two halves of a single work. This book [APL] provides a language, for building and planning; the other book provides the theory and instructions for the use of the language."

And the connection to Notes, from the preface to the paperback edition:

"Today, almost ten years after I wrote this book, one idea stands out clearly for me as the most important in the book: ***the idea of the diagrams****.*

These diagrams, which, in my more recent work, I have been calling ***patterns****, are the key to the process of creating form.*

The idea of a diagram, or pattern, is very simple. It is an abstract pattern of physical relationships which resolve a small system of interacting and conflicting forces, and is independent of all other forces, and of all other possible diagrams. The idea that it is possible to create such abstract relationships one at a time, and to create designs, which are whole by fusing these relationships – this amazingly simple idea, is, for me, the most important discovery of the book.

Because the diagrams are independent of one another, you can study them and improve them one at a time, so that their evolution can be gradual and cumulative. More important still, because they are abstract and independent, you can use them to create not just one design, but an infinite variety of designs, all of them free combinations of the same set of patterns."

A Pattern, a' la *Notes*, is a set of requirements, with a set name; is an independent subsystem; is visualized with a constructive diagram which, in turn, is the integrated composite of a form diagram and a requirement diagram. A force, ala *Notes*, is a requirement.

Patterns are independent in proportion to the degree to which the requirements (forces) are cohesive and coupling with other Patterns is minimal. The points of minimal coupling among patterns reflects "natural disjunctions" of the sort revealed by the unselfconscious process, but seldom, if ever, by the selfconscious process.

Independence arises from the domain, the problem space, the problem and this makes those patterns, supposedly, composable and makes it possible to have a pattern language to express various compositions.

Patterns evolve.

Patterns are descriptive, not proscriptive. Although a pattern might contain very explicit instructions about the implementation of that pattern— e.g. pattern 179, Alcoves — there are no 'instructions' on how to use the pattern or the pattern language to some end.

Designs that are compositions of Patterns, at least if they are appropriate to the domain/problem space and sufficiently evolved, are more likely to be Timeless – to exhibit QWAN – than designs that are not.

All of the above speaks, at least to me, of ways to think about the decomposition of complex and large problem spaces. Given that I was deeply immersed in Objects, Behavior-driven object design, and Smalltalk, everything I learned from Alexander became infused in everything I knew, thought I knew, or came to know about objects.

Some obvious connections:

- A problem space is a domain and is a system.
- A problem space (system) can be decomposed into simpler sub-systems.
- The criteria used to define subsystems are critical. In this regard, behavior is productive while data structures or functions are not.
- A subsystem is a "diagram," later known as a pattern, is an object – hence the idea that everything is an object.
- An object is a system that can be decomposed into subsystems that are also objects.
- Subsystems are autonomous, lightly coupled, and deeply cohesive.
- Subsystems (patterns) are differentiated on the basis of which forces / requirements they "resolve:" i.e. what resolution services they provide the whole system. Objects are differentiated on the basis of their behaviors – the services they provide the system as a whole.
- Forces = requirements = expectations / responsibilities.
- Requirements are resolved (in whole or in part) by a service behavior.
- Subsystems / diagrams / patterns / objects; are composable.

Composition, however, presented a bit of a problem – at least as far as *Notes* was concerned. Despite the fact that Alexander believed that it was, "… *possible to create such abstract relationships one at a time, and to create designs, which are whole by fusing these relationships …"* he failed to demonstrate, or even discuss, composition.

Notes provides a very powerful mechanism for decomposition and for **description** – but offers nothing in the way of **prescription**; using the diagrams / patterns to compose a larger whole. *TW* is equally silent on this subject, but *APL*, the companion volume is supposed to where practical and usage information will be found.

OOP also lacked any treatment of composition – how to put objects together in various ways in order for the collective, interacting with each other, could accomplish a desired goal.

In the case of objects, this lack arises from the almost total lack of discussions about OO design. It was all about programming – with the result that OO programs, even in Smalltalk, were essentially indistinguishable from procedural programs in C, Java, even COBOL.

Tons of material was presented about how to define and design an object. Class libraries grew in size. At one point it was stated that ninety-percent of all existing business applications could be built with the classes in the Smalltalk/V library on the distribution CD.

But nowhere to be found was any discussion of design. To be absolutely fair, Rebecca Wirfs-Brock wrote two books with a lot of discussion of design; and, ten years too late, I too attempted to write about design in *Object Thinking*.

The rich and deep resource of a Smalltalk class library was essentially unused, simply because there was little written about using classes, messages, and objects; only words about how to create them and program with them.

More problematic was the absence of any insight in how to compose an application (program or programs composed of objects) that "Fit" its context - that resolved the forces inherent in that context, that has any hope of exhibiting QWAN.

TW and *APL* will solve this problem?

O' Glorious Promise!

TW sets the tone and outlines the objective: to compose and evolve "living" environments that simultaneously reflect and support the life of the inhabitants. It is easy to make the analogy between 'built environments' like towns and buildings and rooms to applications, and programs and artifacts (computers executing software) and even objects as the smallest exemplar of a subsystem which is a pattern.

Each page of *TW* is filled with exactly the right words, powerful concepts, and deep challenges.

"QWAN" and all that we can say about it (alive, whole, comfortable, free, exact, egoless, eternal) and why each of these contradicts itself and all of them mislead and fail to capture or express. QWAN remains ineffable but experienced.

"Patterns of Events" – the precisely human activities that must be supported by the spaces (software) that enable and encourage them to occur. Events that can themselves be consciously chosen in ways that enhance what it is to be human and for a human individual to be alive. Chosen in ways that enable communities of individuals to form and thrive.

"Patterns which are alive" – the synthesis, the synergy, that arises from living patterns of space and the humans inhabiting those spaces as they pursue those patterns of events that ensure their individual and collective livingness.

"The creative power of language" – patterns provide the vocabulary and the perceived ability of natural language to support composition and creation by juxtaposing vocabulary to form a multiplicity of sentences.

All of this to form and articulate a theory of how to ensure that the environments, and the artifacts we necessarily construct, are in harmony with ourselves and our innate natures as human beings living in a natural world.

A vision that begins with a clear and comprehensive worldview of what it means to be human. A vision that includes how we can choose and design actions and activities that allow us to form communities that enhance our being. A vision that includes how we can design and use space and structure to shape the physical world around us to support the realization of an ideal.

Ideas first found in *Notes* are developed in *TW*. Together both books certainly say the right thing and articulate, to me, a compelling vision. It is up to *APL* to bring this vision to fruition and Alexander promises it will.

A confession: my only interaction with APL at this point was totally superficial. I had not studied that volume nor had it played any role in my thinking about objects. The brief bits I had read provoked boredom and a notion that it was only relevant for architects.

It took a chance event, the crashing of a workshop at an OOPLSA in Washington D.C. in the early nineties; a workshop on patterns in software, particularly object-oriented software to make me pay more attention to patterns and pattern languages.

I was not an official member of that workshop and attended only part of the time, so I was not involved in the community that eventually formed Hillside and founded the PLoP conferences. But I admired and respected many of the people that were in that group and thought, "if they find value in *APL* them perhaps I have missed something important." So, I began to investigate.

APL is supposed to show us how to realize the philosophy of *TW* and the precursor roots found in *Notes*.

O' Bitter Disappointment!

The promise was a pattern language, with individual patterns being analogous to the units of language, and having the generative power of language – i.e. freely composable into an *"infinite variety of combinations."*

Instead we get a rigid network with a single unidirectional relationship (larger-to-smaller / container-contained) among patterns. The only "sentences" you can construct with this language or of the form: "embellish this with these" or "complete that with this."

Alexander modified the ideas in APL, particularly this linear relationship, with the notion of a 'semi-lattice' as noted previously. Nikos Salingaros has extended the matrix idea even further, and provides a much better interpretation of Alexander's words than does Alexander himself.

Dancing In The Street is enhanced by a **Mosaic of Subcultures** each of which is a **Community of 7000** with **Garden Walls** providing **Subculture Boundaries**. This would seem to be a 'grammatical' sentence in the pattern language, but it is in fact nonsensical because it violates the explicitly stated relationships of container-contained or embellished-by. Before suggesting that this is an unfair criticism, consider:

> *"This order, which is presented as a straight linear sequence, is essential to the way the language works."*

In *Notes* and *TW*, patterns were defined as separate subsystems and each pattern is supposed to identify and resolve all relevant forces with a single, albeit malleable, solution. This changed in *APL*:

> *"In short, no pattern is an isolated entity. Each pattern can exist in the world, only to the extent that it is supported by other patterns: the larger patterns in which it is embedded, the patterns of the same size that surround it, and the smaller patterns that are embedded within it."*

The non-isolable constraint on a pattern reduces still further, effectively eliminates, the possibility of any kind of generative language of patterns. It also diminishes to near zero the ability to comprehend individual patterns, to adjust and optimize them, or customize them to varying contexts.

Moreover, patterns as defined and presented in *APL* completely contradict the stated purpose of the diagrams and subsystems (proto-patterns) as discussed in *Notes*. (emphasis is mine)

> *" … so organized that* ***adjustment can take place in each one of these subsystems independently****. This is the reason for its success.*

It seems that the patterns in *APL* actually serve as an exemplar of the self conscious process that Alexander was so critical of in *Notes*.

> *In the selfconscious situation, on the other hand, the designer … [must] decide which subsets of requirements to deal with independently.*

My contention is this, These concepts will not help the designer in finding a well-adapted solution unless they happen to correspond to the system's subsystems. But since the concepts are on the whole the result of arbitrary historical accidents, there is no reason to expect that they will in fact correspond to these subsystems.

The patterns in APL, with a few possible exceptions, seem to exemplify the arbitrary clustering of a few forces, and, as a result, provide solutions that are totally idiosyncratic to that arbitrary clustering. They are not composable, they are not adjustable, they are not complete.

Consider one pattern (one that has no stars, hence is speculative) as an example:

85 - Shopfront Schools

"Around the age of 6 or 7, children develop a great need to learn by doing, to make their mark on a community outside the home. If the setting is right, these needs lead children directly to basic skills and habits of learning.

Instead of building large public schools for children 7 to 12, set up tiny independent schools, one school at a time. Keep the school small, to that its overheads are low and a student teacher ration or 1:10 can be maintained. Locate it in the public part of the community, with a shopfront and three to four rooms."

The body of the problem consists of an, apparently universal, assertion that children need to learn by doing. The second assertion is of a causal relationship between "the setting" and the more or less automatic discovery by children of basic skills and good learning habits.

The assertion about learning by doing (and the age constraint) are absolutely arbitrary and we have nothing to guide us in determining their validity or even the relative value of this assertion among the myriad assertions that have been made about education and learning.

The discussion adds a few more "facts:"

- Students can learn abstract concepts like ∏ via experimentation.
- Low student teacher ratios improve individual learning.
- A three-fold reduction in student-teacher ratios can be achieved with 'mini-schools.'

If we take all of the assertions and facts as somehow representing the forces that need to be resolved (the requirements to use the language of *Notes*) we would expect the solution to reveal how those forces were resolved.

Instead, we get a barely overlapping age group (with the problem statement), the assertion that schools need to be set up one at a time, a pretty arbitrary ration of 1:10, an assertion that the school be in a public part of the community with a storefront and 3-4 rooms.

And, of course, we must embed or surround our **Storefront Schools** in/by **Children's Homes** and a **Network of Learning**. And within each storefront school we must embed or co-locate: **Pedestrian Street, Self-Governing Workshops and Offices, Accessible Green, Building Complex** and **Opening to the Street**.

Nowhere in the pattern, or the complex of associated patterns, do we find a comprehensive, let alone complete, enumeration of the requirements / forces and the relationships among them that are in play when considering the best way to education 6-7 (7-12) year olds. Without this it is impossible to adapt the pattern – the enable learning by doing in the area of poetry or science rather than a trade like Cobbler.

We are left to speculate as to whether or how the 'solution' in Storefront Schools relates to the solution in University as a Marketplace. It seems that there are some connections. Both are embedded within Network of Learning, and both share a peer/embedded Building Complex. This hints that there are commonalities, perhaps a deeper pattern(s) but we can only speculate based on what is here.

All in all, my reaction to APL was deep disappointment. Not only did it seem like Alexander failed to live up to the promises of Notes and TW, it actually looked like a betrayal – an antithesis. Even though APL is supposed to be a 'Gate' that is passed through and then forgotten, it seems to me that it is a gate to the wrong garden.

Years later the "Gang of Four" (authors of the *Design Patterns* book) faced a mock trial for "crimes against Alexander" and found guilty. Their guilt, if any, was no less than what should be laid at the feet of Alexander himself.

None of the criticism above means that *APL* is without value. As a collection of stories illustrating aspects of a grand narrative of how to think about building and the relationship of the built with the builders and the inhabitants – it has immense value. Despite failing to be what it claims to be, it is an essential elaboration of the *TW* story.

It appears to me that the 'mystical' Alexander used APL (the work of a committee, not a solo work, BTW) to collect various 'stories' and quasi-rituals that would provide an indirect route for others to follow in order to replicate his insights as expressed in *TW*. He faced the same dilemma as any other mystic – knowing something that is ineffable and yet compelled to say something – even knowing it is wrong.

I noted earlier that Alexander is a Janus – a mystic contrasting and competing with a mathematician. It was the mathematician that made all the promises about *APL*, including that it was a pattern language, that it fails to keep.

My ambivalent (closer to hostile) attitude towards *APL* kept me on the sidelines of the patterns community. I have written and published patterns, shepherded others, and attended many of the pattern conferences around the globe. I just couldn't muster the same enthusiasm or level of commitment as those in the core of the movement.

This "outsider" perspective, coupled with my training as a cultural anthropologist, made me an observer of the patterns community rather than a member. As an observer, it was interesting how quickly the community became "institutionalized" and began exemplifying "the self conscious process." I am thinking of the heated debates about format, of the need for fidelity to the literal Alexander, and the necessity for a pattern to have a geometric essence.

It was tempting, at this point, to simply leave Alexander behind with the conclusion that the grand dream I found in *Notes* and *TW* was not realizable – at least not through such a concrete and prescriptive device as *APL*. But it seemed that Alexander himself shared some measure of my own disappointment and disillusionment. He used that reaction as a stimulus to continue the quest to a deeper level – to an attempt to discover and reveal, *The Nature of Order* (*NO*). So, I concluded I needed to forgive *APL* and see what *NO* might offer.

I approached *NO* with very few expectations, simply curiosity. *Notes* and *TW* had significant influence over my thoughts about software development, often confirming existing biases, other times stimulating new insights. With *APL* none of that was true. And *APL* more than *Notes* and *TW* provided the lens, through which, I first read *NO*.

One of the first things I saw in *NO* was a glimmer of something claimed in *APL* – Poetry.

"In a poem the meaning is far more dense. Each word carries several meanings; and the sentence as a whole carries an enormous density of interlocking meanings, which together illuminate the whole.

The same is true of pattern languages. … it is also possible to put patterns together in such a way that many many patterns overlap in the same physical space: the building is very dense; it has many meanings captured in a small space; and through this density it becomes profound."

I did not see poetry realized in *APL*. In fact reading those few pages about poetry made we wonder what Alexander was smoking – as I saw nothing in *APL* to support his claims.

In *NO*, however, when you can speak of a Boundary that is a Center, simultaneously differentiating and separating other centers while creating Deep Interlock and Ambiguity among both of them and yourself – you begin to see what Alexander called poetic density.

The rigid, Newtonian, structural prose that seemed to be the fate of *APL* is replaced with a quantum (think superposition) dynamic non-deterministic generative language.

But the real question I was asking myself when I approached NO, was, "can *NO* fulfill the promise of *TW*?" My tentative response is, … almost.

TW was unabashedly mystical, *NO* is more grounded in biology than the ineffable. The mystery is still there, but it is the mystery of the not yet known. The emphasis on an inescapable whole is still there along with the corollary that nothing can be understood outside of its context.

The only aspect of *NO* that I find totally problematic is its deism, which very much feels like a cop out.

My interactions with *NO*, and related attempts to connect it to threads originating in *Notes* are ongoing. Therefore the conversation with Alexander continues and will continue for some time. Moreover, perhaps most important, is that the attempt to integrate *NO,* actually all of Alexander's writing, with other areas of study proceeds' full bore.

As I write this essay, I am semi-retired (will take work if you have some to offer). For almost twenty-five years now, I have enjoyed the friendship of Richard Gabriel and benefited from his knowledge of, and insights into, the work of Alexander. (He was also kind enough to take on the shepherding task for this essay.)

I have been very fortunate the past seven or eight years to converse and collaborate with Jenny Quillien – author of *Delight's Muse*, which she describes as a kind of "*Cliff's Notes* for the *Nature of Order*. (She offers, far more than a mere synopsis; sharing insights gained through working with Alexander as he completed his last major work.) Together we have completed various patterns papers and have presented essays on Alexander and *NO*.

Most recently, a serendipitous encounter with a book, *Thriving Systems*, upon leaving an elevator in a London bookstore led to an opportunity to work with the author, Leslie Wageuspack. Rebecca Rikner is a patterns enthusiast and a co-author of patterns papers who, sometimes, acts as a counter agent when my cynicism with patterns waxes.

This means my conversation with Alexander is no longer a dialog, but a multi-faceted conversation of immense richness. And this conversation is taking me in numerous, all related, and all connected in some way to Alexander, directions.

Beginning with notions of design.

NO deals with living systems – what we currently call complex adaptive systems (CAS). Systems of this type are qualitatively different from the complicated, but deterministic and quantifiable systems of engineering, science, and software engineering.

All of our success in computing and software is because we have been dealing with very simple systems. As von Neumann pointed out:

> "Future computers will require but a dozen instruction types, a number known to be adequate for expressing all of mathematics. "This number should not be surprising since 1,000 words are known to be adequate for most situations in real life, and mathematics is only a small part of life, and a very simple part at that. If people do not believe that mathematics is simple, it is only because they do not realize how complicated life is."

CAS are hard! They also tend to be big, ultra-large scale. They present problems that are different in kind. Resolution of those problems requires a different approach to design, a different way of thinking about design.

I have used the term 'design' frequently throughout this essay and Alexander uses it just as frequently in all of his works. In the preceding paragraphs I made a distinction between two qualitatively different kinds of design – as advocated by Alexander in *NO*, versus, as used by computer scientists, engineers, and software development professionals.

Alexander does not actually define what he means by design. He uses the term and he speaks a lot about problem solving. I did not immediately discover a 'smoking gun' quote, but I have a deeply felt conviction that Alexander recognized that the problems arising from complex adaptive, living, systems requires a different understanding of what design is and how it might be practiced.

One thread of my ongoing conversation with Alexander is incorporated in work, with Rebecca Rikner in our book with the working title of *Design Matters.*

We begin with a simple definition of 'design' – *"the intentional and informed decision to alter a system by adding, deleting, or modifying an element of that system or a relationship between/among element(s)."*

Two things are key in this definition. First, is that design is intentional and, more importantly, is informed. Informed has two components, knowledge and a theory that allows application of that knowledge. All of Alexander's works entwine both aspects of informed with *TW* and *APL* being the most explicit separation of the two.

(Thinking of patterns as 'nuggets of wisdom' instead of reusable 'building blocks' actually enhances the value of patterns – in my opinion of course. I will return to this in just a bit, in the context of work I am doing with Jenny Quillien.)

Traditional notions of design assign too much to the designer – as creator, bringer of something from nothing. Our understanding of design is consistent with the origins of the term and with Alexander in NO. A designer is 'midwife', not 'mother' and her role is facilitating "unfolding." Unfolding is a sequence of "essence preserving transformations" that allows the flower to emerge from the seed.

Alexander uses biology, specifically flowers, to illustrate his notion of unfolding. A seed contains the innate essence of the flower. And that innate essence can be conceived in terms of the fifteen properties. The seed undergoes a series of "essence preserving transformations." At each step of the series the

superficial shape – the apparent mix of and relationships among the fifteen properties can seem to have very different Shapes – but the Form is preserved.

Although there are hints of unfolding and of the proper relationships among builder, building, and dweller as supporting the emergence of Form, it does not become explicit in Alexander's work until *NO*.

I have seen one attempt to tease out the precursor ideas and concepts that led to unfolding and apply it to design of an artifact. Not a software artifact, but a cathedral. Richard Gabriel, in his essay, "The Designed and Designer," touches upon ways that a building expresses itself, with the architect as midwife, such that its innate QWAN is manifest.

In *Design Matters*, we hope to limn an approach to design that is consistent with Alexander's ideas of unfolding, while still being generally applicable to the intentional design of any complex system.

Picking up the pattern as nugget of wisdom idea – Jenny Quillien and I are writing a book, tentatively called, *Story*. It deals with the ubiquitous presence of story in software and business and is grounded on the notion that the only reliable form of knowledge communication among humans is the simple story. It is also the oldest, since humans started telling stories as soon as they developed language.

Patterns come into the picture as a story genre. Most stories can be categorized into a genre in terms of both subject (romance, mystery, historical) and structure (boy meets girl, boy loses girl, girl finds boy, lover ever after). A pattern structure is more explicit than most fiction, but not necessarily non-fiction, especially for peer reviewed journals. From the beginning there have been variations in the pattern format, attempts, we believe, to find a better way to tell stories.

APL is a collection of stories as follows.

> *Today I was working on the really complex dynamic system. I wanted to change it, to make it better, but it was really hard. Looking at the system, I perceived some natural seams and focused on a subsystem defined by those seams. I gave it name and studied it, taking voluminous notes. When I was done, when I felt I understood the essence of the system and what kinds of things people could to with it - changing its Shape to better suit a need, but not its Form – I published the result with a title that I hoped would capture the essense of what I had come to know about this small domain, so that others could find it easily and make a quick decision if my work would facilitate their own thinking about the problem.*
>
> My story is nothing more than my story, how I came to comprehend a complex thing and work with it to suit my needs. You might learn from my story, but no more than the stories of everyone else in the same situation. But you cannot use my story, you must write your own.

Thriving Systems is an attempt to apply the fifteen properties in NO to the construction of software. As mentioned earlier, I discovered the book in London, found it fascinating, shared it with Jenny, who contacted Leslie, the author, and arranged for a week of collaboration among the three of us at Jennie's home in Santa Fe.

This was a wonderful experience – hearing Alexander speak in three different voices – as each of us spoke of our idiosyncratic understanding of Alexander's writings.

Leslie does a very credible job of mapping the NO properties to software. He also, perhaps more importantly, constructs various matrices and diagrams showing how the properties interact with, and reinforce, each other.

Our three-way conversation has yet to generate any results, but it has reinforced some of the ideas I have been developing. For example, earlier in the essay I noted that Alexander was focused on 'living systems' and as a result very little of his work had any applicability to 'dead' systems like computers and software. This conviction is most definitely not shared by Leslie, and Jenny is somewhat undecided, so I must emphasize that this is my thinking not ours.

My interpretation of Alexander has also been influential in the rewriting of my book *Object Thinking*. The new book, working title is *Object Thinking: Rethought*, will focus entirely on decomposition and design of complex systems.

All three of these ongoing conversations, with both Alexander and colleagues, will shape my most ambitious effort, redefining software development. (Or replacing it with another discipline.) *Ars Magna* (after the book published by Ramon Lull who is considered the father of computational science), will be the book that captures this effort.

In many ways, *Ars Magna*, will attempt to parallel Alexander's decades long attempt to redefine the practice of architecture. My book will attempt to define the Timeless Way of software supported systems development. Some key ideas with Alexandrian influence:

- Transcendent Art – not Science. Art being a disciplined way of thinking about and practicing something. Disciplined meaning using a body of knowledge that has been transcended (*a' la*, Passing Through the Gate)
- Living systems not dead ones. Instead of creating artificial replicants (androids for humans) of living systems, use computers and software to enhance bits and pieces of existing systems (artificial heart for a living human).
- Understanding and decomposing systems in terms of 'natural joints' – incorporating all that has previously been said in this essay about objects.
- Design arising from and understanding of essence (Form) and facilitating expression of that form in appropriate or desirable Shape.
- Composition – how to put things together – the one thing that I believe Alexander ultimately failed to achieve.

It is a discussion of that last point, composition, and the possibility of a 'pattern language' that supports guided assembly, that will conclude this essay.

Alexander's works offer important insights on how to do, in the words of Plato, "*perceiving and bringing together under an Idea the scattered particulars, [and the] separation of the Idea into classes, by dividing it where the natural joints are."*

It would seem to be a short step offer techniques for putting those 'Ideas' together in useful ways. Alexander promises exactly this with "pattern languages."

Personally, I see not potential with patterns and pattern languages, but do see how the fifteen properties might provide a tool for composition. But there are two caveats: it is unlikely to be a distinct, new language, *per se*; and second, the only 'speakers' of that language might be those who have "Passed through the Gate."

Let me illustrate.

If I have any particular skill, it is an ability to look at the most complicated problem and immediately see the appropriate object solution. The objects and their nature appear, seeming spontaneously, and they are 'correct'. The object relation diagrams and object models that I quickly sketch are simple, the

objects to the simplest things, what they do is consistent with their innate nature, and the only changes to the 'sketch' are those mandated by the choice of the programming language used to implement them.

This skill developed over time, but it was based on an 'object language' – both a vocabulary and a set of concepts – not a grammar or syntax – that allowed the vocabulary to be used appropriately. Years ago, when I was just learning the 'language' it required a lot of effort to get it right. It was only after I passed through the Object Gate, that any kind of 'fluency' in the language appeared.

I seem to be experiencing faint glimmerings of a similar ability with regard to the fifteen properties. For some totally unknown reason, as I was writing this part of the essay, the following story popped into my head.

> *'In the dark of the night, the Soviet Army and their construction minions from East Germany descended on a political boundary and erected a stone and steel wall. Windows in the sides of buildings that happened to lie on that boundary were bricked over. Only a few checkpoints, allowing passage through the wall were created. On the East side of the wall a series of gradually more difficult to penetrate barriers were established. A dismayed populace had to learn how to use this Wall, even to the extent of whether to approach it. Guards, allowed a learning curve, slowly escalating their response to incursions from verbal warnings to immediate execution.*

The story simply appeared in its entirety, in a near dream. But as it played out in my mind I saw a Boundary immediately transform itself into Center (without ceasing to be a Boundary). I saw other Centers forming, the Deep Interlock and Ambiguity of the 'no mans land', and Gradients, both in space (open land, barbed wire, the Wall) and in time (the response of the guards).

Strange story, to suddenly appear decades after the Wall was built and destroyed, but the point is the totality of what was appearing in my mind – the story and its simultaneous articulation in terms of the fifteen properties.

Subsequent to the initial experience, I have thought of the Wall scenario more broadly, in term of the political, social, and economic systems in which the Wall was introduced, still seeing Centers and unfolding. More interestingly, I could sense how the Shape of what was done, the manner in which change was introduced into the larger scale system, doomed it to long term failure.

Frustratingly, most of these 'insights' are beyond my means to articulate – I see them, and believe them, but cannot say them. I am certain, again without being able to say why, that communication with others who share the same insights would be as simple as pointing and saying, "see, there."

All of the preceding constitutes grounds for continuing the conversation with Alexander that began twenty-six years ago. The conversation will contain some arguments. For example I think that the property Alternating Repetition is nonsense – but if you changed it to Rhythm, particularly Jazz Rhythm, then Alexander would be on to something.

In any case, the dialog with Alexander has been valuable, and now that it is a conversation that includes colleagues that know and understand more than I, it will be of immense value in the future as well.