Christopher Alexander: An Introduction for Object-Oriented Designers

Doug Lea*
SUNY Oswego / NY CASE Center

Software developers lament *"If only software engineering could be more like* X *..."*, where *X* is any design-intensive profession with a longer and apparently more successful history than software. It is therefore both comforting and troubling to discover that the same fundamental philosophical, methodological, and pragmatic concerns arise in all of these *X*s (see, for example, [23, 33, 43, 46, 18, 45, 48, 50]). In part because it is considered as much artistry as engineering, writings about architecture have most extensively explored and argued out the basic underpinnings of design. Even within this context, the ideas of the architect Christopher Alexander stand out as penetrating, and bear compelling implications for software design.

Alexander is increasingly well-known in object-oriented (OO) design circles for his influential work on "patterns". This paper considers patterns within a broader review of Alexander's prolific writings on design. These include core books *Notes on the Synthesis of Form*[1], *The Timeless Way of Building*[5], and *A Pattern Language*[4] (hereafter abbreviated as *Notes*, *Timeless*, and *Patterns* respectively), other books based mostly on case studies[15, 3, 6, 7, 8], related articles (especially [2, 9]), and a collaborative biography[29].

This review introduces some highlights of Alexander's work. The format is mainly topical, roughly in historical order, interspersed and concluded with remarks about connections to software design. It focuses on conceptual issues, but omits topics (e.g., geometry and color) that seem less central to software. Some discussions are abstracted and abbreviated to the point of caricature, and in no case capture the poetry of Alexander's writings that can only be appreciated by reading the originals, or the concreteness and practicality of pattern-based development that can only be conveyed through experience.

*This is a work in progress. I encourage comments and reactions; mail to `dl@g.oswego.edu` or Doug Lea, Computer Science, SUNY Oswego, Oswego, NY 13126 USA. Copies (`ca.ps`) may be ftp'ed from `g.oswego.edu`. Thanks to Richard Helm, Ralph Johnson, and Chamond Liu for help with drafts.

Quality

Alexander's central premise, driving over thirty years of thoughts, actions, and writings, is that there is something fundamentally wrong with twentieth century architectural design methods and practices. In *Notes*, Alexander illustrates failures in the sensitivity of contemporary methods to the actual requirements and conditions surrounding their development. He argues that contemporary methods fail to generate products that satisfy the true requirements placed upon them by individuals and society, and fail to meet the real demands of real users, and ultimately fail in the basic requirement that design and engineering improve the human condition. Problems include:

- Inability to balance individual, group, societal, and ecological needs.
- Lack of purpose, order, and human scale.
- Aesthetic and functional failure in adapting to local physical and social environments.
- Development of materials and standardized components that are ill suited for use in any specific application.
- Creation of artifacts that people do not like.

Timeless continues this theme, opening with phenomenologically toned essays on "the quality without a name", the possession of which is the ultimate goal of any design product. It is impossible to briefly summarize this. Alexander presents a number of partial synonyms: *freedom*, *life*, *wholeness*, *comfortability*, and *harmony*. But no single term or example fully conveys meaning or captures the force of Alexander's writings on the reader, especially surrounding the human impact of design, the feelings and aesthetics of designers and users, the need for commitment by developers to obtain and preserve wholeness, and its basis in the objective equilibrium of form. Alexander has been working for the past twelve years on a follow-up book, *The Nature of Order*, devoted solely to this topic (see [29, 9]).

Method and Structure

Notes is Alexander's most conventional and still most frequently cited book, and most clearly reflects Alexander's formalist training. (He pursued architecture after obtaining science and mathematics degrees. He is also an artist, Turkish carpet collector, and licensed contractor.) It has much in common with other works on systems, design, and engineering that appeared in the late 1950s and early 1960s attempting to integrate ideas from cybernetics, discrete math, and computing, exuding an optimistic tone that real progress was being made.

Notes (see also [15, 12, 40]) describes how, before the advent of modern architectural methods, artifacts tended not to suffer from adaptation, quality, and usability failures. The "unselfconsciously" constructed artifacts of tradition are produced without the benefit of formal models and methods. Instead, a system of implicit and often inflexible rules for design/construction progress in an evolutionary fashion. Over time, natural forces cause successive artifacts to better adapt to and mesh with their environments, almost always ultimately finding points of equilibrium and beauty, while also resulting in increasingly better rules applied by people who do not necessarily know why the rules work.

Historically, the modern "rational" design paradigm was both a contributing factor towards and a byproduct of the professionalization of design (see, e.g., [37, 18]). Rational design is distinguished from traditional craftmanship by its "selfconscious" separation of designs from products (or, to continue the evolutionary analogy, genotype from phenotype), its use of analytic models, and its focus on methods that anyone with sufficient formal training may apply. Analytic designers first make tractable models (from simple blueprints on up) that are analyzed and manipulated into a form that specifies construction.

Rational design was in many ways a major advance over traditional methods. However, as discussed in *Notes*, the notions of analysis and synthesis are badly, and harmfully, construed in architecture and artifact design, leading to the sterile study of methods that have no bearing on the vast majority of artifacts actually built or the work involved in developing them. (Wolfe[51] provides a breezier account of some of this territory, but focusing on the schools and cults of personality found in modern architecture, that luckily have few parallels in software engineering.)

The main problem lies in separating activities surrounding analysis and synthesis rather than recognizing their duality. While it is common to exploit the symmetries between form and function (roughly translatable as system statics versus dynamics), further opportunities for integrating views become lost. Like an organism, a building is more than a realization of a design or even of a development process. Model, process, context, and artifact are all intertwined aspects of the same system. Artificial separations of models, phases, and roles break these connections. One consequence is that abstract representations lose details that always end up mattering, but each time in different ways. The micro-adaptations of tradition are lost, and resist model validation efforts in those rare cases in which they are performed. Alexander provides examples from houses to kettles in which fascination with the form of detached, oversimplified, inappropriate models leads to designs that no user would want.

In *Notes*, Alexander argues that the key to methodological continuity, integration, and unification is to temper, or even replace intensionally defined models with reliance upon complete, extensionally-described sets of constraints, specific to each design effort. To match its context, a solution must be constructed along the intrinsic fractures of the problem space. This ecological perspective generates design products that are optimally adapted to the microstructure of local conditions and constraints, without the "requirements stress" characteristic of the products of classic methods.

Notes includes presentation of a semiformal algorithmic method that helps automate good partitioning under various assumptions. To use it, one first prepares an exhaustive list of functional and structural constraints. The major illustrations employ 33 and 141 constraints respectively, each collected and refined over periods of months. The algorithm takes as input a boolean matrix indicating whether any given pair of constraints interact – either positively or negatively, although concentrating on the negative since "misfits" are easier to identify and characterize. The method results in indications of groupings that minimize total requirements interaction and resulting complexity. This statistical clustering algorithm arrives at subsystems by minimizing the interaction of problem requirements that each one deals with. The goal is to mirror the micro-structure that each part in a well-adapted unselfconsciously designed system would possess. This method relies upon a consideration of all such constraints, again leading him to argue for empirically and experientially guided analysis.

Even though exemplified with architectural artifacts, Alexander's concerns and methods apply equally well to software systems, subsystems, objects, etc. While there are many obvious differences between houses and software, most are matters of degree at this level of discussion. Consider, for example:

- Software entities engage in greater dynamic interaction (e.g., send messages to each other).
- Sometimes, describing software is the same as constructing it (as in programming).
- More of a software design is hidden from its users.
- Software generally has many fewer physical constraints.
- Some software requirements are allegedly more explicit and precise than "build a house here".

None of these have much bearing on methodological issues. As noted by Dasgupta[18], Alexander's early writings on structure and method have influenced designers in all realms, including computer scientists ranging from Herbert Simon to Harlan Mills. Variants of Alexander's decomposition algorithm have been applied to OO software [13]. One can find passages in standard presentations of OO decomposition (e.g.,[14]) that surely have indirect roots in this work. Although apparently independently conceived, Winograd & Flores[50] is especially close in spirit, and includes discussions that Alexander might have written had he been dealing with software:

> Many of the problems that are popularly attributed to "computerization" are the result of forcing our interactions into the narrow mold provided by a limited formalized domain.
>
> The most successful designs are not those that try to fully model the domain in which they operate, but those that are "in alignment" with the fundamental structure of that domain, and that allow for modification and evolution to generate new structural coupling.

These themes form a basis for most of Alexander's later writings. However, later efforts are also in large part a response to failures in the methods and algorithms presented in *Notes*, as discovered by Alexander and others [2, 29, 33, 38, 49]. While they remain useful guides and tools, the methods encounter problems including the possibility of missing relevant constraints, assumptions that requirements are completely knowable beforehand, ignoring the intrinsic value-ladenness of requirements specifications, inability to deal with relative weights among constraints or higher-level interactions, failure to accommodate the fact that design components may interact in ways that requirements do not, and inflexibility in adapting to future constraints. These problems, along with observations that people blindly following such methods do not always create better products, led to work increasingly removed from mainstream architectural design practices.

Patterns

Timeless and *Patterns* were written as a pair, with the former presenting rationale and method, and the latter concrete details. They present a fresh alternative to the use of standardized models and components, and accentuate the philosophical, technical and social-impact differences between analytic methods and the adaptive, open, and reflective (all in several senses) approach to design that Alexander is reaching for.

The term *pattern* is a preformal construct (Alexander does not ever provide a formal definition) describing sets of forces in the world and relations among them. In *Timeless*, Alexander describes common, sometimes even universal patterns of space, of events, of human existence, ranging across all levels of granularity.

Patterns contains 253 pattern *entries*. Each entry might be seen as an in-the-small handbook on a common, concrete architectural domain. Each entry links a set of forces, a configuration or family of artifacts, and a process for constructing a particular realization. Entries intertwine these "problem space", "solution space", and "construction space" issues in a simple, down-to-earth fashion, so that each may evolve concurrently when patterns are used in development.

Entries have five parts:

Name. A short familiar, descriptive name or phrase, usually more indicative of the solution than of the problem or context. Examples include *Alcoves*, *Main entrance*, *Public outdoor room*, *Parallel roads*, *Density rings*, *Office connections*, *Sequence of sitting spaces*, and *Interior windows*.

Example. One or more pictures, diagrams, and/or descriptions that illustrate prototypical application.

Context. Delineation of situations under which the pattern applies. Often includes background, discussions of why this pattern exists, and evidence for generality.

Problem. A description of the relevant forces and constraints, and how they interact. In many cases, entries focus almost entirely on problem constraints that a reader has probably never thought about. Design and construction issues sometimes themselves form parts of the constraints.

Solution. Static relationships and dynamic rules (microprocess) describing how to construct artifacts in accord with the pattern, often listing several variants and/or ways to adjust to circumstances. Solutions reference and relate other higher- and lower-level patterns.

But not everything of this form counts as a pattern. Ideally, pattern entries have the following properties:

Encapsulation. Each pattern encapsulates a well-defined problem/solution (cf.,[41, 42]). Patterns are independent, specific, and precisely formulated enough to make clear when they apply and whether they capture real problems and issues, and to ensure that each step of synthesis results in the construction of a complete, recognizable entity, where each part makes sense as an in-the-small whole.

Generativity. Each entry contains a local, self-standing process prescription describing how to construct realizations. Pattern entries are written to be usable by all development participants, not merely trained designers. Many patterns are unashamedly "recipes", mirroring the "unselfconscious" procedures characteristic of traditional methodless construction. An expert may still use a pattern in the same way that an expert chef uses a cooking recipe – to help create a personal vision of a particular realization, while still maintaining critical ingredients and proportions.

Equilibrium. Each pattern identifies a solution space containing an invariant that minimizes conflict among forces and constraints. When a pattern is used in an application, equilibrium provides a reason for each design step, traceable to situational constraints. The rationale that the solution meets this equilibrium may be a formal, theoretical derivation, an abstraction from empirical data, observations of the pattern in naturally occurring or traditional artifacts, a convincing series of examples, analysis of poor or failed solutions, or any mixture of these. Equilibrium is the structural side of optimality notions familiar in computing, and can be just as hard to find a basis for, meet, or approximate [28]. Alexander argues for establishment of *objective* equilibria based in the "quality without a name" even (or especially) when surrounding aesthetic, personal, and social factors. He also notes the elusiveness of this goal – artifacts more often than not fail to achieve this quality despite the best of efforts.

Abstraction. Patterns represent abstractions of empirical experience and everyday knowledge. They are general within the stated context, although not necessarily universal. (Each entry in *Patterns* is marked with a "universality" designation of zero to two stars.) Pattern construction (like domain analysis[44]) is an iterative social process collecting, sharing, and amplifying distributed experience and knowledge. Also, patterns with a structural basis in or similarity with natural and traditionally constructed artifacts exploit well adapted partitionings of the world. Sometimes, patterns may be constructed more mechanically, by merging others and/or transforming them to apply to a different domain. And some patterns are so tied to universals that they emerge from introspection and intuition uncontaminated by formalism. Heuristics based on participatory design, introspection, linkage to existing artifacts, and social consensus all increase the likelihood of identifying central fixed and variable features, and play a role even when that environment is purely internal and/or artificial, but where each part helps generate a context for others.

Openness. Patterns may be extended down to arbitrarily fine levels of detail. Like fractals, patterns have no top or bottom – at the lowest levels of any design effort, some are merely opaque and/or fluid (e.g., plaster, concrete). Patterns are used in development by finding a collection of entries addressing the desired features of the project at hand, where each of these may in turn require other subpatterns. Experimentation with possible variants and examination of the relationships among patterns that together form the whole add constraints, adjustments and situation-specific specializations and refinements. For example, while only a small set of patterns would typically apply in the design of a certain housing community, each house will itself be unique due to varying micro-patterns. Because the details of pattern instantiations are encapsulated, they may vary within stated constraints. These details often do impact and further constrain those of other related patterns. But again, this variability remains within the borders of higher-level constraints.

Composibility. Patterns are hierarchically related. Coarse grained patterns are layered on top of, relate, and constrain fine grained ones. These relations include, but are not restricted to various whole-part relations[16]. Most patterns are both upwardly and downwardly composible, minimizing interaction with other patterns, making clear when two related patterns must share a third, and admitting maximal variation in sub-patterns. Pattern entries are arranged conceptually as a *language* that expresses this layering. Because the forms of patterns and their relations to others are only loosely constrained and written entirely in natural language, the pattern language is merely analogous to a formal production system language, but has about the same properties, including infinite nondeterministic generativity.

Process

Patterns includes brief recipe-like accounts on how to apply and compose patterns. However, Alexander discourages slavish conformance, and describes development mainly through concrete examples illustrating how groupings at different levels of hierarchies tend to be based upon different levels of concerns. Coarser-grained patterns are less constraining in detail than finer-grained ones. Exact commitments are postponed until the consequences of lower-level construction and/or experimentation can be assessed.

Even though high-level patterns hold throughout development, this process need not, for example, generate a classic blueprint drawing before construction. Also, because the relations among larger and smaller patterns do not always represent strict containment, there may be interactions among subpatterns and other higher-level interactions requiring experimentation and resolution. *Patterns* includes entries (e.g., *Site repair*) describing how to deal with particular kinds of interactions. All "joints", "transitions", and "spaces" among components are explicitly designed using other patterns that balance the needs of the parts versus the needs of the whole.

Pattern-based design activities resist accommodation within a linear development process, and raise challenges in the construction of suitable process models that still meet costing, predictability, and control criteria. Since the early 1970s Alexander has experimented with several overall development processes that preserve the integrity and promises of pattern-based design, as applied to projects at all scales, including houses, a cafe, a medical facility, apartments, two universities, a rural housing community, and an urban community [5, 29, 3, 9, 6, 7, 8]. The resulting process principles and development *patterns* include:

Collective Development. Development is a social process. Participation from all levels (users, policy-makers, etc.) is required for decisions affecting multiple parts or users, as well as those concerning future growth and evolution. Rather than a plan, a group adopts a (stateful) process that balances collective and individual needs, and preserves the rationale for particular decisions.

Participatory Design. Users can help design things that they really need and want, that are better adapted to their surroundings, and that are more aesthetically pleasing (see also [47, 40, 34]). Even if the design participants are not the permanent, ultimate users, participation by someone impacted by the artifact is better than the alternative. Architects may reject user requests only when their knowledge of local constraints is demonstrably greater.

Responsibility. Architects hold financial and legal charge for the consequences of their activities, and control corresponding cash flow. This provides both authority and responsibility for adaptation across development.

Decentralization. Larger efforts can be subdivided into expanding centers or domains, that increasingly influence one another in the course of growth. Localized experimentation, discovery, and change are intrinsic to such adaptation. This includes situations in which conditions change and designs evolve. The diagnosis and local repair of problems with existing parts are part of any design effort.

Integration of Roles. Designers operate at several levels. Primary roles should be assigned with respect to problem task or domain, not phase or level. Architects must sometimes be builders, and vice versa. They cannot otherwise get things right. Intimacy with all aspects of an effort allows the builder-architect to firsthand discover constraints, needs and desires.

Integration of Activities. Design is interwoven with synthesis in a mainly bottom-up fashion. Construction proceeds in an order governed by pattern interdependencies, the continuous analysis and repair of failures, and commitment to detail, variety, experimentation, and wholeness. Concurrent development of mostly-independent parts allows construction to branch out from multiple centers, ultimately "stiffening" into final form.

Stepwise Construction. Artifacts are constructed one pattern at a time, each of which results in a complete, recognizable form adapted to other already-constructed artifacts and partially committed plans. Efforts are focused upon operations, not components. Each operation is complete in itself. Creativity and accomplishment are maintained at all levels of this process.

Patterns and OO Design

The form and features of patterns, and the methods and processes surrounding them are in no way special to architectural design. The entries in *Patterns* represent "special theories" of the world. Alexander notes[29] that his characterization of patterns meshes well with common definitions of scientific theories. The heuristics governing the construction of patterns are all but indistinguishable from those for theories. (See also [18, 49, 38], who note that while such correspondences add an aura of respectability, they also open up design to the controversies surrounding modern scientific method.) Patterns are less general than descriptions of the base semantics of the pattern language itself, yet equally far removed from the realm of "neat tricks". The careful interplay between contexts, problem-space forces, and constructive solutions make this framework an ideal basis for capturing other kinds of design knowledge and practices as well.

In fact, Alexander's patterns bear a straightforward relation to OO constructs. Patterns may be viewed as extending the definitional features of classes. In OO design, classes have two principle aspects, analogous to those of patterns:

- The external, problem-space view: Descriptions of properties, responsibilities, capabilities and supported services as seen by software clients or the outside world.
- The internal, solution-space view: Static and dynamic descriptions, constraints, and contracts among other components, delegates, collaborators and helpers, each of which is known only with respect to a possibly incomplete external view (i.e., a class, but where the actual member may conform to a stronger subclass).

The best classes also share the properties of appropriate abstraction, encapsulation, openness, and equilibrium. Like patterns, classes are normally generative, supporting parameterized instance construction as well as higher-order instantiation in the case of generic (template) classes. Classes are intrinsically composible, although these compositions need not always be expressed as classes, e.g., at topmost decomposition levels.

Indeed, since patterns can describe concepts and structures (e.g., coordinated groups) that are not themselves objects, the term *pattern* may be more fitting than *class* (or alternatively, the notion of a class should be broadened) at least at the level of OO design variously termed "abstract", "architectural", and/or "functional" (see., e.g., [20]). Patterns can thus raise the expressiveness and level of description supported by familiar OO constructs. Conversely, OO concepts may be applied to strengthen pattern-based design notions:

Languages and Tools. Alexander grammatically arranges pattern entries (although in an implicit fashion) to exploit the generative properties of formal languages [29]. In computing, just about every possible formal, semiformal, and informal set of constructs have been collected as a language of some sort. For example, as shown in the Demeter project [36], a set of OO classes may be represented grammatically using rewrite rules denoting pattern-like compositional layering. However, it is unnecessary to construe a collection of patterns or classes themselves *as* a language. In programming, it is usually more convenient to express descriptions *in* a broader language, to facilitate manipulation, compilation, etc. Extensions of OO modelling, design and/or programming languages may serve well in representing patterns. Such formalization also allows for construction of design tools. Several Computer Aided Architectural Design (CAAD) systems have represented Alexander's patterns in software. Most recently, Galle [24, 25] has described a CAAD framework supporting pattern-based design built as a partially object-oriented expert system. Aspects of this system might be abstracted as patterns, and used in the construction of similar CASE design tools. However, it will surely take some time before OO design tools and books reach the utility and authoritativeness of *Patterns*.

Subclassing and Refinement. In addition to supporting compositional relations, all OO notations include a second kind of structuring rule to describe possible alternative paths though a set of concepts, capturing both the composition / decomposition and abstraction / refinement design spectra within a linguistic framework. OO methods and languages thus add a new set of concepts to this aspect of Alexander's framework. While the notion of variability within broad classifications permeates his writings, Alexander does not explicitly employ the idea of structured refinement through subclassing. This probably stems from the fact that there is no good reason for formalizing the concept in architectural design, where there is little use in explicitly capturing the refinements between a pattern and its realization. Instead, the pattern is (often gradually) *replaced* by its realization. However, in software, these intermediate forms can play all sorts of roles in development, including use as branch points for alternative specializations, bases for differential design, descriptions of common protocols in OO frameworks, and a means for swapping in one component for another.

Inheritance and Delegation. OO design techniques incorporating various subclassing, delegation, and composition constructs conquer a potential obstacle found in the application of pattern-based design in other realms. Alexander's patterns provide a basis for *design* reuse without any necessary implications for *component* reuse, thus limiting the routine generation and predictable use of standardized components with known cost and properties, and running into quality-control problems intrinsic to reliance on one-shot implementations. This is generally not the case in OO design. Even when an existing or standard component isn't what you want, it often happens that alternative specializations, delegation structures, and/or subclasses can share much code via standard OO programming tactics. In fact, this happens so often that OO programmers are surprised, complain, and are sometimes unable to cope when it does not (e.g., fairly often in concurrent OO programming[39]).

Adaptation and Reflection. Further out, OO concepts may also help crystalize the senses of methodological unity, adaptation, openness, and reflection that pervade Alexander's work. The lack of a crisp distinction between software "design" and "manufacturing" already makes development practices harder to classify along the continuum from craftmanship to analytic engineering[34]. This becomes accentuated when software systems themselves include provisions for self-adaptation and redesign. So while it sounds overly metaphysical to, for example, view buildings as clever devices to propagate architects or blueprints (cf., [19, 21, 49]), in software these dualities have very practical consequences. Work in OO and AI (e.g., [35, 31, 50, 30]) has led to reification and metalevel reasoning constructs that, although by no means completely understood, allow creation of useful systems in which the borderlines between designer, model, design, and product nearly vanish, as is necessary for example in computer assisted manufacturing (CAD/CAM/CIM)[11], where the market-driven trend has been to move away from systems that merely increase productivity or reduce defects in mass-produced products. Instead, systems must rely on both adaptive development methods and adaptive software mechanisms to enable the reconfigurability required to obtain flexibility and user-perceived quality in manufacturing small runs.

Process Integration. While OO process models remain underdeveloped, their potential synergy with pattern-based models is obvious. The average OO developer personifies the builder-architect (hacker-designer?) ethic at the heart of pattern-based development processes. More than anything else, *experiences* with OO versions of patterns have been the driving force leading OO researcher-practitioners to examine and exploit the many relationships between the semantic bases, usages, activities, and processes of OO and pattern-based development. Most work is still in the exploratory phase; including reconceptualizations of basic OO techniques and idioms (e.g., those found in [17, 14, 20]), OO frameworks([32]) and micro-architectures ([10, 26, 27]), as well as the methods, processes, tools, formalizations, development patterns, education, and social contexts best supporting their development. It may yet turn out that the ideas that have long isolated Alexander from the mainstream commercial architectural community[9, 22] will find their widest and most enduring impact in object-oriented software engineering.

References

[1] Alexander, C., *Notes on the Synthesis of Form*, Harvard University Press, 1964.

[2] Alexander, C., "A Refutation of Design Methodology" (Interview with Max Jacobson), *Architectural Design*, December, 1971.

[3] Alexander, C., M. Silverstein, S. Angel, S. Ishikawa, & D. Abrams, *The Oregon Experiment*, Oxford University Press, 1975.

[4] Alexander, C., S. Ishikawa, & M. Silverstein, *A Pattern Language*, Oxford University Press, 1977.

[5] Alexander, C., *The Timeless Way of Building*, Oxford University Press, 1979.

[6] Alexander, C., *The Linz Cafe*, Oxford University Press, 1981.

[7] Alexander, C., *The Production of Houses*, Oxford University Press, 1985.

[8] Alexander, C., *A New Theory of Urban Design*, Oxford University Press, 1987.

[9] Alexander, C., "Perspectives: Manifesto 1991", *Progressive Architecture*, July 1991.

[10] Anderson, B., & P. Coad (Organizers), "Patterns Workshop", *OOPSLA '93*.

[11] Ayers, R., & D. Butcher, "The Flexible Factory Revisited", *American Scientist*, September-October 1993.

[12] Basalla, G., *The Evolution of Technology*, Cambridge University Press, 1988.

[13] Bonine, J., "A Theory of Software Architecture Design", unpublished draft manuscript, 1993.

[14] Booch, G., *Object Oriented Design with Applications*, 2nd ed., Benjamin Cummings, 1993.

[15] Chermayeff, S., & C. Alexander, *Community and Privacy: Toward a New Architecture of Humanism*, Doubleday, 1963.

[16] Civello, F., "Roles for Composite Objects in Object Oriented Analysis and Design", *Proceedings, OOPSLA'93*, ACM, 1993.

[17] Coplien, J., *Advanced C++: Programming Styles and Idioms*, Addison-Wesley, 1991.

[18] Dasgupta, S., *Design Theory and Computer Science*, Cambridge University Press, 1991.

[19] Dawkins, R., *The Selfish Gene*, Oxford University Press, 1976.

[20] de Champeaux, D., D. Lea, & P. Faure, *Object Oriented System Development*, Addison-Wesley, 1993.

[21] Dennett, D., *The Intentional Stance*, Bradford Books, 1987.

[22] Dovey, K., "The Pattern Language and its Enemies". *Design Studies*, vol 11, p3-9, 1990.

[23] French, M. J., *Invention and Evolution: Design in Nature and Engineering*. Cambridge, 1988.

[24] Galle, P., "Alexander Patterns for Design Computing: Atoms of Conceptual Structure?" *Environment and Planning B: Planning and Design*, vol 18, p327-346, 1991.

[25] Galle, P., "Computer Support of Architectural Sketch Design: A Matter of Simplicity?" *Environment and Planning B: Planning and Design*, vol 21, 1994.

[26] Gamma, E., R. Helm, R. Johnson, & J. Vlissides, "Design Patterns: Abstraction and Reuse of Object-Oriented Designs", *Proceedings, ECOOP '93*, Springer-Verlag, 1993.

[27] Gamma, E., R. Helm, R. Johnson, & J. Vlissides, *Design Patterns*, Addison-Wesley, forthcoming.

[28] Garey, M. & D. Johnson, *Computers and Intractability*, Freeman, 1979.

[29] Grabow, S., *Christopher Alexander: The Search for a New Paradigm*, Oriel Press, 1983.

[30] Hamilton, G., M. Powell, & J. Mitchell. *Subcontract: A Flexible Base for Distributed Programming*. Sun Microsystems Laboratories Technical Report TR-93-13, 1993.

[31] Hewitt, C., P. Bishop, & R. Steiger, "A Universal Modular ACTOR Formalism for AI", *Third International Joint Conference on Artificial Intelligence*, Stanford University, August 1973.

[32] Johnson, R., "Documenting Frameworks Using Patterns", *Proceedings, OOPSLA 92*, ACM, 1992.

[33] Jones, J. C., *Design Methods*, 2nd ed., Van Nostrand, 1992.

[34] Karat, J. (ed), *Taking Software Design Seriously: Practical Techniques for Human-Computer Interaction Design*, Academic Press, 1991.

[35] Kiczales, G., J. des Rivieres, & D.G. Bobrow, *The Art of the Metaobject Protocol*, MIT Press, 1991.

[36] Lieberherr, K. & I. Holland, "Assuring Good Style for Object-Oriented Programs", *IEEE Software*, September 1989.

[37] Lucie-Smith, B., *A History of Industrial Design*, Van Nostrand, 1983.

[38] March, L. (ed), *The Architecture of Form*, Cambridge University Press, 1976.

[39] Matsuoka, S., K. Taura, & A. Yonezawa, "Highly Efficient and Encapsulated Reuse of Synchronization Code in Concurrent Object-Oriented Languages", *Proceedings, OOPSLA '93*, ACM, 1993.

[40] Norman, D., *The Psychology of Everyday Things*, Basic Books, 1988.

[41] Parnas, D., "On the Criteria to be Used in the Decomposition of Systems into Modules", *Communications of the ACM*, December, 1972.

[42] Parnas, D., "Designing Software for Ease of Extension and Contraction" *IEEE Transactions on Software Engineering*, March 1979.

[43] Petroski, H., *To Engineer is Human*. St. Martin's Press, 1982.

[44] Prieto-Diaz, R., & G. Arango (eds.), *Domain Analysis: Acquisition of Reusable Information for Software Construction*, IEEE Computer Society Press, 1989.

[45] Rowe, P., *Design Thinking*. MIT Press, 1987.

[46] Schön, D., *Educating the Reflective Practitioner*, Jossey-Bass, 1987.

[47] Schuler, D., & A. Namioka, *Participatory Design*, Lawrence Erlbaum, 1993.

[48] Simon, H., *The Sciences of the Artificial*, MIT Press, 1981.

[49] Steadman, P., *The Evolution of Designs*, Cambridge University Press, 1979.

[50] Winograd, T., & F. Flores, *Understanding Computers and Cognition: A New Foundation for Design*, Addison-Wesley, 1986.

[51] Wolfe, T., *From Our House to Bauhaus*, Pocket Books, 1981.