2001

Designing Philosophers

Paul R. Taylor
Monash University, ptaylor@csse.monash.edu.au

Designing Philosophers

Paul R. Taylor

Department of Computer Science and Software Engineering,
Monash University, Melbourne, Australia.
ptaylor@csse.monash.edu.au

Abstract

From the classical conceptualisations and discourses of Aristotle and Plato to the disturbing situatedness of the existentialists, philosophy provides perspectives on aspects of existence, thought and knowledge that illuminate and explain as well as connecting the somewhat specialised and inaccessible theory of software and systems engineering to the wider consciousness. This paper summarises the positions of selected philosophers that have been interpreted by some influential authors in software engineering to strengthen their arguments and in turn influence software methodology. Consideration of these philosophical perspectives can usefully influence the design of research projects and the analysis of results, particularly where the enquiry involves the business, industrial and other situated contexts in which contemporary system design and development is done.

Keywords

Design philosophy, design models, design process, design methods

INTRODUCTION

If pressed, most software engineers can recall the point in their early schooling where they made a decision to stream into mathematics and the sciences and away from literature, arts and the creative pursuits. During tertiary education and subsequent professionalisation, the typical software engineer only brushes with philosophy through a thorny concurrent programming problem or a self-motivated interest on the side. This disconnection is illustrated by the fact that only a handful of significant and even seminal books engage philosophy in a practical and involving way. In this unfortunate case of denial, a generation of software engineers has lost a chance to relate, in the reflections of everyday design and development, the science and craft of software to the timeless foundation of knowledge. It would seem to be at least informative then, to review the ways that philosophy has been called upon to directly motivate or support directional changes in the recent history of software development.

A theme around which to base this brief survey that touches on issues of knowledge, perception of ourselves and the world, and particularly action in all its forms is needed to orient the investigation. Design is a good candidate because it is so central to software development, and it relates dimensions of knowledge, thought and perception. It is also very widely represented in the software engineering and information systems literature. The remainder of the paper surveys a small but influential subset of philosophers (chosen because of their citation and incorporation in some influential software engineering and information systems texts) and the ways that their philosophy is interpreted to software and systems engineering.

DESIGN, KNOWLEDGE AND LEARNING

Designing is an activity that resolves ideas and concepts into artefacts. The process of design is one of creating conceptual representations of solutions and then satisfying ourselves that the solution works acceptably. Design builds on a base of knowledge, perception and interpretation, all of which in turn have different meanings depending upon one's preferred philosophical position. Much of philosophy has been an attempt to understand how the mental and physical domains are related, and because design is an activity that involves fluidly working between both domains interchangeably, philosophy has particular relevance to understanding design.

Arisristotelian and Platonic concepts

Aristotle formulated his philosophy on absolute concepts and rules, and argued that all of the world's knowledge consisted of combinations of these conceptual atoms. His philosophy accounted for the formation of knowledge through fresh combinations of existing concepts to form new composites. A consequence of this view of knowledge is that any concept, no matter how complex, must decompose into basic concepts.

Aristotle and Plato devised classification, a foundation of almost all sciences. The history of classification in both biology and chemistry reveal a chain of related shifts in the basis of classification (from observable

characteristics to habitats, breeding habits, evolutionary progressions and now DNA sequences) that suggest the nature of the classification process to be incremental, iterative and grounded in perspective. Because classification bases must take a stance on the problem of ambiguity—what do you do when an object fits two or more categories—most classifiers end up choosing the basis that yields a classification that suit their particular application.

Booch (1994) used Aristotelian categorisation to explain how, in practice, it is often difficult to give convincing categorical definitions to even the most mundane of objects. 'The identification of classes and objects is the hardest part of object-oriented design', Booch (p. 133) writes. Hordes of programmers discovered how hard classification could be (or alternatively, some more experienced classifiers discovered how badly novices could get classification wrong) in the transition to object-oriented programming (Kaindl 1999).

Static inheritance structures reflect the limitations of Aristotle's absolutism outside of minimalist or constrained laboratory settings—classification works fine if the problem domain reduces to a small, stable set of abstractions. Plato preferred to think of concepts as being primarily defined by our practices and actions rather than as absolute things, occupying a place of their own in the world of knowledge. Concepts exist only because we exist, and concepts allow us to share a vocabulary of shared meaning. Plato viewed knowledge acquisition as the process of coming to recognise one or more of these shared concepts. Plato developed a process of dialogue along the lines of a twenty-questions game to support classical categorisation. Is it animal, mineral or vegetable? Does it have fur or feathers? Can it fly? Ideas become concepts when two questioning minds debate, the argument and counter argument serving to bring forth a more complete and complex understanding than previously existed. Established concepts then become exemplars to assist in learning and in defining new exemplars, and the transfer of knowledge reduces to a personal process of acquisition through dialogue.

These two classical views help to understand what it is the designer works with. Using Aristotelian concepts, we formalise our knowledge by providing rules and criteria for determining when concepts apply. Using Platonic concepts, we provide typical examples that may then be compared to specific observed or experienced phenomena to assess similarities and differences. Aristotelian definitions lead to clear distinctions. The question of whether a given phenomenon fits a given concept is decidable—it does, if and only if it has the required properties. No discussion is necessary. Platonic definitions lead to more fuzzy distinctions. It cannot always be unambiguously decided that a given phenomenon is an instance of a specific concept. Attempts at making distinctions will often lead to discussions of similarities and differences between the phenomena and prototypical examples, and the personal background of the observer will influence decisions on how the categorisation is made.

Many concepts in design appear to be Platonic, particularly those that relate to *doing* design. When designers talk of adapting prototypical solutions, such as the way software architects use schemas and design patterns (Beck et al. 1996; Coplien 1996; Seen et al. 2000), Platonic conceptual reasoning is played out. Theorising about design moves the reflective designer away from Platonic and toward Aristotelian conceptualisation. The two forms of thinking about design are interspersed in written works that relate theory to practice, and in the minds of reflective practitioners. Schon (1987), in considering how expert practitioners self-assess, defines reflection-in-action as the act of stopping in the instant of designing to consider why a particular action was taken, and then incorporating the assessment in the current design act, and reflection-on-action as the act of reflecting on how a particular design act was done after its completion. A great deal of design practice is performed without substantial investment in Aristotelian thinking.

In software design, this chasm of concepts cannot be avoided, because programming languages (as typified by their support for classification) are blatantly Aristotelian in character, and it is the mapping from action-oriented imitation to the rigid categories of software structure that accounts for much of the perceived pain of making software systems fit imprecise human skills and modes of work.

The philosophies of both Aristotle and Plato are, as Dahlbom and Mathiassen (1993).put it, 'mechanistic', in that they attempt to define reality and all of the kinds of systems that interact with it as mechanisms that obey universal laws. In this view, all forms of knowledge and perception sit under the umbrella of universal science and all investigation of knowledge and phenomena must be subject to positivism and the scientific method (Weatherall 1979). Some design theorists clearly believe that this is the direction that design theory and research must be pushed, as evidenced by the movement to formalise a discipline of design science (Cross 1993; Warfield 1994). Others argue that there can never be a unifying science of design because of the irreconcilability of viewpoints in design research (Love 2000; Sargent 1994). Still others never subscribed to—or have abandoned—the positivist paradigm altogether (Coyne 1991; Seaman 1999). They look to Descartes and the social relativists that were to follow for a definition of reality as a social construction.

Descartes' Separation of Mind and Body

Descartes shaped scientific method profoundly by separating the mind (soul substance) from the body (physical substance), thereby allowing one to be studied independently of the other (Hirschheim 2001). This allowed subsequent philosophers to distinguish between the way the world is and the way we perceive it. The mind is not just a passive machine for receiving information about the world—rather it shapes these impressions and adds unique interpretations. Compared with mechanistic philosophy, this was a 'romantic' philosophy that distinguished between the world in itself and the world of phenomena experienced by every individual person, so that it became impossible to hold absolutes, to know about something tangible or conceptual, with absolute certainty. To accept Descarte's perspective implies accepting that anything wrong with the world is something that is actually wrong with the perceiver, and as a result, the early romantics tried to find answers to questions about the world in human nature. Later romantic philosophers transferred the origin of reality from the individual to culture, claiming that our perceptions of the world are cultural in origin. Dahlbom and Mathiassen summarise the implications for design:

> *Our ideas, rather than being representations of an external real world measured by their similarity to that world, are constructions measured by their internal coherence* (Dahlbom and Mathiassen 1993, p. 42).

The question of whether theories are representations of worlds or constructions of worlds becomes an important one in systems design. The former implies that the information system is a representation of the work processes in an organisation, whereas the latter implies that systems are an attempt to construct an organisation of such work processes.

The ideas of a 'constructed reality' circulating inside people's heads and of reality as existing in this fashion exclusively, seem irreconcilable to the engineer. We design, we build, people use, we evaluate, and the realness of our artefacts cannot be disputed. At the point where the philosophical relativists step up, it may be useful to think of their contributions as applying to user's perspectives of systems, and not to the systems themselves. This allows the designer to assume the usual objective stance whilst leaving room for others to have subjective views. The perception that there is a need to provide such an escape route tells us a lot about software engineers and the rationalistic software culture. Many information systems researchers have commented on the rationalistic tradition of system design and development, typified by De Marco (1978), Yourdon (1986) and Jackson (1983):

> *These step-by-step, rather linear processes force our thinking into narrow pathways, where emphasis is placed on isolating single problems and searching for the one 'right' solution. While these ideals may be suited to certain types of scientific problem solving, we feel that they are ill-suited to the dynamic and generally chaotic conditions of developing computer systems for the workplace.* (Greenbaum and Kyng 1991, p. 8)

If we can resist the urge to escape the insanity of the subjectivists for a while, the notion of socially (ie. collectively) constructed reality does have some interesting implications for the way engineers think about design and our artefacts in the world.

Nietzsche and Perspectivism

In defining concepts and conceptual schemas as constructive creations of the world, the 'romantics' introduced the idea of multiple, culturally formed realities and perspectives. Rather than looking for the true perspective, the romantics wanted to use different perspectives to open up different and meaningful dimensions of reality. The definition of new perspectives brings new meanings, new uses and deeper understanding. With multiple perspectives in effect, the straightforward question 'what is this object' resolves to 'whatever you conceive it to be'. The individual's propensity to adopt a perspective is motivated by self-preservation—man chooses to 'impose meanings that suit his taste for survival' (Bullock and Woodings 1983).

Nietzsche used perspectivism to argue that preferred metaphors, theoretical positions in science, and established interpretations tell more about the background and interests of their proponents than they do about the world. When a perspective such as 'object-orientation is a superior paradigm for software design' is put forward and vehemently argued for, perspectivism accounts for the proponent's motivation as being interests to defend instead of a concern to understand the truth or otherwise of the claim. Rather than establishing evidence or truth, Nietzsche would seek to ascertain the controllers and the controlled, the power relationships represented by such a question. Perspectives play a role in understanding the relationship between stakeholders and the designer in the design process, and as a consequence how a designed artefact is perceived and subsequently used. The use of sceptical thought framework such as this can be very useful in understanding design and product evaluations.

Sartre and the Design of Existence

After perspectivism replaced absolute truth with as many relativist constructions of truth as there are distinct actors or roles, Jean-Paul Sartre furthered the extension of relativism to existentialism—the thought that the world exists only because we want it to, and that things have no purpose other than that invested by us. Sartre described man as being entrapped in an existence without essence, and so constructs 'a system of projects directed toward the future' to infuse existence with meaning (Bullock and Woodings 1983, p. 677). Rather than humans existing to fulfil a purpose, they have to design their own purpose, always going beyond what they already have. Life may be regarded as an ongoing project of setting and reaching goals, then setting new ones, always inflating expectations and constantly striving to define meaning through becoming.

Existentialism is one of the few philosophical movements to explicitly account for the relationship between man and the designed or built world. 'Men, endowed with will and consciousness, find themselves in an alien world of objects which have neither' (Bullock et al. 1977, p. 297). We design meaning by using a phenomenological approach (bracketing or disregarding pre-existing meanings) to investigate these objects, and then make existential 'leaps'—acts of pure decision—to define our own characters. But no single 'leap' is ever definitive. When we reach our goal to establish a level of design quality that we have previously targeted, or a particular formulation of the design process that we think will bring understanding or enlightenment, we find ourselves reassessing its meaning with questions like 'what is design?' or 'what is quality?' Existentialism accounts for the driving force behind material progress—the absence of absolute models which contain universal meaning, and our inability to rest comfortably with what we have achieved motivates us to design the world around us, as if the act of perpetual designing served to account for our identity.

Heidegger and the Grounding of Phenomenology

If Sartre's work explained the separation of mind and body, human striving and the need for design to justify our existence, Heidegger explained 'being' in terms of our relationship with the objects that surround us. Heidegger's work developed out of phenomenology, the work of his teacher Husserl. Heidegger was principally interested in 'being'. He used the term 'Dasein' to describe the separation of subject and object—the particular situation that humans find themselves arbitrarily 'thrown into'—a condition characterised by having consciousness but being surrounded by inert material objects (Steiner 1979). Bullock's commentary illustrates how Heidegger—a pragmatist who believed in a tangible and situated existence—tied his philosophy to experiences:

> *Faced with the necessity of choice and death, we are all individually presented with a moral and ethical dilemma. Most choose to trivialise their freedom of choice by denying death and living a conventional life, but some live authentic lives by resolutely confronting death and exercising the creative nature of freedom. Man, as a temporal being and the former and pursuer of projects, must conceive the world of things around him in terms of its availability for his active purposes.* (Bullock and Woodings 1983, p. 316)

Far from being an obtuse concept, 'thrownness' turns out to be very ordinary and familiar one, known by every person as the fluid and unpredictable interactions of everyday life. Winograd and Flores (1986) draw heavily on Heidegger's notion of 'thrownness' in their influential work on computers and cognition, which served to reorientate artificial intelligence research away from static Aristotelian concepts and a priori planning toward a more contextual or situated view. (Suchman (1987) did the same for planning at about this time). They summarise Heidegger's philosophy, with implications for software and system design, as follows:

- Our implicit beliefs and assumptions cannot all be made explicit—there is no neutral viewpoint from which everyone can see all things, and a truly objective understanding of most worldly phenomena is difficult and often impossible to achieve. This point reinforces Budgen's (1994) characterisation of software design as being one of Rittel and Weber's (1984) 'wicked' problems.
- Practical understanding is more fundamental than theoretical understanding—the Western philosophical tradition, Winograd and Flores observe, is based on the assumption that a detached theoretical point of view is superior to the involved practical viewpoint. The scientist devising theories is closer to truth than those with an everyday experience of phenomena. Heidegger reverses this, claiming that when we interact with the world non-reflectively we primarily know it. Detached theorising can be informative but it can also obscure the phenomena by categorising and isolating them. This view is consistent with the move towards sociological and qualitative research techniques evident in information systems research. In design research, the diverse and primarily practical nature of the activity is so pervasive that it is held up as one of the primary hindrances to the formation of a science of design (Sargent 1994).
- We prefer to relate to things directly, not via models—Heidegger rejected 'mental representations' in favour of 'concernful acting in the world'. Winograd and Flores handle this point carefully because on the face, it sounds like a decree to dump every conceptual model in computer science. Their interpretation

stresses 'concernful activity' (informed action) over 'detached contemplation' (observation only, with no direct experience of phenomena), thereby berating detached or remote theorising and model-making over applied theorising.

- Meaning is fundamentally social, it emerges from interaction and not purely from individual action—the rationalistic view of cognition, Winograd and Flores propound, is individual-centered. For example, we teach meaning in language to individual learners, whereas linguistic meaning in language can only emerge in the context of social activity.

Winograd and Flores suggest that Heideggerian 'thrownness' should change the way that perceptions, actions and attributes (or properties of things) are regarded in computer systems design. They speculate that the software designer needs to stop being the 'objective detached scientist who makes observations, forms hypotheses, and consciously chooses a rational course of action' (p. 36). Two further concepts illustrate 'being' in a way that the software designer might identify with.

Heidegger claimed that properties were emergent in use, which implies that an object itself outside of use has no properties. Winograd and Flores use an example of a hammer to illustrate. When a craftsman is driving a nail with a hammer, the hammer does not exist in the consciousness of its holder—the craftsman uses it as an extension of his forearm, sensing the transfer of energy from hammer head to nail, and the reaction of the nail and timber in the subtle bounce of the hammer immediately after impact. But if the hammer slips, or the blow glances the nail, the holder becomes (sometimes painfully) aware of the presence of the hammer—its 'hammerness' emerges in momentary failure. Heidegger used the term 'readiness-to-hand' to describe this melding of actor and object, the craftsman and his hammer, and the term 'breaking down' to describe the jolt back to awareness when the object misbehaves momentarily. The hammer's properties are defined by the current context of use. As observers, we may talk about the hammer and reflect on its properties, but for the craftsman engaged in the thrownness of driving home a nail, there is no hammer there at all, and the observer can never know this. Achieving this kind of complete transparency in an artefact, and anticipating and minimising the impacts of its 'breaking down' behaviours, define the ultimate goal for any designer of artefacts for use.

Winograd and Flores develop these and other points to a set of recommendations for designing computer systems that include allowance for language and conversation as the primary information media, the recognition of triggering and breakdown events in the design process, consideration of the user's interpretations of every visible element of the system's design, along with a broad dose of situated ethics.

APPLICATION TO SOFTWARE DESIGN

If nothing else, the act of bringing philosophy to bear upon the human act of design and the theory of software engineering specifically is enlightening. It may not change the way we do our designing tomorrow, but it does provide a descriptive link between the human context of system-use and the mechanistic thinking of construction. The following comments provide additional views on the nature of design that illustrate how our strategic view of practice may be usefully changed by an awareness of philosophical foundations.

Processes of Transcendence

These and other foundations of thought about objects, action and perception provide a philosophical basis for understanding. Although proponents of both the objectivist ('mechanistic' or rationalistic) and subjectivist ('romantic') schools have at times claimed that the other did not exist, a middle ground that attests truth to both sides is commonly agreeable. This affords that the world contains both objective truths (such as the weight, temperature or density of a physical artefact) that exist independently of the presence of a person, and subjective truths (such as the intended modes of use) that depend upon a person's perception of the phenomenon or object.

Philosophy is relevant to design because it provides a basis from which to understand the potential meanings of observation, the source of data from which hypotheses are derived to generate theory. In the process of doing design, these philosophical bases may be useful in shedding light on certain phenomena observed in the process. They may also enable the designer to anticipate and address certain dangers such as the multiple viewpoints of perspectivism or the different ways that concepts are understood.

Dahlbom and Mathiassen cite another benefit. They suggest that all our attempted definitions of important tacit qualities like knowledge are flawed, and that we know this before we start. But we must continue to make inadequate definitions in order to have something to critique, so that we know what questions to ask next time. Beyond direct application, our repeating attempts to derive Aristotelian concepts from Platonic intuitions generate reflection, introspection and dialog. Even when such attempts fail, the researcher or practitioner gains

a grain of information, or the opportunity to synthesise a new fragment of knowledge, or a new perspective that can be used to plan and scale the new attempt. It is this recurring 'process of transcendence' that drives useful learning, meaningful investigation and innovation.

Self-conscious and Unselfconscious Design Transfer

The distinction and application of the expert's or designer's use of tacit and explicit knowledge provides a window on the design culture's philosophical position. Design is a behaviour that is substantially practiced from tacit knowledge, internalised skills and constructive memory. Practiced skills have both explicit and implicit means of transfer. The real distinction between these forms of transfer, in Alexander's view, is discerned by looking at the way in which design, or the production of useful objects, is taught in either case:

> *I shall call a culture unselfconscious if its form-making is learned informally, through imitation and correction. And I shall call a culture self-conscious if its form-making is taught academically, according to explicit rules'* (Alexander 1964, p. 36).

In the unselfconscious context, the teaching of craft skills is performed through demonstration. The novice imitates the skilled craftsman until the 'feel' of the various tools and techniques have been learned. Learning occurs by practicing the actual skill, during which sample artefacts may be produced as a result of the learning process. With the 'self-conscious' design process, techniques are taught by first being formulated explicitly as theory and then being explained theoretically, independently of the physical object:

> *In the unselfconscious culture, says Alexander, the same form is repeated over and over again, and all that the individual craftsman must learn is how to copy the given prototypes. But in the self-conscious culture there are always new cultures arising, for which traditional given solutions are inappropriate or inadequate; therefore it is necessary to bring to bear some degree of theoretical understanding, in order to be able to devise new forms to meet the new needs* (Steadman 1979).

Another useful distinction is made by Walker (1976) as a result of his modelling of the design roles and the designer's social context. The difference between the self-conscious designer and the unselfconscious designer is their output—the self-conscious designer produces design which is fabricated or implemented by another party, a sub-contractor, builder or trades-person, whereas the unselfconscious designer (or crafts-person) fabricates their design themselves, and makes no distinction between design and fabrication. Design transfer, as with design, is intimately inter-twined with such issues as pedagogy, design and cultural transmission.

A close examination of the way that designers are taught and learn—unselfconsciously, self-consciously or both—provides a useful window on the prevalent philosophical perspectives of the culture (Borenstein 1991). When software design educators start using design studios rather then lectures, for example, a shift toward a more subjective philosophical basis will have occurred (Rosenman and Gero 1998; Taylor 2000).

CONCLUSION

The effect of pedagogic streaming on an awareness of philosophy does not matter all that much—what does matter is its usefulness in bringing understanding to difficult questions in the perception, thought and knowledge and learning. The brief overview in Section 2 has attempted a brief presentation of what philosophy might be relevant and how it has been applied. Winograd and Flores, along with Suchman, drew upon Heideggerian views to motivate a substantial change in direction for artificial intelligence research in the mid-eighties. Dahlbom and Mathiassen used Aristotle and Plato, Descartes and others to contrast the mechanistic and romantic schools of systems development in the mid-nineties when a broad awakening to the social context of systems development was gaining momentum. Many other authors have lightly dabbled in philosophy to support their theories.

An awareness of philosophy amongst information systems and software engineering theorists, writers and practitioners can contribute a value framework that helps builds bridges between the disciplines and brings an increased sense of perspective. A deeper awareness of the nature or perception and use can only serve to improve the artefacts we design and the methods we use.

REFERENCES

Alexander, C. (1964). *Notes on the Synthesis of Form*, Harvard University Press, New York.

Beck, K., Coplien, J. O., Crocker, R., Dominick, L., Meszaros, G., Paulisch, F., and Vlissides, J. "Industrial Experience with Design Patterns." *International Conference of Software Engineering*.

Booch, G. (1994). *Object-Oriented Analysis and Design with Applications*, Benjamin Cummings, Redwood City, California.

Borenstein, N. S. (1991). *Programming as if People Mattered: Friendly Programs, Software Engineering and Other Noble Delusions*, Princeton University Press, Princeton, New Jersey.

Budgen, D. (1994). *Software Design*, Addison-Wesley.

Bullock, A., Stallybrass, O., and Trombley, S. (1977). *The Fontana Dictionary of Modern Thought*, Fontana, London.

Bullock, A., and Woodings, R. B. (1983). *The Fontana Dictionary of Modern Thinkers*, Fontana, London.

Coplien, J. O. (1996). "Software Patterns.", Lucent Technologies, Bell Labs Innovations, New York.

Coyne, R. D. (1991). "Is designing mysterious? Challenging the dual knowledge thesis." *Design Studies*, 12(3), 124-131.

Cross, N. (1993). "Science and design methodology." *Research in Engineering Design*, Vol 5, 63-69.

Dahlbom, B., and Mathiassen, L. (1993). *Computers in Context: The Philosophy and Practice of Systems Design*, Blackwell, Cambridge, MA.

DeMarco, T. (1978). *Structured Analysis and System Specification*, Prentice-Hall, Englewood Cliffs, New Jersey.

Greenbaum, J., and Kyng, M. (1991). *Design at Work: Cooperative Design of Computer Systems*, Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Hirschheim, R. (2001). "Information Systems Epistemology: An Historical Perspective.", London School of Economics, London.

Jackson, M. A. (1983). *System Development*, Prentice-Hall, Englewood Cliffs, New Jersey.

Kaindl, H. (1999). "Difficulties in the Transition from OO Analysis to Design." *IEEE Software*, 94-102.

Love, T. (2000). "Philosophy of design: a meta-theoretical structure for design theory." *Design Studies*, 21(3), 293-313.

Rittel, H. J., and Weber, M. M. (1984). "Planning problems are wicked problems." Developments in Design Methodology, N. Cross, ed., Wiley, p. 135-144.

Rosenman, M. A., and Gero, J. S. (1998). "Purpose and function in design: from the socio-cultural to the techno-physical." *Design Studies*, 19(2), 161-186.

Sargent, P. (1994). "Design science or nonscience." *Design Studies*, 15(4), 389-402.

Schon, D. A. (1987). *Educating the Reflective Practitioner: Toward a New Design for Teaching and Learning in the Professions*, Jossey-Bass Publishers, San Francisco.

Seaman, C. B. (1999). "Qualitative Methods in Empirical Studies of Software Engineering." *IEEE Transactions on Software Engineering*, 25(4), 557-572.

Seen, M., Taylor, P., and Dick, M. "Applying a Crystal Ball to Design Pattern Adoption." *TOOLS Europe (33)*, Mont-Saint-Michel, France, 443-454.

Steadman, P. (1979). *The Evolution of Designs: Biological Analogy in Architecture and the Applied Arts*, Cambridge University Press, Cambridge.

Steiner, G. (1979). "Martin Heidegger.",, The Viking Press, New York.

Suchman, L. A. (1987). *Plans and Situated Actions: The problem of human machine communication*, Cambridge University Press, Cambridge.

Taylor, P. "Designerly Thinking: What Software Methodology can learn from Design Theory." *International Conference on Software Methods and Tools (2000)*, Wollongong, 107-118.

Walker, D., and Cross, N. (1976). *Design: The man-made object*, The Open University Press.

Warfield, J. N. (1994). *A Science of Generic Design: Managing Complexity through Systems Design*, Iowa State University Press, Iowa.

Weatherall, M. (1979). *Scientific Method*, The English Universities Press Ltd., London.

Winograd, T., and Flores, F. (1986). *Understanding Computers and Cognition: A New Foundation for Design*, Addison-Wesley, Reading, Massachusetts.

Yourdon, E. (1986). *Managing the Structured Techniques*, Yourdon Press, New York.

ACKNOWLEDGEMENTS

This work on the philosophy of design underpins the author's research on 'situated' software architecture and design—further details are available at http://www.csse.monash.edu.au/~ptaylor/. The project is supervised by Associate Professor Christine Mingins (CSSE, Monash University) and Professor Richard Mitchell (Inferdata).

COPYRIGHT