2001

Patterns as Software Design Canon

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

Patterns as Software Design Canon

Paul R. Taylor

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

Abstract

This paper explores the value of the architecture profession's notion of 'the canon' as a design transfer and knowledge transmission model for software architecture, and the overlap with the design patterns movement. The emerging body of architectural and design patterns has already become a kind of software 'canon'—patterns and pattern languages exemplify design and act as design archetypes, have shared visibility through conventional and online media, and result from an established and intensive canonisation process consisting of shepherding (mentored authoring) and writer's workshops (a review protocol). Research is underway to understand the need for a 'software design canon' and its accompanying discourse, and how well the design pattern form can fulfil this need.

Keywords

Design patterns, design reuse, knowledge management, design transfer, expertise transfer, design process, design methods.

INTRODUCTION

All forms of art, literature and design have their universally acclaimed masterpieces, frontispieces, icons, emblems and brands. In renaissance sculpture we look to Michaelangelo's art for the personification in stone of masterful precision, interpretation and artistry of the period. In literature, any sample of school teachers will recite substantially similar shortlists of canonical Australian writers and poets—Lawson, Patterson, Facey, Richardson—the must-reads for every student. Many fields of design also have their canon. In furniture, the canon includes the puritan-inspired classical pieces of the Shakers, Thomas Moore's reinterpretation of traditional English craftsmanship amidst the crushing landslide of industrialisation, and Marcel Breuer's translation of the nascent modernist style into the office and home (Hauffe 1998; Naylor 1990). In interior design, Charles Rennie MacIntosh's Willow Tea-rooms (Glasgow) and Frank Lloyd Wright's Kauffman office (Victoria & Albert Museum) are equally definitional. The most prominent canon of all, the architectural canon, is too diverse and extensive to cameo in this fashion without raising intellectual ire and equivocal eyebrows.

The notion of a universally acknowledged set of works that epitomise the intellectual history of a profession is appealing and has value on many levels—as history, exemplification, as a source of knowledge, and as a resource to define and display a discipline's progress and achievement over time. Equally appealing to systems and software architects is the notion that physical architecture and the highly evolved culture of the designed artefact may have something to teach us about exemplifying significant and outstanding design—and how the software and systems engineering discipline could create an ongoing design discourse that addresses functional, aesthetic, social, economic and ethical dimensions that are currently not adequately aired.

There are two dominant senses of 'canon' that are both of religious origin—firstly, as consisting of a collection of sanctioned works, and secondly, as being a set of principles derived from dogma, rules, or standards of judgement (Downton 1998). The religiosity sits comfortably with some views in architecture, not to mention software design. The musical meaning of canon involves exact imitation of a theme by otherwise independent voices. Imitation underlies the very purpose of all canons as a collection of works, significant in various ways. Canonical works are regarded as exemplars, and as such, they are frequently copied, mutated and bent to new shapes by other designers who choose to re-use or re-interpret aspects of the knowledge and expression embodied in the building or artefact.

In architecture, the canon is broad and distinctive, and is demarcated by periods and styles. A period of design history such as the Art Deco period is encapsulated by both whole sections of cities (Dunedin) and by individual monumental buildings (Museum of Modern Art, Circular Quay). The themes in the works of canonised periods such as this are subject to repeated reinterpretation and reproduction. A style of architecture such as Postmodernism from which we are arguably still extricating ourselves is iconified by a small number of buildings. Venturi House, a house with an oversized roof, gables and fascias (designed at the height of modernist-inspired flat-roofed concrete block construction) as an attempt to reclaim 'house-ness' (Venturi et al.

1977) is typical of these in both form and intent. It is canonical because it reinstated commonplace and everyday semiotics, and changed the way architects of the day perceived 'house'. Rather than using semiotics, classification or teleology as a single unifying thread, the architectural canon succeeds in knitting together diverse theories, their corresponding movements, styles, periods and materials by ensuring each member has significance to the art and science of architecture. Even more than context, it is the ongoing debate of what 'significant' means in the contemporary period that fuels design discourse in the built world.

Purpose and Organisation

This paper is motivated by the idea that the architectural canon, a concept widely acknowledged by architects and design theorists may provide a useful example of how a software architecture canon could be achieved. The architectural canon proves an exemplary knowledge transfer medium in two distinct and complementary ways. Firstly, as instruction—the techniques of visualisation, canonisation and knowledge transfer have relevance to software architecture. And secondly as metaphor—'architecture is an established and powerful metaphor, and its canon may perform the same metaphorical function for software structuring (software architecture), aspects of the development process (requirements, architecture, detailed design, construction) and development roles (the software architect, the software engineer). In software design, a potential mechanism has recently emerged—another motivation is to see how the goals and practices of the current movement to mine and document software design patterns (as typified by Gamma (1995) and Buschmann (1995)) align with those of the architectural canon.

The paper is organised as follows. THE ARCHITECTURAL CANON explores the mechanisms of the architectural canon and compares these with the software design patterns movement. THE CANONISATION PROCESS compares the architectural canonisation process with current pattern mining and promotion processes. RESEARCHING THE NEED FOR A SOFTWARE CANON describes planned research that will assesses how effectively design knowledge is used by expert designers and the role of a software canon, and the final section draws some conclusions about the directions in which the design patterns community may need to move in order to be more 'canonical' and hence more useful as a design knowledge transfer vehicle.

THE ARCHITECTURAL CANON

The architectural canon serves two epistemological functions—first as a repository of different kinds of architectural knowledge, and secondly as a transmitter, or a vehicle for the transmission of this knowledge (Downton 1998). If the canon is defined as a collection of works plus dialogue (or discourse) that assess and position the works, then the dialogue assumes the role of establishing how membership is determined and which specific works will be included. The discourse owes its existence to dialogue, display, analysis, criticism and negotiation through the architectural press and the profession's shows and journals. (Architectural journals, with all their inherent biases, play a lead role in the discourse). Knowledge of architectural theory and history is referenced, refreshed and created as a result of this dialogue, and is brought to bear on the two kinds of knowledge embodied in the works themselves—the knowledge intentionally included by the architect and the knowledge that is formed through the making of the work.

The architectural canon also expresses contemporary values in a highly tangible way. Debates about the inclusion and exclusion of works in the canon encourage a conceptualisation of values and contributions that would have otherwise remained implicit in the example works, in their architectural forms and employed techniques. Macarthur (1998) comments that canonical buildings do not necessarily 'scream out' their statement on the architectural history of their day, but provide examples of how the current values were chosen to be expressed and deployed in the design norms of their time. Thus clear communication of the social and economic values of the day in a building may count more towards its canonisation than functional distinction or even design originality. This distinction between the designed object and the discourse surrounding it over time is critical to our understanding of the value of the canon—some (Macarthur, for example) would argue that the canon's most important value derives from the discourse that it fuels, and that this discourse selects and moulds the substantial issues that shape the direction of the design discipline and profession. A canon continually forces articulation of the issues of the period and continuously re-presents the rhetorical question, 'how then should we design?'

A Software Equivalent

Has the discipline of software design ever really had an equivalent of the architectural canon? The question draws attention to a number of points of comparison. A critical comparison upon which the notion of software canon hinges is the nature of the fabric. Software is effectively invisible, and the software artefact has a strange kind of tangibility and visibility. As a design medium, it differs fundamentally from traditional media in the

intangibility of its structure. The experience of walking through an Italian Basilica or of taking a guided tour of one of Frank Lloyd Wright's prairie houses stirs a degree of admiration, conceptualisation and emotion in most of us, but the notion of 'visiting' a canonical software architecture or experiencing a software artefact is obscure at best. While these kinds of experiential responses do happen, experiencing software design requires a high degree of familiarity with the implementation language, the software architecture or artefact and the entire software infrastructure. It is even difficult to suggest canonical pieces of software. Coplien, when questioned on 'great' pieces of code, suggested EMACS, the Lisp interpreter, and parts of the UNIX kernel (1999). But even the simple act of naming these three claims to exemplary software design kicks off a Platonic discourse that immediately yields some of the benefits of a canon. Why is UNIX canonical? Because in its day it used a lightweight process architecture and a structural consistency and purity not seen before—one proponent might say, and in doing so, expresses values and interpretations for others to critique, debate and resolve. In computer science and software engineering, this kind of dialogue has not been entirely missing (any journal search will turn up hundreds of critiques of UNIX) but rather is separated in time and space from the experience of the artefact itself. It may be argued that this separation makes for a different kind of canon, but a canon nonetheless.

Clearly, a software design discourse has value, but source-code level critique is too low-level and too dense to be useful. The sheer size and accessibility of some potentially canonical works—the Lisp interpreter or UNIX for example—is another barrier to the development of a widely shared sense of understanding and visibility. The simple act of reading enough UNIX kernel source to gain an appreciation of its structure, source mechanisms and its design on a number of levels of abstraction could take months of effort. The highest level of expertise must be attained before an individual can sensibly discuss such architecture. No shared sense of canon can ever develop with these inaccessible works. The software patterns movement promises to elevate the level of abstraction of commonly perceived software design. Extending the work of architect Christopher Alexander, a movement of software designers mining, documenting, criticising and in effect canonising units of conceptual and actual design has produced a substantial volume of software patterns, or templates that capture software design problems, the contexts that they occur in, the 'forces' that the designer must consider, the solution (a fragment of code or architecture, usually illustrated with pseudocode or full source), and a discussion of how the solution resolves the forces present. As this is not a paper about software design patterns *per se* but about the potential of the patterns movement to become the basis of a software design canon, the reader should look elsewhere for examples of design patterns (Gamma et al. 1995), architecture patterns (Buschmann and Meunier 1995), organisational patterns (Coplien 1995), the hundreds of pattern study groups that operate around the world (Alexander 2000) and the theories and experiments of Christopher Alexander (Grabow 1983; Lea 1998).

Qualities of the Canon

Most canons are a collective abstraction subscribed to by a body of disciples and adherents. In some cases, the ability to perceive the canon defines a distinct level of membership, while the rarer ability to evaluate candidates or contribute pieces constitutes a considerably higher status within the fold. Some canons exist partly to ensure professional, religious or sectarian exclusion, and many canons provide this function to some degree, explicitly or unknowingly. But much more effective mechanisms exist to exert such control. A profession's canon promotes knowledge on many levels, from the promotion of theory and paradigm to specific design techniques.

However, a canon is not typically a pull-apart reference kit or a how-to-do-it manual—a profession or discipline's canon should be definitive without being prescriptive. The canon makes theory and paradigm tangible—it shows the way, sets standards, guides, is open to interpretation and reflection, but never prescribes, enforces or dictates. A profession's canon therefore provides an anchor or reference point for knowledge in a universally accessible but highly interpretable form. Table 1 suggests a preliminary comparison of some of these characteristics of design canons with some suggested equivalents from the domain of software design.

Canonical Design Characteristic	***Realisation in Architecture***	***Realisation in Software Design***
History	Built forms, buildings of the period.	Typically not represented.
Reinterpretation	Design, social and political movements.	Paradigm shifts.
Expression of values	Styles and stylistic movements; design semiotics.	Paradigms.
Calibration	Canonical exemplification.	Code-sharing, components.
Embodiment of judgement	Visible design of buildings and artefacts.	Hidden software architectures and system structures.
Embodiment of ethics	Planning and regulation.	*No obvious representation.*

Canonical Design Characteristic	***Realisation in Architecture***	***Realisation in Software Design***
Assertion of power	Monumental design.	Industry and market domination, defacto and collaborative standardisation.
Allocation of power	Peer and professional assessment of buildings and artefacts.	*No obvious representation.*
Exemplification	Canonical discourse.	Patterns, components.
Tacit-explicit knowledge conversion	*Ad hoc* interpretations from images and first-hand experience of canonical buildings.	Pattern writing, pattern shepherding.
Knowledge transfer	Two-dimensional images in architectural and design publications.	Patterns, narratives, mentoring, case studies, folklore.
Design transfer	Canonised buildings and designs.	Planned transfers, handovers, structured reviews., mentoring.
Indicator of theory	Architectural academic discourse.	'High road' methodology, academic literature.
Indicator of practice	Canonised buildings and designs.	'Low road' methodology, patterns, components.
Pedagogy	Group collaborative design, guided studio instruction.	Individual and collaborative design and programming exercises.

Table 1: Purposes of a canon in both architectural and software design.

Canonical Scope

The notion of canon is, like all good abstractions, applicable on a number of scales. Downton (1998) suggests that we can usefully think about canons on personal and peer scales as well as on a global scale, and defines the 'personal canon' as those works that are exemplary *for an individual*. There is nothing absolute or 'right' about such a canon and an individual might recognise his or her canon as a subset of *the* canon. Ultimately, a 'personal canon' serves the individual designer by helping to organise knowledge for personal practice rather than the collective.

The act of forming one's personal canon differs between architects in the two realms. The maturity and explicitness of a software designer's 'personal canon' is easily taken as a mark of the individual's design maturity and experience, because the process of reading, understanding and internalising software design solutions takes time and considerable hands-on experience. The recent explosion of patterns study groups around the world evidences this investment. The motivation is often not purely altruistic—a software architect with a deep understanding of a broad personal design canon is likely to be perceived as an experienced and marketable individual. But the same is true of the commercial architect. The time and effort invested in mastering software design alternatives is generally appreciated, but by contrast, architects of the physical world assume that they can understand and digest canonical works from two-dimensional magazine images and brief commentaries from eloquent observers.

The idea of personal canon creates the argument that the legitimators of the canon, those with the power to prescribe it in a given place and time, simply promote personal canons publicly with, to varying degrees, publicly scrutinised grounds for including and excluding works. To an extent their authority is derived from the quality of their arguments for particular inclusions and exclusions.

Downton also defines a 'tailored canon' as one that contains works of limited applicability, but held as significant for a group or school, particularly as exemplars or as carriers of themes and ideas of special importance to that school or view. Tailored canons may hold a rather particular selection of works to support a skewed position, and have local application and significance only. Within *the* canon, there is perhaps an inner canon of works of indisputable membership and centrality which co-exists with the strict canon, the outer canon. An alternative conception of canon offers a graduated degree of membership or centrality. Some works are held to be of the greatest possible significance; other works are simply good examples of a school or a particular architect, others are tenuously connected through their association with more significant works. In the architectural canon, these distinctions are not made explicit but rather are perpetually debated. In the software design canon, to the degree that one exists, the space of canonised designs is more distinctly divided by narrow problem domains and solution technologies, and this provides a basis for determining canonical scope. Noble's 'Patterns for Small Memory Systems' (2000) is a good example. Nevertheless, as with the architectural cannon, the realm of software design currently has no way of conducting a centralised discourse on the validity

of specific solutions or practices, and the software patterns movement continues to generate equivalents of Downton's 'tailored canons' in specific design domains.

Canon as an Index of Types

A substantial value claimed for the architectural canon is as a catalogue of exemplary design types. This claim, described using the following pedagogical scenario, parallels the use and value of named software design archetypes closely:

> *The architecture student, who, in explaining a design project, has recourse to the Pazzi chapel, is likely to be doing so in relation to a question from the instructor about the definition of space. What is meant will be something like '…here I intend that a complex volume will be articulated so as to appear as the interpolation of smaller simpler solids'. Now, historians might be interested in whether Brunelleschi had such terms available to him, but to make the studio function all that is required is that students and teachers each have in their heads a number of models of buildings of agreed qualities (Macarthur 1998, p. 206).*

For most architects, Macarthur continues, history is merely the index to lots of interesting buildings. For practicing architects, the canon is less about the politics of value, inclusion and exclusion, and more to do with what vehicles of translation and comparison are needed in current (educational) discourse. In software design, the intangible and invisible nature of the software fabric has meant that past canonical designs live on only in the minds and memories of those who worked with them, in the kind of conceptual schemas used by experts in all fields. This had meant much rediscovery as software designers have tended to ignore history as a source of design exemplars. Design patterns fill this gap by providing a common template for naming and expressing design exemplars. As a result, students and teachers, mentors and apprentices can discuss architectural and design in terms of Pipes and Filters, layers and Blackboards, or a detailed design in terms of Observers, Facades and Proxies, and code-level design in terms of Iterators and Smart Pointers. This association between a name and a conceptual design archetype has a long history in computer science (Coplien 1998).

Visibility and Distortion

To be effective as a knowledge transfer medium, a canon must be accessible, visible, and reasonably free of distortion. In architecture, the canon typically takes the form of architectural anthologies, where canonical works appear as descriptions, discussions and images. The scope for ambiguity and even vanity afforded by these media is obvious. In this dimension, the architectural canon is open to a from of distortion that compromises meaningful comparisons.

Downton (1998) recognises the power of the fixed image and its ability to distort, along with other sources of inaccuracies in the ways the designers access the canonical works. These include the fact that many canonical images are fixed in time and do not show how the building ages, the loss of scale and important detail that occurs when an image is reduced to fit on a page, and the treatment of aesthetic appeal over functionality and habitability. This last point is particularly important—buildings and design are often judged from artistic perspectives that bear no relation to how the building's occupants perceive or occupy the building. Image as image, or image for image's sake, as typified by publications that are oriented in the space between architectural design and art, can defeat practical communication of all but the most basic information. Brand also criticises architects for allowing publishers to glorify the printed page architectural image, even to the point where architects have admitted designing for the photographic image (Brand 1994).

These limitations emphasise the propensity of the architectural canon to defeat its most important purpose—that of discourse—by falling for the seduction of image over substance, or the visual over the functional. There is often no option to see the building's plan. Assessments follow from perception of the image rather than occupation or habitation, and no assessment of how well the building works is possible. Critical factors such as maintenance costs and ease of extension cannot be assessed. The architectural canon is value-laden and this is both its biggest weakness and its source of ongoing relevance.

Most software patterns (as a candidate software canonisation mechanism) shines in the face of each of these criticisms. The pattern is described unambiguously and transparently using static class and dynamic object interaction models, and with extensive discussion that relates the pattern's solution to its problem and context. Design patterns such as those of Gamma (1995) and many other authors (Buschmann and Meunier 1995; Foote and Opdyke 1995; Pree 1995; Schmidt 1995; Taylor 2000) typically include detailed discussion of solution alternatives, explanation and justification of the recommended solution, factors that the designer should know about before choosing to use the pattern, factors that assist the designer in assessing the pattern's fit to a particular problem and its context, as well as discussions of known uses, alternative forms, and relationships with other patterns and architectures. Because each individual use of a software design pattern necessitates the

designer to understand the pattern's intention and purpose, its function and mechanism, any hint of obfuscation or lack of clarity in any part reduces the likelihood of the pattern being widely understood, adopted, and evolving under the natural selection of collective evaluation and use.

THE CANONISATION PROCESS

In architecture, the works in the canon are those deemed significant at a point in time. The rise and fall of a canonical work parallels the history of its significance. Some works go from no argued significance to acquiring great status. Mies van der Rohe's German Pavilion at the World Fair is an example of an architectural work that increased in importance after the event, while the relative importance of Gothic architecture to contemporary architects is reducing as a canonical style. The canon is regarded as an evolving organism: "*a* or *the* canon cannot be complete unless and until time, architecture, criticism and history cease...it is constantly in the process of renegotiation and redefinition if not re-conception" (Downton 1998, p. 44).

Inclusion

Inclusion into a canon is granted on the basis of consistency with currently perceived values. Most often it is the qualities or properties that works possess that first earn them a recognition that is further cemented by consideration of other factors. Canonical works, in all media, are works of quality—works that exemplify particular qualities deemed significant at the time. In the built world, consensus on the canon is not required, nor is it even desirable. Macarthur (1998) observes that we should not make the mistake of assuming that consensus on the canon is vital to its existence, or that the canon demands some consensus of us. The canon's existence is guaranteed by the necessity of exemplification in design practice, and it would be a failure if discord about it were less strident than opinions about the full range of contemporary architecture and design.

The design pattern mining and authoring processes that have developed and been used widely in recent years reflect similar levels of acceptability but less tension over inclusion. The process, in summary, is as follows:

- A practitioner identifies a potential pattern—it should not have been described before, it should be borne of the author's direct experience, and it should solve a specific and recurring problem with a proven solution;
- The author drafts the pattern(s)—this usually involves a good deal of introspection as the author grapples with the problem being solved, the forces in the problem context that demarcate the problem and make it unique, and the clear expression of the known solution;
- Each drafted pattern is related to other patterns, both in the author's pattern language and in other languages—this phase of pattern mining is possibly the hardest, and is rarely done well at first attempt.

Once a design solution has been expressed as a pattern, it has a *name* which allows it to be referenced, a description of the *problem context* that clarifies when the pattern may be applied, a statement of the *forces* that exist in the problem context, the *solution* itself, and the *consequences* of applying the pattern's solution—all of which adds precision to the application of the pattern's encapsulated knowledge. While the solution is an important part of a pattern, it is these other components of a pattern that *situate* and *orient* the solution, and distinguish the pattern from a simple expression of a recommendation, a heuristic, a business rule or an axiom.

Pattern authors are encouraged to draw on the help of the patterns community as they move beyond a first draft. Since 1994, a series of pattern writing conferences have been held annually in the United States, Germany, and more recently Australia and South America, for the sole purpose of reviewing patterns. When a pattern author submits a draft, the conference committee allocates a 'shepherd'—a more experienced pattern author—to work closely with the author in refining the draft patterns. What follows is an exercise in literary criticism, where the shepherd guides but does not control or dictate the emerging work. Shepherds sometimes find that the authors pull out, either because their potential pattern dissolves under scrutiny or because the author cannot commit to the iterative process. Once the draft has been shepherded, typically over a 6 to 8 week period, the shepherd makes a decision in conjunction with a program committee member to accept the work into a writer's workshop, a group review protocol similar to that used for many years by poets.

At the conference, the writer's workshop inverts the conventions of conference paper presentations. The model is based on a traditional form of poetry workshops in which tight communities of peer poets and writers review each other's work in the strictest confidence. Workshop attendees will have read the patterns beforehand, and time is explicitly scheduled for this important preparation. The author then becomes a 'fly on the wall'—not speaking a word, but listening as the workshop reviewers, many of whom are authors themselves, work through the pattern by following a structured agenda and discuss their understanding of the pattern. This allows the author a rare opportunity to hear how others have understood and interpreted the pattern. Misunderstandings become immediately clear during this time. The workshop concludes when the author re-enters the circle and

has the opportunity to request clarifications of the reviewers. The workshop process is carefully monitored by experienced pattern shepherds to protect the dignity of the author amidst often blunt and open criticism.

RESEARCHING THE NEED FOR A SOFTWARE CANON

The architectural canon is compromised by the enforcement of values and the vainglorious interpretation. Is a value-free architectural canon possible, or does its existence depend upon a lack of distinction between the observer's personal values, perception of aesthetics and the re-expression and reinterpretation of forms that have gone before? Must a software canon be completely value-free, or do value-laden interpretations drive the discourse?

Despite the self-explanatory and self-justifying nature of the pattern form with its elaborations of context, problem, forces, solution and resulting context, it is essentially a structure for comparing designs. Even in the most rational view of software and systems development, there is place for human judgement in the act of design. The creative act of synthesis in the fact of multiple conflicting constraints and fabrics, the generation and assessment of alternatives, and the final implementation decisions are typically based upon pragmatic rather than theoretical or provable factors. Even rational design processes are frequently unrepeatable, and it is only the hardest of formalists who continue to shrink from this fact. How then can design patterns and a pattern-based canon achieve the utopia of value-free design?

The Situated Software Architect

It is likely that they cannot, and should not even attempt to. The role of values is one research theme in a study that is addressing the factors and mechanisms that effect situated software architecture and design in business and industrial contexts. Situatedness refers to the ability of actors to enact strategies, plans and decisions on the basis of contextual factors rather than *a priori* plans and conceptions of the world (Gero 1998a; Gero 1998b; Gero and Kulinski 2000; Suchman 1987). The aim of this study is to move away from questions of why particular case-studied projects succeeded or failed, and away from specific methodologies or even attitudes toward adoption, and toward a focus on the design practices of the 'situated software architect'. Interviews will be conducted with highly experienced software architects working in business and industry with current software development technologies, to assess the following themes:

- The situated design process—the way that the business-driven context shapes the personal design process;
- Reflection, situatedness and action—the modes of designing, including attitudes and approaches to reflection, feedback, the effects of situatedness on the design process, and how and why action occurs;
- Access to, and codification of design knowledge—the ways that designers find, develop and formulate design solutions under pressure and other constraints;
- Design transfer, and design knowledge transfer—the ways that software and systems architects transfer complex software architectures, and transfer knowledge about these artefacts;
- Situated design maturity—an understanding will be built as to the architect's perception of the meaning of maturity in software and systems architecture and design, both their own and their peer's.

The research will produce a multi-dimensional qualitative analysis and conceptual models of the experience of software architects practicing in commercial contexts. Of particular relevance is the focus on the personal designer's access to shared design knowledge. It is expected that this research will provide substantial illumination on the flow of software design knowledge in a professional designer's team, network and immediate context, and that this will in turn substantiate or refute the arguments for a software canon, its purposes, and its most useful form.

The Need for a Software Canon

Whether the architects in the proposed study recognise a need to externalise, exchange, and enrich their individual experience is yet to be seen. Another critical question will be the form that designers need to receive canonical design knowledge—or, more bluntly, what kind of 'work' they are prepared to do to participate in the process of contributing and receiving. Representation in a canon of any kind allows transmission through both time and space, so that the embodied knowledge can be learned and passed on. Often this is by emulation of parts or aspects of the canonical artefact. Learning involves reaching a decision about the significance of a work, an investment that will typically be assessed by the learner at every point. A software design canon will be successful if it effectively serves to ease the burden of learning specific design skills and shortens the path to a given level of experience.

The canon further serves the function of being a cumulative cultural repository of all the works deemed to have significance, importance in the unfolding history of the discipline. The claim is that these are works that are

valuable to know about because they are works of quality and as such mark a stage in the unfolding of an enduring theme. Thus the canonical works, not rules, are the principle conveyors of knowledge of qualities deemed significant or appropriate to emulate. A software canon based upon design patterns looks like a promising medium to record the progress of design history within software engineering, allowing rediscovery, reinterpretation, and knowledge development exactly as the architectural canon does.

Canons transmit knowledge through habitation and contact, however that is possible. In the built world, Downton (1998) claims, "it is through the experiencing of canonical works or their representations that the knowledge embodied is transmitted to others and, if these others are architects or students, to new and possibly canonical works" (p. 45). So the canon is enriched through use and feedback, and if software patterns are to become the *lingua franca* of a software design canon, then a rapid feedback and assessment mechanism is necessary. Similarly it is this embodied knowledge that serves as the foundation for the conduct of theoretical and historical inquiry and thus the creation of further knowledge. New knowledge is formed when tacit knowledge is made explicit and combined with other forms of knowledge (Nonaka and Takeuchi 1995). The architectural canon has worked this way for centuries, but until recently software design knowledge has lacked the common medium for even the most basic form of exchange. Patterns promise to fill this gap by providing a more useful means to describe canonical design practices in which code and source-level mechanisms feature but do not dominate in the attempts to communicate design intentions and solutions.

CONCLUSION

A number of important observations can be made from this comparison of how design knowledge is managed in both the architectural and software realms. Firstly, software design's disconnection from the wider design discourse may be interpreted as a mark of its immaturity, one which follows from the seemingly irreconcilable design practices and construction fabrics. Many authors are recognising that the core of design, an essentially human creative activity, is common across many media (Budgen 1995; Gero 1998b; Lawson 1997; Love 2000; Owen 1998; Oxman 1999; Thackara 1988). As a result, re-examination of the notion of 'canon' as knowledge repository and transmitter is timely—to software design, because it needs to find a way of globally sharing best-practice, canonical design knowledge—a way of propagating this knowledge as rapidly as user interface widgets, web page animations and graphics, and browser plug-in components propagate to millions of people every day.

If the architectural canon re-presents the rhetorical question 'how then should we design', what provides this function for the discipline of software design? Until recently, some evidence suggested that software architecture was little more than craft (Shaw and Garlan 1996), an *ad hoc*, reactive game with designer as pawn and business forces as manipulative players. But software's missing design discourse may be finding voice in the nascent design patterns movement. For the software engineering profession to gain additional benefits of a software design canon, mainstream acceptance of the pattern form must be gained by fostering language-independent (architectural) patterns and new pattern mining efforts in business software development environments, and by finding new ways of making the canon visible and accessible via web-integration with integrated development environments.

Beyond these pragmatic issues, design patterns can express design practices, design justifications, design values, and a history of design that no medium, code or otherwise, has yet been able to do.

REFERENCES

Alexander, C. (2000). "patternlanguage.com." *http://www.patternlanguage.com.*

Brand, S. (1994). *How Buildings Learn: What Happens to Them after they're Built.* Penguin, New York.

Budgen, D. (1995). "Design models from software design methods." *Design Studies*, 16(3), 293-325.

Buschmann, F., and Meunier, R. (1995). *Patterns of Software Architecture: A System of Patterns*, Addison-Wesley, Reading, Massachusetts.

Coplien, J. O. (1995). "A Generative Organisational Pattern Language." Pattern Languages of Program Design 1, J. O. Coplien and D. C. Schmidt, eds., Addison-Wesley.

Coplien, J. O. (1998). "To Iterate is Human, to Recurse, Divine." C++ Report, 43-51.

Coplien, J. O. (1999). "Discussion of canonical software, during Software Patterns workshop.", TOOLS Pacific, Melbourne.

Downton, P. "The Canon: a site of architectural epistemology." *FIRM(ness commodity DE-light?: SAHANZ questioning the canons: Papers from the Fifteenth annual conference of The Society of Architectural Historians*, Melbourne, 43-49.

Foote, B., and Opdyke, W. F. (1995). "Lifecycle and Refactoring Patterns that Support Evolution and Reuse." Pattern Languages of Program Design, J. O. Coplien and D. C. Scmidt, eds., Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software Architecture*, Addison Wesley, Reading, Massachussets.

Gero, J. S. (1998a). "Conceptual Designing as a Sequence of Situated Acts." Artificial Intelligence in Structural Engineering, I. Smith, ed., Springer, Berlin, 165-177.

Gero, J. S. (1998b). "Towards a model of designing which includes its situatedness." Universal Design Theory, H. Grabowski, S. Rude, and G. Grein, eds., Shaker Verlag, Aachen, 47-56.

Gero, J. S., and Kulinski, J. "A situated approach to analogy in designing." *CAADRIA 2000*, Singapore, 225-234.

Grabow, S. (1983). *Christopher Alexander: The Search for a New Paradigm in Architecture*, University of Chicago Press, Chicago.

Hauffe, T. (1998). *Design: A Concise History*, Laurence King Publishing, London.

Lawson, B. (1997). *How Designers Think*, Architectural Press, Oxford.

Lea, D. (1998). "Christopher Alexander: An Introduction for Object-Oriented Designers." (Contact author for details).

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

Macarthur, J. "Some thoughts on the Canon and Exemplification in Architecture." *FIRM(ness commodity DE-light?: SAHANZ questioning the canons: Papers from the Fifteenth annual conference of The Society of Architectural Historians*, Melbourne, 203-208.

Naylor, G. (1990). *The Arts and Craft Movement: A Study of its Sources, Ideals and Influence on Design Theory*, Trefoil Publications, London.

Noble, J., Weir, C., and Bibby, D. (2000). *Small Memory Software: Patterns for Systems with Limited Memory*, Addison Wesley.

Nonaka, I., and Takeuchi, H. (1995). *The Knowledge-Creating Company*, Oxford University Press.

Owen, C. L. (1998). "Design research: building the knowledge base." *Design Studies*, 19(1), 9-20.

Oxman, R. (1999). "Educating the designerly thinker." *Design Studies*, 20(2), 105-122.

Pree, W. (1995). *Design Patterns for Object-Oriented Software Development*, Addison-Wesley.

Schmidt, D. C. (1995). "Experience Using Design Patterns to Develop Object-Oriented Communication Software." *Communications of the ACM*, 38(10).

Shaw, M., and Garlan, F. (1996). *Software Architecture: Perspectives on an Emerging Discipline*, Addison-Wesley.

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

Taylor, P. (2000). "Capable, Productive and Satisfied: Patterns for Productive People." Pattern Languages of Program Design 4, N. Harrison, B. Foote, and H. Rohnert, eds., Addison-Wesley, Reading, Massachusetts, 611-637.

Thackara, J. (1988). "Beyond the Object in Design." Design After Modernism, J. Thackara, ed., Thames and Hudson, London.

Venturi, R., Scott Brown, D., and Izenour, S. (1977). *Learning from Las Vegas*, The MIT Press, Cambridge, Massachusetts.

ACKNOWLEDGEMENTS

This work on design knowledge transfer originated in 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