Software, Architecture, and Participatory Design

Stephen Rank
Faculty of Applied Computing Sciences
University of Lincoln
srank@lincoln.ac.uk

Carl O'Coill
School of Architecture
University of Lincoln
cocoill@lincoln.ac.uk

Cornelia Boldyreff
Faculty of Applied Computing Sciences
University of Lincoln
cboldyreff@lincoln.ac.uk

Mark Doughty
Faculty of Applied Computing Sciences
University of Lincoln
mdoughty@lincoln.ac.uk

ABSTRACT

Much work in software architecture has been inspired by work in physical architecture, in particular Alexander's work on 'design patterns'. By contrast, Alexander's work is little-used in town planning and architecture. In this paper, we examine some of the reasons that this is so, describe some parallels and differences between the fields of physical and software architecture, and identify areas in which future collaboration may be fruitful. The notion of 'participatory design' is important in software engineering and in urban regeneration, but the participatory mechanisms in each field are quite different.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.10 [**Software**]: Software Engineering—*Design*; D.2.11 [**Software**]: Software Engineering—*Software Architectures*; K.4.2 [**Computing Milieux**]: Computers and Society—*Social Issues*

General Terms

Design, Human Factors

Keywords

Software architecture, participatory design, physical architecture

1. INTRODUCTION

Researchers at Lincoln University's Faculty of Applied Computing Sciences and the Lincoln School of Architecture have been engaged in a two-level application of participatory design; software engineers are working with architects to design collaborative systems that architects can apply in participatory design exercises with their user communities. The first such system involved the use of computer game software to help residents in Hull, UK, to visualise and interact with a design proposal for a 'Home Zone' in their neighbourhood [29]. In this paper, we examine some of the ways in which architecture and software engineering are related, and describe important differences between the fields that are often ignored.

2. PARALLELS AND DIFFERENCES

In principle, software is technically easier to evolve than buildings; software is not constrained by physical laws in the same way that physical artefacts are. However, software evolution is a very hard problem, as physical constraints are far from the only forces acting on a software project; business and social forces are more important in software engineering. Faced with the problems of evolving software, software engineers continue to look outside their field for inspiration. Software components, like buildings, are situated within and have a direct effect upon an environment; the notion of co-evolution of the software with its environment has been explored [21, 20]. Evolution of buildings is, by its nature, an activity that requires many disciplines: architecture, civil/structural engineering, construction, *etc*; software evolution is similarly interdisciplinary, requiring not only software engineering skills, but also business knowledge, social informatics and human factors and other skills.

The area of overlap between architecture and software engineering most frequently referred to by software engineers is concerned with theory, to be precise, the writings of the architect Christopher Alexander [1, 2]. Alexander's work in 'design patterns' has inspired considerable interest in the software engineering community. Design patterns provide a succinct means of documenting stereotypical design solutions and are particularly relevant in solving design problems that arise repeatedly. In software engineering, the field of design patterns is large with a rapidly expanding community [17, 12, 3]. In contrast, Alexander's ideas have not been accepted with anything like the same degree of enthusiasm by his architectural peers; his work is little-read in UK or US architecture schools today. Why has Alexander found favour outside his own profession rather than within? This question brings to light some key differences between the goals of software engineers and building architects.

For the architect, artistic creativity and invention are core attributes of the design process. Originality in building form, fabric and programme is seen as highly praiseworthy. Think of the convoluted, curvaceous forms of Frank Gehry's Guggenheim Museum in Bilbao, or the 'gherkin shaped' new Swiss Re building in the City of London by Norman Foster and Partners. In contrast, Alexander's design patterns encourage the reproduction of established architectural forms and spatial arrangements found repeatedly in vernacular architecture, a supposedly 'timeless way of building'. Consequently, his writings and buildings are thought of badly by many architects. As William Saunders, editor of the Harvard Design Magazine points out:

> [Alexander] has little 'cultural capital', particularly in architecture schools in which newness, art, and complexity are valued, and belief in timeless and universal human needs is considered naïve [32].

Conversely in the software patterns community, the reuse of ideas in software design is an explicit goal. The aesthetic qualities of the design of a piece of software are never (directly) perceived by the end-user, while the opposite is true of the design of a building or urban area. Software architects and designers aim to produce software that functions correctly, is easy and cheap to create and modify. Ensuring usability and determining softwares appearance are not normally considered part of the software architect's role, while these are important aspects of a building architect's job.

Some might argue that the architectural profession has been too quick to dismiss Alexander's ideas, that there are valuable lessons to be learned from vernacular building traditions and that ceaseless innovation is not appropriate in all circumstances. However, Alexander's work has been criticised on a more fundamental level. His pattern language seems to offer a basis for a social analysis of the built environment, yet he provides no coherent theoretical foundation for his interpretation of built form.

Alexander began his research at a time when structuralism was most prevalent in academia and, on a superficial level at least, his quest to uncover deep-seated, timeless truths underlying society's relationship with architecture is comparable to the semiotic analyses of the English architectural academic Geoffrey Broadbent [10] or the architectural anthropology of Amos Rapoport [30]. As such, his work exhibits many of the same shortcomings that contemporary post-structuralist and post-modern authors have highlighted in relation to structuralist theory in general. The concern with supposedly timeless patterns betrays a view of society as static or unchanging rather than dynamic. It also offers very little scope for conflict, the possibility that users might disagree with each other or the author over what constitutes an acceptable design pattern. This failure to theorise conflict is further reflected in Alexander's naïvety regarding the political and economic constraints to implementing the recommendations he makes.

While Alexander's work has attracted several criticisms at a theoretical level from physical architects, there is little (if any) software engineering literature which takes this viewpoint. In fact, there is little (though non-zero) criticism of the design pattern movement in general. Contemporary research is concerned with expanding the range of software design patterns to more abstract software architecture and more concrete implementation patterns.

3. PARTICIPATION

Participatory design is another area where there are some parallels between architecture and software engineering. It is a growing practice that has been adopted by practitioners in both fields [33, 36, 31, 37]. In both cases, it is particularly relevant in projects with a large public user base. In software engineering, web development is one such area, particularly web-based collaborative environments and publicly-funded web applications. In architecture, community buildings and area-based urban regeneration initiatives are the main arenas of participatory design practice. Participatory design practitioners aim to ensure that the community who will inhabit a new building or neighbourhood improvement scheme are properly represented during the design process. This is done in various ways, using tools and techniques that enable non-architects to become directly involved in decision making about architectural problems. In the past most of the techniques used were manual, such as drawings, physical models, or interactive displays, and were used in 'face-to-face' situations such as workshops and exhibitions. However, recently, academics and practitioners have begun to explore applications of digital media to participatory design, looking at interactive websites and geographic information systems (GIS) in particular [19, 4]. The application of virtual reality and computer game technology to design collaboration in architecture and urban design is an interesting and relatively new development [16, 23, 29].

There has been a long tradition of encouraging end-user-participation in the design of software systems, particularly within the Computer-Supported Cooperative Work (CSCW) community [8, 9, 6, 7]. This dates from the early participation of Scandinavian workers in the design of computer systems introduced into their work places. With the possibility of more universal human computer interaction, usability studies have become a common practice within computer system design [25], especially in the design of web-based systems [26]. This focus on the needs of the potential end-users rather than customers [27] marks a radical change from earlier system development where the commissioning customers were often the end users, or their direct representatives.

The theory and practice of participatory design is also well established in the architectural profession, dating back to the 1960s and 70s when pioneers like John Turner, Lucien Kroll, Rod Hackney and Ralph Erskine first began to extol the benefits of user involvement in the design process [37]. After a period of decline in the late 1980s and early 90s, the field has been given new impetus by changes in UK urban policy endorsing the notion of community involvement in the physical renewal of cities. Many architectural practices have capitalised upon this new trend and are now offering a range of community consultation services to public and voluntary sector clients alongside more standard architectural services. In addition, there is a growing body of literature examining tools and techniques that can be used to promote public participation in architecture, urban design and planning [18, 24, 38, 36, 31, 14]

In software engineering, there are some comparable ideas and these are receiving increasing recognition. The most recent edition of Sommerville's *Software Engineering* includes a new chapter on the socio-technical systems which emphasises the fact that software is often part of a much larger system involving many engineering disciplines as well as human, social and organisational factors [34]. Within the agile software development community, software development is considered as a co-operative game of invention and communication involving developers and customers where work products of the team should be measured for their sufficiency with respect to communicating with the target group of users [13]. The most widely-known agile software development method, XP, explicitly advocates having a customer always on the developers' site [5]. Nielsen recommends that usability studies are carried out with representative users [25].

One important barrier to non-specialists' participation in either field is the language and conventions that define the professions. In architecture, non-professionals often find paper-based plans and technical drawings difficult to understand. Virtual reality can be used to help participants visualise design proposals in three dimensions, either in the form of video-based animations of digital models or interactive 3-D models based on computer game technology [29]. However, buildings have an obvious representation (their physical form), whereas no such visualisation exists for software, which is 'inherently unvisualizable' [11]. Languages such as UML have been used to communicate with non-specialist users, but these are of limited value to non-technical users.

Professional attitudes can also act as a barrier to user participation. 'High' architecture is often seen as a closed professional community, not interested in input from outsiders. While all architectural activity is necessarily a product of inter-professional collaboration and teamwork, the field is rigidly hierarchical, underpinned by a system of peer evaluation which gives the impression that the most significant buildings are designed by individual geniuses or 'star architects' [35]. Participatory Design is opposed to this mode of thinking. Consequently, it is often viewed with suspicion by architects, represented as an adulteration of the design process and of lower-status than more 'artistic' architectural endeavours.

Correspondingly, in software engineering, open source software engineering is often criticised for its lack of rigour and mature processes. It too has the potential to be more democratic, involving more stakeholders in its design, development, and evolution. Currently, however, users are often restricted to the highly technically literate, and there is little involvement of non-programming users in many open-source projects. There are a small number of exceptions, but there is currently no clear way that an open-source project can easily take advantage of the expertise of non-technical users. Most open-source software projects (indeed, most software projects of whatever kind) are unsuccessful, and there are still no silver bullets guaranteeing success.

Attempts to involve stakeholders in design (of software and of the built environment) give rise to conflicts. In architecture, participatory design practitioners have developed techniques to make conflicting user needs visible to participants, to encourage dialogue and to build consensus between users. In software, conflicts are often resolved in shorter order, though there is increasing recognition that consensus amongst stakeholders is not automatic, and must be strived for, particularly at early stages in the lifecycle. Using tools to support detection and management of inconsistencies is becoming more common, along with the acceptance that inconsistency is a reality in any large software system [22, 28], and it has been recognised that "an important part of the requirements engineering task is facilitating collaborative work, consensus building and negotiation between stakeholders" [15]. There is potential here for software engineers to learn from architects with experience in participatory design.

4. CONCLUSIONS

There are many parallels between the fields of physical architecture and software architecture, though there is not a simple 1:1 correspondence between them. This paper has identified some areas for collaboration, and some differences, between the two fields. Table 1 summarises the main similarities and differences between work in the two fields.

Collaboration between architecture and computer science in Lincoln is being taken forward with work in the design of collaborative environments to support participatory design in urban regeneration. We aim to develop software simulators which will enable non-experts to gain an understanding of proposed architectural designs for urban areas which are being regenerated. These simulators will use 3-D visualisations of urban areas to present users with proposed designs. The simulators will be embedded within collaborative environment allowing potential users to communicate directly with the architects involved in the design, thus facilitating their further participation in the design process.

5. REFERENCES

[1] C. Alexander. *Notes on the Synthesis of Form.* Harvard University Press, 1972.

[2] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. *A Pattern Language: Towns–Buildings–Construction.* Open University Press, 1977.

[3] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice.* S.E.I. Series in Software Engineering. Addison-Wesley, 1998.

[4] M. Batty. Models in planning: Technological imperatives and changing roles. *Journal of Applied Earth Observation and Geoinformation*, 3(3):252–266, 2001.

[5] K. Beck. *Extreme Programming Explained: Embrace Change.* Addison-Wesley Professional, 1999.

[6] J. Blomberg, J. Giacomi, A. Mosher, and P. Swenton-Wall. Ethnographic field methods and their relation to design. In Schuler and Namioka [33], pages 123–154.

[7] J. Blomberg, F. Kensing, and E. Dykstra-Erickson, editors. *PDC'96: Proceedings of the Participatory Design Conference.* Palo Alto, CA: Computer Professionals for Social Responsibility, 1996.

[8] S. Bødker, J. Greenbaum, and M. Kyng. Setting the stage for design as action. In J. Greenbaum and M. Kyng, editors, *Design at Work: Cooperative*

Software Architects	Physical Architects
Concerned with structure	Concerned with appearance
Aim to reuse	Aim to be original
Patterns considered valuable	Patterns not always considered helpful
End result invisible to users	End result's appearance very important
Cost important	Cost important
End-user input important	End-user input important

Table 1: Comparisons: Software and Physical Architecture

Design of Computer Systems. Lawrence Erlbaum Associates, Hillsdale, NJ, 1991.

[9] S. Bødker, K. Grønbæk, and M. Kyng. Cooperative design: Techniques and experiences from the Scandinavian scene. In Schuler and Namioka [33], pages 157–175.

[10] G. Broadbent, editor. *Signs, Symbols and Architecture*. David Fulton Publishers, 1980.

[11] F. P. Brooks. No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 20(4):10–19, Apr. 1987.

[12] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. Wiley, 1996.

[13] A. Cockburn. *Agile Software Development*. Addison Wesley, 2001.

[14] D. Driskell. *Creating Better Cities with Children and Youth: A Manual for Participation*. Earthscan, London, 2002.

[15] A. Finkelstein. Requirements engineering: a review and research agenda. In *1st Asian-Pacific Software Engineering Conference*, 1994.

[16] T. Fukuda, R. Nagahama, A. Kaga, and T. Sasada. Collaborative support system for city plans or community designs based on VR/CG technology. *International Journal of Architectural Computing*, 1(4):461–469, 2003.

[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[18] T. Gibson. The real planning for real. *Town and Country Planning*, pages 187–189, July 1995.

[19] R. Kingston, S. Carver, A. Evans, and I. Turton. Web-based public participation geographical information sustems: An aid to local environmental decision-making. *Computers, Environment and Urban Systems*, 24:109–125, 2000.

[20] M. M. Lehman. Feedback in the software process. Position Paper at the SEA Workshop: *Research Directions in Software Engineering*, Imperial College, London, Apr.14–15 1997.

[21] M. M. Lehman and L. A. Belady. *Program Evolution: Processes of Software Change*. Number 27 in APIC Studies in Data Processing. Academic Press, 1985.

[22] W. Liu, S. M. Easterbrook, and J. Mylopoulos. Rule-based detection of inconsistency in UML models. In *Proceedings of the Workshop on Consistency Problems in UML-Based Software Development*, Fifth International Conference on the Unified Modeling Language, Dresden, 2002.

[23] C. Lou, A. Kaga, and T. Sasada. Environmental design with huge landscape in real-time simulation system. *Automation in Construction*, 12(5):481–485, 2003.

[24] New Economics Foundation. *Participation Works: 21 Techniques of Community Participation for the 21st Century*. New Economics Foundation, London, 1998.

[25] J. Nielsen. *Usability Engineering*. Academic Press, 1994.

[26] J. Nielsen. *Designing Web Usability: The Practice of Simplicity*. New Riders, 2000.

[27] D. A. Norman. *The Invisible Computer*. MIT Press, 1999.

[28] B. Nuseibeh, S. Easterbrook, and A. Russo. Making inconsistency respectable in software development. *Journal of Systems and Software*, 58(2):171–180, September 2001.

[29] C. O'Coill and M. Doughty. Computer game technology as a tool for participatory design. In *Proceedings of the eCAADe 2004 22nd conference: Architecture in the Network Society*, Copenhagen, Denmark, September 2004.

[30] A. Rapoport. *The Meaning of the Built Environment: A Nonverbal Communication Approach*. Sage Publications, London, 1992.

[31] H. Sannof. *Community Participation Methods in Design and Planning*. John Wiley, London, 2000.

[32] W. Saunders. Pattern language. *Harvard Design Magazine*, 16, 2002. Winter/Spring.

[33] D. Schuler and A. Namioka, editors. *Participatory Design: Principles and Practices*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1993.

[34] I. Sommerville. *Software Engineering*, chapter 2: Socio-technical systems, pages 20–42. Pearson Education, Harlow, UK, seventh edition, 2004.

[35] G. Stevens. *The Favoured Circle: The Social Foundations of Architectural Distinction*. MIT Press, Cambridge, Massachusetts, 1998.

[36] The Architecture Foundation. *Creative spaces: a toolkit for participatory urban designers*. The Architecture Foundation, London, 2000.

[37] G. Towers. *Building Democracy: Community Architecture in the Inner Cities*. UCL Press, London, 1995.

[38] N. Wates. *The Community Planning Handbook*. Earthscan, 2000.