THE KIND OF PROBLEM A SOFTWARE CITY IS

CHRISTIAN ULRIK ANDERSEN & Søren Pold

New urban interfaces introduce software to the city. To understand software cities we must compare the city with the software at a specific level. Building on the architect Christopher Alexander's idea of a 'pattern language' the article will present the activities that urban software fosters, and question how the underlying computational processes change the complex life forms of the city, and the response of urban planners.

The Kind of Problem a Software City Is

The final chapter in Jane Jacobs's *The Death and Life of Great American Cities*, from 1961, is entitled "The Kind of Problem a City Is." In it, she gives an account of the relations between urban development strategies and the progression of science. From being able to deal with problems of simplicity and disorganized complexity, science in the 20th century became capable of managing organized complexity. Statistical material and mathematics made it possible to manage a large number of variables as an organic whole. So-called 'urban renewal' projects were launched in both the United States and Europe to solve the urban problem of disorganized complexity. Slum areas that, according to statistics, had high rates of crime, infant mortality, and so on, were replaced with new and efficient infrastructures and a geographical separation of the use of the city into areas (residential, commercial, industrial, etc.). Several cities have been 'renewed' from the fifties onward by replacing urban areas of high complexity and diversity with new and functional areas of low complexity and diversity. Jacobs, however, claims that many urban planners do not know much about the actual interactions that take place in the city. [1] In contrast to this, they need "to think of cities as problems in organized complexity – organisms that are replete with unexamined, but obviously intricately interconnected, and surely understandable, relationships." [2] They must seek to think in processes that explain the general by the specific, rather than in statistical information that oppresses people and their relations.

In many cities today, the infrastructures no longer just consist of buildings, roads and paths but also of information flows embedded in software interfaces and networks (smartphones, tablets, wireless networks, surveillance systems, media displays, etc.). The software itself in many ways provides the means to effectively map out how the software city is used; for example, which smartphone applications are the most popular, what areas are under surveillance, or even the whereabouts and routines of an individual. Is it possible not to reduce our use of the software city to information overviews but instead focus on the specific use of the software city? Taking a lead from Jacobs, we need to ask ourselves what kinds of specific life forms occur in this environment and how we support their diversity and complexity. Architecture itself provides good techniques to do this.

A PATTERN LANGUAGE

In the 1970s, the architect Christopher Alexander, inspired by Jacobs, developed the notion of 'a pattern language.' A pattern is a way to summarize experiences, individual practices and practical solutions in a way that makes it possible for others to reuse them. Alexander's book comprises of 253 patterns, each with its own context, problems and solutions that sometimes help complete larger patterns or need other patterns to be completed. As an example, Alexander uses the pattern ‘accessible green’ based on the observation that people need open green places to go to; but when they are more than three minutes away, the distance overwhelms the need. Consequently, green spaces must be built “within three minutes’ walk […] of every house and workplace.” [3] This pattern helps fulfill larger patterns such as ‘identifiable neighborhood’ and ‘work community.’ [4]

According to Alexander, “towns and buildings will not be able to come alive, unless they are made by all the people in society.” [5] The general idea is that a successful environment depends upon an ability to combine physical and social relationships. The pattern language creates such combinations: it is a lively language, not exclusive to architects, that responds to the needs and desires of the people and thus connects architecture to people. Alexander’s book is a pattern language for towns, buildings and constructions, but this pattern language is only one amongst many. Any society, or even individual, will have their own languages to combine the physical with the social. However, the problem is that these languages are often not very sophisticated; people are unable to speak and must therefore develop their own language. [6] By suggesting that architecture and urban planning at the time was built on a language that was not refined, he implicitly raised the same critique of modern urban planning and architecture as Jacobs: architecture that is merely functional, and does not build on social relationships, is brutal. A successful environment must be sensitive to the needs, desires, hopes and aspirations of the people living in the environment.

If Alexander provided the beginning of a pattern language for towns, buildings and constructions, how developed, refined and sophisticated is our pattern language for software cities? To what extent do we build an awareness of our social relations, and not least our needs and desires for these relations, into the process of making the software city?

Alexander’s ideas have in many ways been more influential in the design of software systems than in architecture. This means that the pattern language for software development is quite sophisticated. Before we begin to critique the level of refinement in the pattern language of the software city, let us consider the experiences created in software development.

DESIGN PATTERNS IN SOFTWARE DEVELOPMENT (LESSONS TO LEARN)

Design patterns in software development express a relation between programming, design, and use. This is particularly evident in the Scandinavian participatory design tradition, which in the 1970s had already begun to focus on how to bring the different stakeholders of a workplace into the process of system development. Participatory design not only includes the design of human-computer interfaces but also the technical parts of system development. Ward Cunningham, a pioneer in this field, initiated the Portland Pattern Repository that in the mid-nineties was accompanied by the WikiWikiWeb: the world’s first wiki. It consists of numerous patterns that, using the schemata of Alexander in general ways, describe problems and solutions in graphical user interface design and programming. [7]

‘Ward’s wiki’ became popular because it allowed programmers to share and co-edit their experiences. By using this, they developed a sophisticated pattern language that combined the human use situation with the structure of the program; similar to Alexander’s vision of a pattern language for towns, buildings and constructions. The combination of physical and social relationships has, however, also been corrupted along the way.

Firstly, the sharing of a language, which was the starting point for the Portland Pattern Repository, has partly disappeared. With the exception of open and free software projects, the code of software is copyrighted and not accessible to other programmers. Secondly, and partly related to this, interface design patterns seldom combine the physical and social relations in an 'open' way. In today's interface design it is compulsory to meet the user at eye-level in order to combine the physical and social relations in a 'useful' way. Software successfully responds to the needs and desires of its users to write text, edit images, collaborate, socialize, play, and so on; and developers regularly consult its users via interviews, questionnaires, supervising debate forums or even by letting them produce add-ons to their program, like apps in Facebook or macros in *World of Warcraft*. However, it is not desirable to discuss our social relations merely in terms of 'ease of use.' Social relations demand openness to the actual patterns of the users. The patterns of use should not only encompass user-friendliness but also other ways of using, including unintentional use and even oppositional or critical use. If not, pattern language does not respond to the needs and desires of social life.

Massive multiplayer online games illustrate this very clearly. Here, life is obviously not restricted to what is 'in the box': participants produce strategy guides, additional stories, make t-shirts, etc. However, game developers will often try to seize control of these activities. One example of this is the *WoWGlider*, a third-party program that automatically controls a player's character via scripts. The program, and similar 'glider bots' are popular because they allow players to acquire game skills without being present in the game. In 2006, a "high ranking officer of Vivendi" and a lawyer for both Vivendi and Blizzard (the game developers of *World of Warcraft*) approached the maker of *WoWGlider (*Michael Donnelly) at home, and accused him of violating their copyrights. [8] Later, they also filed a lawsuit and his company was eventually sentenced to pay Blizzard six million US dollars. This is just one amongst several examples of how the interests of players and game developers are in conflict. It has nothing to do with the usability of the game but is entirely a political issue: in order to protect their software and develop their business, developers prohibit certain types of behavior.

Today, software is no longer just for work but a platform for social life, and examples similar to *WoWGlider* are growing in numbers. The challenge for participatory software design today is to not only include people's use in the design of software but also openness towards creative and alternative use. In this, political issues concerning ethics and ideology become increasingly important, and conflicting interests and unbalanced relations of power and control often restrict the development of a lively pattern language. Successful examples are rare, but are, for instance, found in the FLOSS movement (Free/Libre Open Source Software) that explicitly combines the openness of technical structures to, for example, individual freedom and the refusal of intellectual copyright. [9]

The software city is to a large extent a non-work context where issues similar to the ones that are noticed in social software are relevant. In developing a pattern language for software cities, the first step is to critically evaluate the way we currently combine the physical with social relations (technical infrastructures with the complexity of life), in the software city. Experiences with both social software and FLOSS are valuable in this context.

PATTERNS IN THE SOFTWARE CITY

The standard image of a software city is somewhere where media saturation is obvious and clearly visible, like Shibuya in Tokyo or Times Square in New York. However, less spectacular implementations of software in cities also demand attention.

We have previously studied the digital layers of the mid-sized Swedish town Lund. In general terms, we found that software is embedded via 'log-in spaces' (as found in restricted networks of various sorts) and 'iSpaces' (as found in the individual use of personal and mobile laptop computers, tablets or smartphones), paired with a 'hypertextual connectivity' that connects physical space with virtual networks (as found when, for exmaple, a location-based smartphone application links to a social web service). [10] Both log-in spaces and iSpaces suggest that the migration of software into urban space propagates two kinds of activities: surveillance and configuration.

As a general pattern, surveillance is not only visual but also structural. Surveillance does not only take place via cameras but also, and more often so, by following seamless transactions: for example, when logging on to a network, transferring money, or using personal identification numbers to keep records.

As a way of interacting with software, configuration means to change a system on a user level, including actions as diverse as image editing, setting software preferences or shooting monsters in a computer game. Configuration patterns are found when we use the software city to play games, socialize via services such as Facebook or Twitter, find weather reports, use a GPS for creating routes, etc.

Surveillance and configuration patterns reflect two particular and interrelated views on the public sphere: 1) When public wireless networks are restricted and its users tracked it is to avoid violations of copyrights, illegal conspiring, terrorism, and so on. Hence, surveillance fulfills a need to protect land and property. 2) When I, using my smartphone apps, can find information about the place I am, connect to my friends anywhere in the world or track my whereabouts and share them in public, the configuration pattern fulfills a need to support the exercise of the individual.

In this sense, inhabiting the software city is like inhabiting *The Sims*. The ability to perform as an individual in this environment is equal to the sum of acquired platforms (smartphones, credit cards, laptops, etc.) and applications (location-based guides or social networking software such as Facebook, Foursquare or Twitter) one possesses – similar to the acquirement of objects as a way to increase one's character level in a game. Every action is registered by the system, and you are given the right to configure the system, but you are never given the right of a citizen to negotiate the system itself. The problem of the software city is the same as in many computer games: life does not always fit into the box of the game. Actions such as evading surveillance, hacking networks and disrupting configurations are often considered illegal. Even common practices such as information sharing is considered a violation of copyrights. In this sense the physical infrastructures of the software city do not always correspond with the social practices and desires of the people. At the same time, we must also consider that life in the software city fits surprisingly well into a box. People do not mind 'living in *The Sims.*' Surveillance is widely accepted, and though CCTV is often part of public debate, registration of computers, tablets or smartphones on the Internet is widely accepted; which is similar to the tracking of money transfers when people use their credit cards or smartphone apps for Internet banking. Likewise, people generally do not critically evaluate their use of location-based or social services for their smartphones, as long as there is a benefit to them.

To discuss the software city in terms of 'ease of use' may prevent the development of a diverse software city. The software city must therefore encompass the idea of citizenship, where the combination of iSpaces or log-in spaces with our social relations can be debated openly and lead to new visions. This development of a shared pattern language for the software city also implies critique of urban planning.

PLANNING THE SOFTWARE CITY

Though our use of software in the city is rarely planned, the software cities themselves are planned, and are usually thought of as ‘smart cities’ and ‘media cities.’

Smart cities cover a range of initiatives. The general idea is that a city’s performance is dependent on its ability to support and include aware citizens in issues such as economy, governance and sustainability. This demands that, for example, the people can be smart: that they have the necessary infrastructures to communicate, a high level of education, a pleasant environment with secure health systems, no crime and a rich cultural life, and so forth. [11] This is supported by current developments in software, for example, cloud computing, where services are offered via a network and thus support the mobility of the citizens. However, city planners and governors can also use software for data monitoring, analytics and visualization to sustain the smart city. This may include a wide range of services that compute and visualize data for crime, education, traffic, energy consumption, etc. These are not only management tools but are also tools that involve the participation of the citizens, such as, for example, reporting the water quality via a smartphone application. [12] The general idea is to not only anticipate but also innovate through data analytics.

Smart city initiatives appear to combine a city’s physical infrastructure with its social and intellectual capital. Learning from our critique of the use of patterns in software development, we must however also realize that in the smart city urban life is programmed into the software in ways that do not always correspond with the citizen’s practices, needs and desires. To develop a sophisticated pattern language we should not only debate the ‘usability’ of cloud computing or how monitoring systems make a ‘safer world’ (patterns of surveillance and configuration), but also some of the issues that are at stake in social software. Smart cities, for instance, heavily affect our perception of private and public. In other words, we need to reflect on what a common good means; who owns the data; what it is used for; when it is public/private; how it is monitored; how it is visualized; what data visualizations are used for, etc.

Media cities involve using software as a spectacle that can evolve new neighborhoods. The urban renewal project MediaSpree in Berlin is an example of this, out of many across small and large cities. In the past decade, large property investments have been planned along the banks of the river Spree in the Eastern parts of Berlin. The aim is to create high profile architecture that integrates media, small and large-scale use with public access to the river. [13] For various reasons (including lack of capital) many of the MediaSpree initiatives have been interrupted and several places are temporarily occupied by cultural initiatives; including the Berlin techno scene and clubs such as Club Maria, Berghain, Bar 25 and Tresor. In MediaSpree, the integration of media in architecture via displays and façade media is aimed at creating a spectacle that attracts people. In contrast to this, the media integration of these clubs is almost invisible at street level, but nevertheless very evident. The club scene is part of a global network of electronic music enthusiasts, connected via blogs and other services for cultural co-production. Every weekend thousands of people fly to Berlin to take part in the scene. Even though the club scene is not a clearly visible architectural landmark, indeed it often strives to be invisible, it is a lively part of Berlin and an attraction that possibly even outshines many of MediaSpree’s initiatives in terms of city life and branding. [14]

The smart city and the media city each in their own way demonstrate how the software city often is controlled top-down by corporate and urban planners. However, the development of a diverse software city does not have to be spectacular or 'smart.' Although media architecture, cloud computing and data visualization should not be dismissed, there is a need to focus on the spaces in-between that do not have clear ownership, areas "devoid of meaning," as the Danish sculptor Willy Ørskov has defined the 'terrain-vague'; spaces that can be of potential significance. [15] These spaces may be physical locations, like the banks of the Spree, but they may also be spaces for new software-based practices. It is in these spaces that we see glimpses of the software city as something other than just log-in spaces or iSpaces. In other words, there is a need for a new pattern language for the software city that is based in emergent cultural practices appearing in the terrain-vague; a language written by people, rooted in their practices and a sustainable combination of social relations with the physical infrastructures of software.

ACKNOWLEDGEMENTS

This research has been funded by the Danish Council for Strategic Research, 09-063245, (Digital Urban Living).

References and Notes:

1. *Jane Jacobs, The Death And Life of Great American Cities (New York: Random House and Vintage Books, 1961), 10.*
2. *Ibid., 438-9.*
3. *Christopher Alexander, et. al., A Pattern Language: Towns – Buildings – Construction (New York: Oxford University Press, 1977), 305-8.*
4. *Ibid., xiii.*
5. *Ibid., x.*
6. *Ibid., xvi-xvii.*
7. *Cunningham & Cunningham, Inc., "Nouns and Verbs," http://c2.com/cgi-bin/wiki?NounsAndVerbs (accessed June 28, 2011).*
8. *Brandon Boyer, "Blizzard, Vivendi File Suit Against WoW Bot Creator," Gamasutra Website, http://www.gamasutra.com/php-bin/news_index.php?story=12848 (accessed June 28, 2011).*
9. *This has also been named a 'recursive public' concerned with the "maintenance and modification of the technical, legal, practical, and conceptual means of its own existence as a public." In: Christopher M. Kelty, Two Bits: The Cultural Significance of Free Software (Durham: Duke University Press, 2008), 3.*
10. *Christian Ulrick Andersen and Søren Pold, "The Scripted Spaces of Urban Ubiquitous Computing – the Experience, Poetics, and Politics of Public Scripted Space," in Fibreculture Journal 19, ed. Ulrik Ekman, December 9, 2011, http://nineteen.fibreculturejournal.org/fcj-133-the-scripted-spaces-of-urban-ubiquitous-computing-the-experience-poetics-and-politics-of-public-scripted-space/ (accessed April 1, 2012).*
11. *European Smart Cities, Homepage, http://www.smart-cities.eu (accessed June 28, 2011).*
12. *IBM Website, "IBM: The Smarter City," http://www.ibm.com/thesmartercity (accessed June 28, 2011).*
13. *Wikipedia, "Mediaspree," http://de.wikipedia.org/wiki/Mediaspree (accessed June 28, 2011).*
14. *Tobias Rapp, Lost and Sound: Berlin, Techno und der EasyJet (Frankfurt/M.: Suhrkamp, 2009).*
15. *Willy Ørskov, Terrain-Vague (Copenhagen: Borgen, 1992), 27.*