Extreme Designing

Proposal for the transfer of concepts from the agile development to the architectural design process

Stephan Droste
Universität Stuttgart, casino IT
(http://www.casino.uni-stuttgart.de)
stephan@casino.uni-stuttgart.de

Abstract. *Obviously, design collaboration, the design process, and its methods are strongly interdependent. In order do understand collaborative processes and their requirements, methods of design process are focused prelimarly. After the hype during the last decades collaborative design seems to remain in a self-centred discourse with little concrete application outside the academic world, while in the same time collaboration is omnipresent in conventional architectural design. Interestingly, the initiation of the so called agile methods in software design were initiated by new tools and paradigms in software design and on the other hand defective conditions in the collaborative process, corresponding widely to the challenges of the architectural design process. This paper opposes principles of software development to the architect's approach to (early) design. Subsequently some implications for the extension of (collaborative) design tools are suggested.*
Keywords. *Design process; collaborative design; design methods; agile processes; software development.*

Introduction

Motivation

Starting point of this paper is the work on creative processes in teams and on their potential technical support. Based on a study, which explores the needs of teams during the creative planning process in architecture, a digital environment should be conceived. While the study led to several insights, the biggest one was the understanding that creative processes in teams of architects lack sufficient analysis and methods.

Coincidentally, out of the experience from software development the insight arose, that there are several analogies to architectural design. Furthermore the process of teams developing software has already been concerned and some of the resulting methods were widely spread in the past decade.

This led to the idea to oppose software development and architectural design and to possibly transfer practices from one to the other.

Problem

The creative process of designing architecture is hard to conceive. As it is described as argumentation

and bargaining (Rittel, 1963), the "externalization" in course of team activity seems to be rewarding. The basic methods of team creativity like brainstorming, synectics or morphological analysis as described by Kluge and Zysno (1993) are well approved, but appropriate to singular problems and not to complex designs.

To target the whole process of a design project, the comprehension of the management is essential. Since the creative team should not be organized hierarchically (Hacker, 1986; Kluge and Zysno, 1993) a self-managing system has to be established. Unfortunately, this kind of team organization especially in creative design projects in architecture is neither comprehensively described nor addressed by supporting software systems.

Basic approach

The parallels between software development and architectural design are astonishing. This is an aspect that is abet by Fabio Gramazio's remarkable statement in 2009 during a presentation in Stuttgart regarding the equalization of the activity of programming and of designing, basically as cognitive feedback processes. Comprehensible mainly by one who is trained in both arts, this analogy should be discussed and assimilated in architectural awareness.

Disposition

This work initially exposes an explorative study on the teamwork of students during a complex design project. The examination concentrates on the self-managed cooperation and collaboration.

Juxtaposing a compendious introduction to team-managed software design follows, with special attention to the agile methods. Subsequently the contingent transfer of terms and definitions is discussed. The conditions and requirements for the architectural angle are defined to propose the application of the conveyed methods.

Creative team process

General

Discussing the architectural design process requires a more accurate definition of the perspective and stage in the process. Many studies suggest the early consideration of external experts. But it is evident that the technical and methodic demands increase with every supported profession. In order to focus the architect's approaches, the examination is narrowed to the particular domain of the architect: the conceptual early design stage, for instance occurring typically in architectural competitions or design workshops. Incidentally the all-rounder architect must be able to advocate – a role yet to be defined - each related profession.

Still the process might be collaborative. In my perception collaboration is widely practiced - in academic design studios as well as in professional offices. Architectural designing is often characterized as iterative feedback process, chaining research, analysis, sketching, review, modeling, refacturing. Collaboration in the form of criticism, alternative solutions and assistance is common practice. In a sense the methodical approach of a single designer can presumably be taken as collaboration with himself.

The management aspect of process modeling usually is considered for the planning stage, which embraces multiple professions. In contrast the (early) design stage is mostly evaluated at personal level and implies locally limited experience. Thus, every team, studio and office cultivates a particular method.

Explorative Study

In order to understand the collaborative process, the opportunity arose to explore the teamwork of students in the second year at the University of Stuttgart, designing a complex construction in groups of five. Altogether, four teams where attended during studio work and presentations. Selected students where interviewed concerning their experience in

and estimation of the team and its development.

Basically, it was not intended to accomplish the research as a quantitative study, but rather in terms of finding general concepts and methods in the design team.

Observations

The teams where partially joined by the students themselves, while every team had some previously connected members. The mixture of gender was mentioned by most participants, but apparently did not affect the collaboration. The whole project was structured into three main phases with requested achievements, so an overall schedule was given. The instructor's supervision is ignored in this study, except for the resulting perception in the team. The daily routine of the teams was self-managed.

The rough evaluation is the declining tendency of discussion during the project. This is due to better understanding, division of work and time pressure or ignorance. It can be recognized that the teams are classified by different formation of division, whereas each team develops its position between cooperation and collaborating. The highly segmented teams typically cultivated cooperating experts, arguing mostly about interferences. The collaborating teams formed a more distinct identification with the project and the group. Noticeably, no team is clearly assigned to one side, in fact the differences occur on the several levels of work, like conception, technical consideration and (graphical or physical) implementation.

Digital tools are used extensively, but most participants adopted the use of programs during the project. So it could neither be expected a sophisticated appliance, nor the reasonable choice of systems. Mainly the group followed one "expert" or the majority. At times the programs are not used in the way they are proposed. Instead it is notable, how the familiar programs like instant messengers, email, chat systems and regular shared directories are used for the exchange of data, resulting finally in inconsistent data, distributed over the team's storages.

Insights

Basically the self-managed team process takes a while to be configured, since there are no methods for the comprehensive management. In contrast the singular team based problem-solving tasks are performed almost naturally. Design decisions are mostly made by classic methods like brainstorming and systematic analysis. Concurrent designs are elaborated and opposed, if the task is fundamental. On the other hand, in shortage of time often a voting is preferred.

The knowledge about the whole project leads to higher satisfaction of the individual and to more flexibility in the team. In many cases the teams split up into subteams, commonly pairs, to solve complex issues, which additionally supports the distribution of skills across the group.

The agile method

Software Development

Conventional software development is practiced in typical engineering manner. The complex projects are analyzed during an extensive planning phase, considering imaginable eventualities. Afterwards the project is broke down into easy to handle single issues and tasks. Finally the implementation of each component is passed to a programmer.

This approach is suitable for monolithic structures that can be well defined and valuated. This however is decreasingly possible, because of frequently modified requirements and the need for extensibility. Technically, from a programmers view, the changing specifications are controllable, since practices like object orientation and automated testing.

Extreme Programming (XP) is one example of the relatively new class of methods in software development that focuses on iterative development, collaboration and management processes, so called the Agile Software Development. The term was manifested 2001 in the Agile Manifesto as a common denominator of different protagonists in the field of collaborative process modeling. Related methods to

XP include Scrum, Crystal, DSDM and Adaptive Software Development. By now these methods are considerably used in companies of any size.

Extreme Programming

The XP was developed from 1996 in the course of a software development at Chrysler and is well-known due to early publication by Kent Beck (1999) and surely as a result of its ironic naming. Beside Kent Beck usually Ward Cunningham and Ron Jeffries are referred to as initiators.

Goal of XP is the organization of development processes, in a way that evokes increased quality, sustainability and productivity. In order to achieve this, a micro-incremental concept is developed that forces continuously reviewing of the design, the code, and the tasks. Furthermore team-building and knowledge sharing is supported as a basis for sustainable work. The ideas of XP are mostly specified as (best) practices to provoke a lean, self-organizing collaboration, common knowledge and cognition about the project.

Interesting on XP is the initial point that refers to the typical occurrence of software development. That is project-orientation, undecided clients, fuzzy definitions, infeasible deadlines and teams that are heterogeneous, fragile and finally stressed. This setup must be familiar for each architect: the conditions are apparently similar.

While the concept of a micro-incremental process and the general working conditions are alike, the consequences of the survey are astonishingly different. XP conceives a set of terms and rules to make the process traceable. This does not imply a formalized process, but in fact a lean formalization on a meta-level, that permits a dynamic adaption of the whole process. Sometimes XP is referred to as risk management rather than development methodology.

Methods

The XP simply consists of a set of yet proved methods, which are related with and support each other:

- On-site customer
- Planning game
- Metaphor
- 40hour week
- Refactoring
- Simple design
- Pair programming
- Testing
- Short releases
- Coding standards
- Collective ownership
- Continuous integration

The basic approach is to work as early as possible on a consistent, executable and tested prototype of software, that is developed further in short and small iterations. The whole management process targets the support of this idea. Furthermore the sole responsibility of each team member is advocated, which contrarily creates a less fragile team.

Transfer of the methodology

Comparison

Since programming by nature is computer related, it produces artefacts that can be evaluated technically. This nature lacks concerning architectural artefacts. It is expected, that the corresponding levels in this section lead to new approaches in the modeling of digital architectural tools to improve computer supported (collaborative) design.

In order to compare the activities of programming and designing, the tools used in the compared processes are analyzed to find common and dividing aspects. This includes mainly sketch, working model, drawing, (presentation) model and pieces of text for the architectural domain and code, models, architectures in the programming domain.

The conception can be divided in two levels, cognitive and technical. Although the cognitive possibility and the technical feasibility are interdependent, it is reasonable to attempt the comparison separately

Cognitive methods

Intriguingly, some of the basic methods, namely simple design, metaphor and refacturing, are strongly influenced by Ward Cunningham, as he transferred the Alexander's pattern language into programming practice. It can be supposed that the "back porting" might be managed.

Working by pairs is already practiced by architects and though until now the method is not named and estimated, its use is obvious. The same is the working time and the on-site customer, which are surprisingly counted as method. Coding standards are a matter of style and simplify the internal communication, the transfer to style guides is straightforward.

Finally the planning game is one of the key methods. The idea is to shorten the development cycles that result in a completed product. Every cycle starts with the planning game, so the working plan has to be simple and face issues that can be managed in short time. Again architects can easily understand this method; though in XP it is preliminary thought of frequent refacturing to adopt the architecture (hence this term also for structure of software) to the iteratively grown, more complex plan. This is maybe the crucial point, where architects have to abandon the habit, to keep (artistic) design decisions.

Technical methods

Short release cycles, collective ownership, and testing are more related to the technical opportunities. This becomes clear if one brings to mind that the more abstract coding of object orientation, the development of automated tests and resulting from those, the facile integration of code in the daily routine, led to the invention of the agile methods.

Technically these methods rely on a consistent set of data. In software development a repository that holds the actual and previous states of the project typically achieves this. Although it is technically possible, in architectural projects this kind of data management is usually practiced only in large offices and projects, because of the pretended management overhead. In addition, the conventional digital tools in architecture (pre-BIM/parametric) usually do not provide sufficiently structured data, particularly if several systems for different issues are used in parallel.

Testing is a central method to ensure the consistence of the present working state. In XP, if a proposed change leads to failing tests, the code has to be reviewed and adopted immediately, otherwise the last change has to be revoked. Testing from the architectural point of view is until now the most difficult method to transfer. The question is the one for description of quality in architecture. But also in software development the tests do not state the quality of code, which is achieved by reviews and refacturing. In fact, they process checks on the plausibility of methods as issued by programmers and functions as issued by the customer or user. This level related to architectural projects means results of simulation incorporating for example evaluations of physics, structures, behavior of users or legal regulations. Until now it is rather unusual to formulate simulations as automated tests, perhaps mainly because they often require specifically prepared data. Though it is not impossible.

Plan as code

As in the recent history of CAAD the integration of all kinds of data (geometry 2D-XD, semantic, properties, etc.) proceeds, it is to be expected that working on a centralized data base will be possible. Since usually only a single perspective on the data base is used to work on an issue, a centralized service, which might be capable to prepare this perspective, could be thought of. A promising approach in these terms is FREAC as described by Schneider and Petzold (2009).

To think this integration of data out, the difference between programming code and coded architecture disappears. Furthermore, if the iterative changes can be implemented in the real world and if in turn the environmental feedback has impact on the data, the whole built environment can be seen as the code to be modified by architects.

Summary

Teams of architects already collaborate in design. While each team based on the members' experience usually develops this process, it lacks an overall strategy of team-management. Extreme Programming brings together several methods in order to form such a strategy. Since architects already use the singular methods, the process as a whole should be adoptable. The problems yet to be resolved have rather technical character, and are already embraced in current development.

Conclusion

Due to new techniques, facilities and requirements programmers have performed a paradigm shift from planning of complex, static products to iteratively changing, applied systems. It seems as if the master plan is an inertial, uncertain instrument to face fuzzy problems. To come full circle to the theme of this conference, the new agile planning paradigm might show a possibility to handle complex, running systems in small iterations and constantly reviewed. This includes definitely ecosystems like cities.

References

Achten, H and Beetz, J 2009, 'What happened to Collaborative Design?', *27th eCAADe Conference Proceedings*, Istanbul, pp. 357-366.

Alexander, C 1977, *A Pattern Language*, Oxford University Press, New York.

Beck, K 1999, *Extreme Programming Explained. Embrace Change*, Addison Wesley.

Hacker, W 2002, *Denken in der Produktentwicklung*, Vdf Hochschulverlag, Zürich.

Hacker, W 1986, *Allgemeine Arbeitspsychologie*, Huber, Bern.

Kirk, SJ and Spreckelmeyer, KF 1988, *Creative Design Decisions*, Van Nostrand Reinhold, New York.

Kluge, A and Zysno, PV 1993, *Teamkreativität*, Minerva-Publ., München.

Rittel, HWJ 1963, 'Die Denkweise von Planern und Entwerfern' in Reuter, W (ed.) 1992, *Planen Entwerfen Design*, Verlag W. Kohlhammer, Stuttgart.

Schneider, S and Petzold, F 2009, 'A Virtual Design Platform', *27th eCAADe Conference Proceedings*, Istanbul, pp. 205-212.

Schneider, S Richter, N Petzold, F & König, R 2010, 'Open architectural design', *Proceedings of the 15th International Conference on Computer Aided Architectural Design Research in Asia*, Hong Kong, pp. 629-638.