Poiesis Prax
DOI 10.1007/s10202-006-0029-0

ORIGINAL PAPER

Is informatics a design discipline?

Peter Purgathofer

Abstract The article discusses the theory and practice of software development in the light of design theory. It tries to show that the design process cannot be forced into a predefined operational sequence. To underline this, the history of design methods is retraced, showing that such approaches were abolished not only in practice, but also the theory of design. The essay then discusses the cognitive framework of contemporary design theory and closes with the proposal that informatics should redefine itself as a design discipline in order to tackle the problems of interaction design.

Zusammenfassung Der Text diskutiert die Theorie und Praxis der Softwareentwicklung im Lichte aktueller Designtheorie. Es wird zu zeigen versucht, dass sich der Designprozess der operationalen Formalisierung entzieht. Dazu wird die Geschichte der Designmethoden nachgezeichnet, die zeigt, dass solche Ansätze nicht nur in der Praxis, sondern auch in der Theorie gescheitert abtrennung sind. Der Aufsatz diskutiert dann die kognitiven Rahmenbedingungen aktueller Designtheorien und schließt mit dem Vorschlag, dass sich die Informatik als Designdisziplin verstehen muss, um die Probleme des Interaktionsdesign wirklich lösen zu können.

Résumé Ce texte s'interroge sur la théorie et la pratique du développement de logiciels à la lumière de la théorie de design actuelle. Il tente de montrer que le processus de design se soustrait à la formalisation opérationnelle. L'histoire des méthodes conceptuelles est par ailleurs retracée, qui montre

P. Purgathofer (✉)
Institut für Gestaltungs- und Wirkungsforschung der TU Wien,
Favoritenstr. 9–11, 1040 Vienna, Austria
e-mail: purg@igw.tuwien.ac.at

que de telles démarches n'ont pas seulement échoué dans la pratique, mais aussi en théorie. L'article se penche sur les conditions cadres cognitives des théories de design d'aujourd'hui et s'achève sur la proposition selon laquelle l'informatique doit s'entendre comme discipline de design pour pouvoir véritablement résoudre les problèmes du design d'interaction.

Since informatics was diagnosed with the ''software crisis'' in 1968 (Naur and Randell 1969) and decided to form the field of software engineering, informatics has a strong focus on the process of software production. While it was agreed upon that there is no ''Silver Bullet'' process that would solve all problems, most of the processes defined since then failed to acknowledge the nature of design work. Subsequently, it can be argued that wrong conceptions over design and the design process bear a large part of the responsibility for bad software. Problems of interface design or interaction architecture are often handled following the structural conceptions of software engineering. As a consequence, substantial problems are not recognized, displaced or treated wrong. It has long been described and criticized that the source of this discord lies in the rationalistic tradition of software engineering (Floyd 1992).

The goal of this text is to argue against the attempt to force the design process into a predefined operational sequence based on methodological prescriptions. I will try to retrace the history of design theory in order to show that such approaches were all more or less abolished, because they could not offer any beneficial contribution to the practice of design. The essay will then discuss contemporary ''soft'' design methods and close with a proposal on the consequences of these theories for the design of interactive systems.

1 Design science

The scientific examination of design has its origin in the 1960s of the twentieth century. At that time, a series of conferences on the methodology of design led to the formation of the British Design Research Society and the American Design Methods Group (Buchanan 1992). Facing the increasing complexity of design work in the light of new materials and production methods, traditional procedures no longer seemed adequate. Trigger of this movement was the advance of system theory and operation research, induced and enabled by new developments in computer technology.

In the literature of the emerging design theory the basic procedure of designing was denoted as ''design by drawing'' (Jones 1970). Such a proceeding is very difficult to describe scientifically, mainly because no common structure or methodology is recognizable. In the developing design theory this was seen as a deficit. Jones explicates: ''The writings of design theorists imply that the traditional method of design-by-drawing is too simple for the growing complexity of the man-made world.'' (Jones 1970). Accordingly, the precursors of design theory strove to develop a methodical approach, establishing

what Cross (1984) retrospectively described as ''the design methods movement'' (Cross 1984).

2 Design methods

Gedenryd (1998) defines design method as follows: ''a normative scheme that specifies in detail a certain working procedure, the activities to perform, and also a specific order in which the activities should be carried out''. He also enumerates four common principles that distinguish all design methods that are described by his definition:

- the division of the design process into distinguishable phases, in which individual activities are accomplished isolated from the others,
- the specification of a logical sequence, in which these activities are to be accomplished,
- the necessity for planning all activities of a phase before their execution, and finally
- the symmetry between product and process , insofar as the partitioning of the process is guided by the components of the product.

These four principles build up on top of each other. In this sense the first principle, the division of the process into distinguishable phases, forms the foundation for the further principles. At the same time, each of the principles is an elaboration of the preceding and a necessary consequence.

The most fundamental of all design methods, which is also the common element of all of such processing concepts, is the division in analysis and synthesis. This model suggests that before you begin to develop a ''solution'', a detailed analysis of the ''problem'' has to take place.

The origin of splitting up the design process in phases is rooted in a deeply embodied occidental thinking tradition that sees analysis and synthesis as separate, sequential activities. As Gedenryd convincingly shows (Gedenryd 1998), this tradition has its roots in ancient Greece, especially the works of Pappus, and has established itself during the time of enlightenment where also our conception of science was constituted. The idea of the sequentiality of analysis and synthesis later became one of the core concepts in cognitive psychology and consequently in problem solving theory which is derived from cognitive psychology. Gedenryd (1998) writes: ''Pappus' text describes a method of analysis and synthesis [...]. The links between design methods, mathematical proofs, and Pappus' original account are remarkably strong, given the vast span of time between them.''

3 Christopher Alexander's "Notes"

One of the most consequent and influential contributions to the methodology of design is Christopher Alexander's ''Notes on the Synthesis of Form''

(Alexander 1964). This text excels in several regards: not only was it one of the first works in this area—Alexander gave a talk on the core concepts in 1962 at the first conference on design methods (Alexander 1963)—it is also extraordinarily well-formulated and makes no compromise. The ''Notes on the Synthesis of Form'' quickly became one of the most influential and most discussed works in design theory.

Alexander's states that the motivation for the ''Notes'' came from the realization that design is a task in itself, separated from production. He elaborates: ''[...] Because we are self-conscious, we need to make explicit maps of the problem's structure, and therefore need first to invent a conceptual framework for such maps. This is all I have tried to do.'' (Alexander 1964).

Above all, Alexander's theory criticizes the designer, whose personality influences the design process too much. He writes: ''The artist's self-conscious recognition of his individuality has a deep effect on the process of form-making. Each form is now seen as the work of a single man, and its success is his achievement only.'' According to Alexander, this furthers the ''cult of the individual'' (This is actually reflected in many areas of design education in Austria where we can find the ''Meisterklassen'' principle in art programs). Alexander describes this as a major mistake and proposes as an alternative: ''I shall really be trying to show that for every problem there is one decomposition which is especially proper to it, and that this is usually different from the one in the designer's head.'' (Alexander 1964).

In this we can also find one of the motives for the attempt to make the design process more scientific through the definition of more rigid, rational methods. The principles of science—repeatability, transparence and provability—are to become applicable by minimizing or removing the individual, subjective and ''irrational'' influence of the designer. For this it is necessary to represent this influence as one which should be rejected. Alexander does this by talking about the ''especially proper decomposition'' for a problem, indicating that it could be objectively correct, which also means that it is implicitly better than a designer could ever make it.

In Alexander's work, the influence of the subjective designer is replaced by a systematic model of the design process. It divides the design process into analysis and synthesis, whereby both phases can be executed (quasi-) automatically on the basis of hierarchical decomposition: ''This first phase of the process must of course be followed by the synthetic phase, in which a form is derived from the program.'' (Alexander 1964). About the first phase (analysis) he writes: ''The starting point of analysis is the requirement. The end product of analysis is a program, which is a tree of sets of requirements. [...] The program is made by decomposing a set of requirements into successively smaller subsets''. He describes synthesis as something that can be deducted from the analysis: ''The starting point of synthesis is the program. The end product of synthesis is the realization of the problem, which is a tree of diagrams. [...] The realization is made by making small diagrams and putting them together as the program directs, to get more and more complex diagrams.'' (Alexander 1964).

These quotations clearly show the mechanistic nature of his approach. With the use of phrases such as ''decomposing [...] successively'', ''as the program directs'', ''derived'', etc. he implies that the process can run without influence of the people involved. As already shown he claims that the structure of the problem—which is not subject of individual interpretation, but can be described objectively—should define the design: ''I shall really be trying to show that for every problem there is one decomposition which is especially proper to it.'' (Alexander 1964).

4 Reversal

Alexander's text stands out of other works in design theory from that time in many ways. What really distinguishes Alexander's work however is that it was publicly recalled by the author only a few years later. In ''The state of the art in design methods'' Christopher Alexander characterizes methodical approaches to design as useless: ''In short, my feeling about methodology is that there are certain mundane problems which it has solved—and I mean really incredible mundane.'' (Alexander 1971). In an interview he formulates his position even stronger: ''There is so little in what is called 'design methods' that has anything useful to say about how to design buildings that I never even read the literature anymore [...] I would say forget it, forget the whole thing.'' (Alexander 1971). Unfortunately Alexander's criticism was not taken as serious as his original work. Lawson retrospectively writes: ''Alexander's work has been heavily criticized, not least by himself, although few seemed to listen to him at the time!'' (Lawson 1980).

Alexander's insight is at least as amazing and visionary as his original work. While the ''Notes on the Synthesis of Form'' are still discussed in what remains of the ''design methods movement'', in practice it was almost completely ignored. Years later Lawson found out that there was only one attempt to use the method suggested in the ''Notes'': ''It is all the more remarkable since there is only one reported attempt to use the methods and that did not result in any obvious success.'' (Lawson 1980). By recalling his own work Christopher Alexander anticipates an insight that would take design theory years to realize. In 1981, Lawson is still careful as he writes: ''This chapter has examined the design process as a sequence of activities and found the idea rather unconvincing.'' (Lawson 1980). In 1998, Gedenryd can talk about the failure of design methods much more open: ''Experience from design practice and from studies of authentic design processes has consistently been that not only don't designers work as design methodology says they should, it is also a well established fact that to do design in the prescribed manner just doesn't work.'' (Gasparski 1990). He becomes even more direct a little later: ''Having said this much about design methods, there is but one thing to add: They don't work, and they don't work at all. In spite of all the good motives—the need for potent and up-to-date design procedures, the noble cause of being rational,

and so on—the failure of these methods is a very solid and widely recognized fact, as is the thoroughness of this failure.''

Christopher Alexander's radical break with design methods led to a splitting of design theory. On one side Alexander's critique and his ''forget the whole thing'' (see above) were ignored, and design methods were further developed, discussed and taught in a scholarly manner. Cross describes this development: ''Nevertheless, design methodology continued to develop strongly, especially in engineering and some branches of industrial design. (Although there may still have been very limited evidence of practical applications and results.)'' (Cross 2001). In the late 1980s this movement reached a bloom with the appearance of a number of standard works and the emergence of several respected journals.

The splitting also led to the development of another direction in design theory, which Gasparski later calls ''science of design'': ''The science of design (should be) understood, just like the science of science, as a federation of subdisciplines having design as the subject of their cognitive interests.'' (Gasparski 1990). If one believes Grant's observations from 1979, then the majority of the design theoreticians changed to this side: ''Most opinion among design methodologists and among designers holds that the act of designing itself is not and will not ever be a scientific activity; that is, that designing is itself a non-scientific or a-scientific activity.'' (Grant 1979). The most influential publications within this branch appeared partially only in recent years, so for example:

- Bryan Lawson's ''How Designer Think'' (Lawson 1980), originally from 1980, of which substantially enhanced editions came out in 1997 and 2006;
- The study of the working methods of twelve architects, ''Design in Mind'' (Lawson 1994) of the same author from the year 1994;
- Henrik Gedenryd's (1998) remarkable thesis ''how designers work'' from the year 1998, which proposes a fundamental reorientation of cognitive psychology for his design theory.

All of these works reject systematic design methods.

5 The allure of design methods

Even if design methods offer only very limited value in practice, they still enjoy a high reputation in scientific contexts. They are accessible for systematic treatment extraordinarily well; they afford excellently to being presented and discussed scientifically, and this finds a fertile soil in the usual publication channels. Additionally, due to their clear formulations, rules and guidelines are easy to explain and thus very fit for the use in educational settings. Because practical experience and/or talent have little or no value in design methods they seem to solve many problems of both design training and design practice. Design methods are well presentable and easily comprehen-

sible in text books with appropriate ''toy examples'' and can thereby be justified argumentatively, even if their value is not provable in practice.

For management, phase models of design also constitute an almost irresistible temptation. Buchanan writes: ''In the abstract, such a model may appear attractive because it suggests a methodological precision that is, in its key features, independent from the perspective of the individual designer.'' (Buchanan 1992). Thus phase models promise independence from the ''talented'' designer and thereby open an enormous rationalization potential. Design methods promise to remove the bottleneck of a rare and therefore expensive resource—the designer—and replace it with systematically examinable qualifications by the application of methodical procedure. Also they transfer more control over and insight into the process to the management.

6 "Soft" design theories

Compared with the high value design methods have gained in an academic context, ''soft'' theories of the design have an image problem; due to their nature it is very hard to present, examine and discuss them systematically. They do not compare with the historically grown and proven forms of scientific work. The publication of a design project conducted on grounds of a ''soft'' design theory—apparently unmethodical and unsystematically—is much harder, because no consistent (and above all comprehensible and repeatable) methodical guide can drive the treatise. In fact, the only relevant aspect of such a project is the designed product. The path that led there is subjective (for each project participant), unique (because unsystematically), uninteresting and/or hardly explicable (because not reproducible) and thus subordinated to the product in every aspect.[1]

In his ''Software design manifesto'', Mitch Kapor asks informatics to open to such approaches. In final consequence, he requires the introduction of a new discipline, oriented in its curriculum not along the principles of engineering, but along those of architecture: ''We need to create a professional discipline of software design. [...] Software designers should be trained more like architects than like computer scientists.'' (Kapor 1990). He recognizes the fact that the substantial difference would not result from choosing different content, but by using different educational methods: ''Software designers should be technically very well-grounded without being measured by their ability to write production-quality code. A large part of the training of designers is in studio work, in which students work on actual projects under the close guidance and supervision of practicing professionals.'' (Kapor 1990). In 1997 Terry Winograd (publisher of the anthology ''Bringing Design to Software'', which contains Mitch Kapor's above quoted articles) argues for

[1] Nevertheless, it would be very interesting to see more discussion of the design of interactive systems. Architecture has developed a culture of critique and discussion about their artefacts. It is high time that informatics followed this lead and started a broad debate about interactive artefacts produced by its agents.

this separation: As with a house or an office building, software is not just a device with which the user interacts; it is also the generator of a space in which the user lives. Interaction design is related to software engineering in the same way architecture is related to civil engineering.''

More than ten years after Kapor's manifesto such a community is still in its beginnings. Current HCI curricula still feed from computer science and psychology and demand integration into the scientific informatics education: ''The recent ACM report on the core of computer science [...] includes human–computer communication as one of nine subareas comprising computer science. [...]. In this report, we have adopted, as an ACM committee, an appropriate computer science point of view.'' (Hewett et al. 1992, 1996) The necessity to copy from the models and methods of design disciplines is still not understood. From a certain point of view this is not a miracle, not least because of the troubles of ''soft'' design theories in the traditional scientific context addressed above. A discipline that lacks a scientifically justified methodological and theoretical background raises the suspicion that it works on the basis of inspiration and talent. Thus it seems to be as improbable a role model for engineering disciplines as for science. It appears that Mitch Kapor's manifesto is still not broadly understood in HCI and therefore cannot be implemented sustainable.

Kapor largely avoids the question how the software development process would have to look like to include ''software designers''. He describes the necessity for a fundamental change of the process: ''We need to take a fresh look at the entire process of creating software—what I call the software design viewpoint: a rethinking of the fundamentals of the process of making software.'' (Kapor 1990). The most substantial change would be to take the control over the process from the hands of the engineers and put it into the hand of the designers: ''[...] in computer programs, the selection of the various components and elements of the application must be driven by an appreciation of the overall conditions of use and user needs through a process of intelligent and conscious design. How is this to be done? By software designers. [...] The software designer is a person whose job it is to design computer programs by taking overall responsibility for the conception and realization of the program.'' (Kapor 1990).

7 Design as interactive inquiry

The central paradigm of contemporary design research can be characterized through the phrase ''doing for the sake of knowing''. This phrase goes back to John Dewey who coined it around 1930 to illustrate his theory of cognition, and explained it in detail in 1938 (Dewey 1938). Donald Schön, building on Dewey's work, developed the model of ''reflection-in-action'' (Schön 1983). Armed with this concept, Schön deconstructs the rational approach of the design methods movement by showing that the designer as a ''reflective practitioner'' is shaping the situation through his reflective practice. The

knowledge of the designer is not only rational and cognitive, but also embodied in action and practice, and again understood by reflecting this practice.

In his most prominent setting, a teacher and a student constitute a hierarchical situation. Schön shows that every move of the teacher can be understood as an attempt to interactively frame the problem. The teacher generates ''moves'' towards a solution and reflects the outcomes of these moves through the practice of sketching. Henrik Gedenryd (1998) expands on Schön's theories and uses this example to show that the design process is always a process of inquiry, where the problem to be solved is not set in the beginning but is subject to change during the discourse. Gedenryd is able to lay out a concise theory of how the design process can be understood, centered around the concept of ''doing for the sake of knowing''. Consequently, he calls his text ''How designers work'' (Gedenryd 1998) in reference and contrast to the standard work in contemporary design theory, ''How Designers Think'' by Bryan Lawson (1980).

During his elaboration of design tools, Henrik Gedenryd (1998) mentions several techniques that are used in the process of design (e.g. sketches, thumbnails, scenarios, prototypes). Following Dewey's and Schön's lead, he explains the interaction with these artifacts, a unity of using and testing, as part of our thinking. A simple tool such as a pencil coupled with the specific ways of using it on paper in order to produce sketches then becomes a ''design instrument'' that affords specific ways of thinking about design situations. He shows that when in Schön's example the teacher ''reframes'' the students problem, what he really does is think using pen and paper, making it impossible to confine the process of thinking into the closed ''box'' that the brain is often seen as.

While there are more design instruments than sketching, they still share the same basic properties. In Buxton (2006), William Buxton describes the essential qualities of sketching. According to his list, sketches are quick, inexpensive, disposable and plentiful. A sketch follows a clear vocabulary that makes it recognizable as a sketch; it is open and free in gesture, constrained in resolution and has only a degree of refinement that does not exceed the necessary. And, by being ambiguous, sketches suggest and explore rather than confirm. It is important to keep in mind that this list, while long, describes the properties of something that allows us to think ''through'' it. Used as a design instrument, sketching (using whatever tools might be appropriate) enables us to be the ''reflective practitioner'' Schön (1983) has been talking about.

8 Design of Interactive Software

It is easy to see that there is an immense gulf between the theory of interactive inquiry and software engineering methodology. One direct consequence of the differences would be to radically rethink the concept of requirements engineering. If the problem cannot be set in advance, and if problem setting is an

important part of the design process, then the notion of a complete collection of requirements at the outset of the process is an illusion. It is not surprising then that the complaint about ever-changing requirements regularily tops the list of software engineering discomforts.

Of course, modern ''lightweight'' software engineering methods such as extreme programming (Wells 2003) acknowledge this problem and suggest a completely different approach. However, from the perspective of design theory the proposed solution is not adequate. Extreme programming makes coding the prime ''design instrument''. Unfortunately, coding is not a very good design instrument. From the above list of properties of sketching, programming lacks most. On the one hand, of course writing code can be quick and inexpensive and the products can be seen as disposable, and extreme programming does just that. Still, all the other qualities are missing. The influence of the inherent logic of programming necessarily locks out all forms of ambiguity, but ambiguity is important because it opens up new design spaces and widens the possibilities. Programming, caught in determinism, fails to do either. This characteristic of coding makes extreme programming, while still a better approach to the problems of software engineering than most deterministic process models, a poor method to design interactive systems.

From the position of design theory, it is most obvious that the design of an interactive system is a radically different ''beast'' than its technical implementation. Thus we should acknowledge that design decisions should be separated from the logic of implementation as thoroughly as possible. However, this does not necessarily mean that the process has to be divided into a design phase and an implementation phase. Much is to be learned from implementation, and many design problems popup only when a product is actually being built. It then is more a question of who should be in charge of the whole process. By putting designers in charge of the whole process, they also receive the authority needed to actually accept liability over the final product.

William Buxton charted out the ''ongoing responsibilities of various teams'' throughout the (software) product life cycle (Fig. 1). In a talk he emphasized a broad understanding of design: ''Of central importance for our purposes is the notion that 'design' includes the design of the business/marketing plans and engineering/manufacturing plans, as well as the product itself.'' (Buxton 2004).

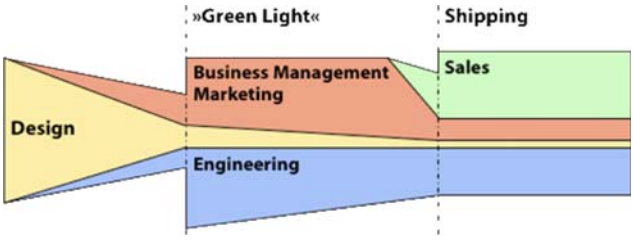

Fig. 1 Bill Buxton's (2004) graph of responsibilities in the product life cycle

In the context of software design, this means that the software will be designed from the ''outside'' to the ''inside''; the interaction design will determine the technologies and their application. The designers work will be structured along the lines of the design theory that is recounted in this article.[2] After a sustainable and sufficiently refined design has been ''green lighted'', implementation kicks in. Still, designers stay in control in order to answer all questions and solve all problems that are detected during implementation.

9 Conclusion

The emergence of ''Interaction Architects'' as proposed by Mitch Kapor will be only one step in the establishment of such processes. For such a process model to become reality, informatics has to redefine itself. Informatics has no intrinsic subject that exists outside of the discipline. Other sciences have nature or society to study, but informatics only deals (almost exclusively) with its own products. Thus, the discipline has to accept that everything it does in the end becomes part of human-built artifacts that change the life of people and the world. This, in my opinion, is the main characteristic of a design discipline.

This does not mean that programmers should be designers, and without doubt there are domains in the discipline that are more affected by this insight than others. Nevertheless, it obviously is a question of focus and setting of the right priorities, if we want the design of software in all aspects to be submitted to the context of its use and the needs of its users. All participants in the development process should have the same agenda, i.e., to make the user experience as effective, practical and enjoyable as possible.

It is this reorientation that bears the change for informatics to become what it long should have been: a design discipline. Only then can the vision of computer sciences ''beyond calculation''[3] come true.

References

Alexander C (1963) The determination of components for an Indian village. In: Jones CJ, Thornley D (eds) Conference on design methods. Pergamon Press, Oxford
Alexander C (1964) Notes on the synthesis of form. Harvard University Press, Cambridge
Alexander C (1971) The state of the art in design methods. In: DMG Newsletters 5(3)
Buchanan R (1992) Wicked problems in design thinking. In: Design issues, vol 8(2)
Buxton B (2006) What sketches (and prototypes) are and are not. Workshop at CHI06, 22–27

[2] Unfortunately, there is a huge lack in design instruments that fit the sketching of interaction. The Swedish interaction design researcher Jonas Löwgren and the Author of this article have started a website (Löwgren et al. 2006) to collect examples, best practices, research questions etc. on interaction sketching.

[3] ''Beyond Calculation'' is the title of a collection of essays edited by Peter Denning and Robert Metcalfe. One of the essays, Terry Winograds ''The Design of Interaction'' [23], coined the term ''Interaction Architect''.

April, Montreal, Canada. Position Paper at www.kid.rcast.u-tokyo.ac.jp/chi06-sketch-ws/final-position-papers/Buxton-SketchesPrototypes.pdf
Reproduction of a slide used by William Buxton in a talk (2004) The role of design in software product development, KMDI, Toronto, April 22
Cross N (1984) Developments in design methodology
Cross N (2001) Design/science/research: developing a discipline. Keynote on the 5th Asian design conference—international symposium on design science, 11–13 October, Seoul National University
Dewey J (1938) Logic: the theory of inquiry. H. Holt and Company, New York
Floyd C (1992) Software development as reality construction. Springer, Berlin Heidelberg New York
Gasparski WW (1990) On the general theory (praxeology) of design. In: design methods and theories, 24(2)
Gedenryd H (1998) How designers work—making sense of authentic cognitive activities. Ph.D. dissertation, Lund University Cognitive Science http://www.lucs.lu.se/People/Henrik.Gedenryd/HowdesignersWork/index.html
Grant D (1979) Design methodology and design methods. In: design methods and theories. J DMG, 13(1)
Hewett TT, Baecker R, Card SK, Carey T, Gasen JB, Mantei M, Perlman G, Strong G, Verplank W (1992, 1996) ACM SIGCHI curricula for human–computer interaction. Report of the ACM special interest group on computer–human interaction (SIGCHI) curriculum development Group. http://www.acm.org/sigchi/cdg
Jones JC (1970) Design methods. Van Nostrand Reinhold, New York
Kapor M (1990) A software design manifesto. In: Winograd T, with Bennett J, De Young L, Hartfield B (1996) Bringing design to software, Addison-Wesley, USA
Lawson B (1980[1], 1997[3]) How designer think. Butterworth Architecture, London
Lawson B (1994) Design in Mind. Elsevier, Amsterdam
Löwgren J, Purgathofer P (2006, ongoing) sketching for interaction, Weblog at twoday.tuwien.ac.at/interactionsketching
Naur P, Randell B (eds) (1969) Software engineering: report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7–11 October 1968, Brussels, Scientific Affairs Division, NATO
Schön D (1983) The reflective practitioner. Bibliography review 63, MIT Press, Cambridge
Wells D (2003) Extreme programming. http://www.extremeprogramming.org
Winograd T (1997) The design of interaction. In: Denning PJ, Metcalfe RM (eds) Beyond calculation: the next fifty years of computing, Copernicus Springer, New York