# On the Perception, Adoption and Implementation of Design Thinking in the IT Industry

**Tilmann Lindberg, Eva Köppen, Ingo Rauth, and Christoph Meinel***

**Abstract** In recent times, addressing the social aspects of IT products has become an important competitive factor on IT markets. IT development is forced to focus on more user-centeredness and the non-technical aspects of design problems. Against this background, design thinking has been discussed and applied as a new design paradigm for IT development. Basing on expert interviews and case study research, we examine in our research project what it means to put design thinking into operation in an IT context. We explain why design thinking is complementary to traditional IT design paradigms and what issues are involved in the subjects of perceiving, implementing and adopting design thinking in IT development.

## 1 Introduction: Design Thinking as a Complementary Design Paradigm for the IT Industry?

IT development processes call for highly trained professionals who are qualified to deal with complex technical issues, as programming languages or software and hardware architecture. Competencies in software engineering are not only important for taking part in programming, but also in designing the software.

As every decision about the design of an IT-system unavoidably manifests at the level of architecture or code, expert knowledge and thinking already play an important role in early design decisions. Consequently, the educational background of hardware and software engineers has a strong influence on mind-set building and

T. Lindberg • E. Köppen • I. Rauth • C. Meinel (✉)
Hasso Plattner Institute, University of Potsdam, P.O. Box 900460, 14440 Potsdam, Germany
e-mail: tilmann.lindberg@hpi.uni-potsdam.de; eva.koeppen@hpi.uni-potsdam.de; meinel@hpi.uni-potsdam.de

* Principal Investigator

H. Plattner et al. (eds.), *Design Thinking Research*, Understanding Innovation, DOI 10.1007/978-3-642-21643-5_13,

problem solving. As a result, IT development has the tendency to take place within an 'exclusive' experts' world (Lindberg and Meinel 2010).

In past and present times, these circumstances have led to the fact that technically and analytically trained IT engineers take on the designer's role as well, although they have not been professionally trained in that field. The word *software design* is indeed strongly associated with technical issues. This 'technical bias' in IT design leads to the tendency that the technical complexity of a design problem receives more attention than its social complexity. IT development has been struggling with the situation that products, functionalities and interfaces shape up as incomprehensible, inappropriate or simply unoriginal for the user, while other features or products considered as essential or meaningful from a user's point of view are not being addressed.

Overcoming this situation has become a key issue for the IT industry. As times went by in which the IT market grew mainly driven by technology push dynamics, the challenges IT development has faced exceed the established focus of an engineering experts' world and ask for the integration of further perspectives on problem framing and solution finding. Within the IT world, this problem has been tackled so far in two different ways. On the one hand, new design disciplines such as interaction or user experience design came up taking on specifically the role of the "user's advocate" within development teams (Buxton 2007; Mandel 1997; Vredenburg et al. 2002). On the other hand, new software engineering approaches, in particular those summarized under the umbrella term "agile development", put strong emphasis on an incremental and iterative development processes that is adaptive to user feedback throughout (Beck and Andres 2004; Pichler 2008).

The present debate on applying design thinking to the IT industry also addresses this problem, however with a primary focus on broadening generally cognitive problem solving patterns in IT development. Design thinking is associated with a problem solving style that supports the consideration of socially ambiguous aspects of a design problem. In contrast to orthodox engineering design paradigms, the corresponding problem solving patterns build upon heuristics and situational reasoning rather than on analytical thinking as focused in the IT engineering curricula. Design thinking differs from analytical thinking in diverse aspects:

- Design thinking relies on the development of concepts by using preliminary and even intuitive knowledge about a design problem, while proofs of concepts are adduced by the negotiation between different and probably conflicting stakeholder perspectives – i.e. the users, clients, manufacturers, law-makers (Dorst 2006; Lawson 2006; Owen 2006; Krippendorf 2006). Analytical thinking, in contrast, asks for definite knowledge to frame design problems into well-structured units before the actual problem solving process starts.
- In design thinking, problems are perceived as 'wicked' (Rittel 1972), saying that there is no definite formulation of a design problem at all. In this perspective, structuring a problem is rather seen as a process of taming instead of defining a problem. The relation between problem and solution therefore is not like deriving the latter from the former, as purely analytical approaches would

**Table 1** Analytical thinking vs. design thinking

| | Analytical thinking | Design thinking |
|---|---|---|
| | Ill-structured | Wicked |
| Problem perception | Well-structured | Tamed |
| Relation problem/ solution | Solution as a derivative consequence of a well-structured problem | Co-evolution of problem and solution |
| Key knowledge | Expert knowledge | Stakeholder knowledge |
| Key process | Defining and deriving | Iterations of observing and synthesizing, ideating and prototyping |
| Design paradigm | Rationalist problem solving | "Reflective conversation with the situation" |

suggest, but like framing problems and solutions *interdependently* in frequent iterations. This is what Cross calls the "co-evolution of problem and solution" (Cross 2007). Analytical thinking, however, pursues the analytical investigation of a problem setting by decomposing all its components or its determining factors and uses this knowledge to compose a design concept as a logical consequence.

- The key knowledge in design thinking is not the expertise of specialists but the knowledge of stakeholders that is supposed to be learned anew for every design process. The process behind design thinking thus builds rather on learning about problem and solution than on applying already learned knowledge, and therefore supports all activities of grasping multiple knowledge and multiple perspectives for inspiration as well as the creative transformation into new concepts (Beckmann and Barry 2007; Brown 2008; Lindberg et al. 2010; Owen 2006).

Seen together, analytical thinking and design thinking suggest different paradigms of designing (see also Table 1). Analytical thinking advocates applying scientific-rationalist problem solving to design problems and is particularly prevalent in areas with a strong focus on technical rationality – such as in IT engineering. In design thinking, however, solving design problems is regarded as a "reflective conversation with the situation" (Schön 1983) or as "reflection-in-action" (Dorst and Dijkhuis 1995) – a discursive and creative activity of developing design solutions in frequent communication with the stakeholders of a design problem. Therefore, design thinking focuses on those 'fuzzy' aspects of a design problem, which are in purely engineering-led approaches left aside, and is thus suggested as a useful supplement to problem perception and solving in 'traditional' IT development approaches.

In consequence, problem solving approaches in design as well as in IT engineering complement each other in principle. Whereas design thinking allows dealing with the ambiguity of design problems as wicked problems, the thinking of IT engineers instead supports the effective technical realization. The inherent difficulties to communicate between experts and non-experts during the IT development process make the complementary use of both approaches complicated and lead to a dominance of analytic-systematic approaches to problem and solution finding. Against this

background, design thinking can take the role of a meta-disciplinary rational, which allows a team across the disciplines to develop a mutual and general understanding of problem and solution, as it broadens disciplinary reasoning and helps, for example engineers, to forget about the patterns for a moment that they have internalized in their academic training – until a problem has been defined precisely enough so that professional rationales and expert knowledge may suitably be applied.

## 2 Research and Methodology: Analyzing Language Games on Design Thinking in the IT Industry

Beyond purely conceptual thought on applying design thinking to IT development, important questions remain open. How can design thinking be conceptualized and distinguished within software engineering on a practical level? How is it understood and adopted to daily work routines? How can it be imparted and organizationally implemented? Finding answers to those questions is still a challenging endeavor, as there is a lack of both conceptual models and hands-on experience. In our research project "Design Thinking in the Development Processes of the IT Industry" we set up initial research to explore those questions.

We conducted expert interviews mainly with IT developers that have been trained in design thinking workshops as well as with trainers and observers of those workshops. The basic insights gained from these interviews showed the topic's complexity quite clearly. Albeit the majority of interviewees regarded design thinking by some means as enriching, we discern partly different views on what design thinking is and how it can be adopted and implemented. We hypothesize that this variety of perspectives affects not only implementation and adoption, but shows also a paradoxical trait of design thinking itself, namely that it is neither perceived as an insubstantial 'buzz word', nor as a delimited concept. Between both extremes, its meaning seems to be strongly subjected to vivid 'language games'.

'Language games' is a concept developed by the philosopher Ludwig Wittgenstein (1984) that explains how one word can carry an infinite series of meanings – depending on the context or situation in which a word is used. Wittgenstein puts language in analogy to games, because each game represents a certain set of backgrounds, goals and rules. Some are constitutive; others are rather implicit and can be modified. Wittgenstein applies this as metaphor to language, stating that rather the language games than the words decide whether communications work: when people play with the same words but according to different rules, there would be a great juxtaposition of language games hindering each other to succeed. Thus we regard it as important to identify and distinguish 'language games' on design thinking in order to create a basic understanding about how to apply design thinking to the IT industry. To do so, we employ a systematic approach to qualitative text analysis based on 'grounded theory'. In what follows, we will give a short overview about our research setting and our method of investigation.

We conducted 30 expert interviews (Bogner et al. 2005) with three groups of people working in the IT industry in Germany and the US: first, design thinking experts that educate IT engineers in design thinking; second, IT engineers that have been trained in design thinking; and third, experts from specialized design disciplines like user experience design that observe these efforts.

All interviewees were involved with design thinking in the form of a particular didactic workshop model either as trainer, participant or observer. Those workshops followed an approach popularized by the design agency IDEO as well as the Stanford d.School model (Plattner et al. 2009), in which small, multidisciplinary teams (generally without a professional design background) tackle a seemingly simple design challenge (for instance: how to improve ticket machines for public transport) and are supposed to understand how far one's own imagination of a viable solution changes after learning about the stakeholder perspectives (in particular the users'). To do so, those workshops suggest a prescriptive process model guiding the team through a design workflow in which first the team learns about the problem, then synthesizes the gained information to a framework of knowledge, using this framework as inspiration for ideation, and develops, prototypes and refines those ideas iteratively by means of frequent user feedbacks.

We developed interview guidelines with slight variation between the first and the other groups. Those guidelines contain three groups of questions: first, questions about the interviewee and his department's role in the company; second, questions about his view on design thinking; and third, questions about his opinion on how to implement and adopt design thinking to IT companies. Each interview lasted approximately 1 h, was recorded and later on literally transcribed.

We used grounded theory as our methodological framework for data analysis (Strauss 1998). Grounded theory is an approach developed in social science for empirically substantiated theory generation and is particularly useful when it is required to frame a fuzzy empirical setting. Condensing empirical data in frequent iterations and comparisons in order to develop coding schemes is the main driver of theory generation. We pursued an 'axial coding' approach, as we presupposed 'language games' as core category for the data analysis process. We used the software MAXQDA to support the data analysis process (Kuckartz 2007). We synthesized our coding schemes to three hypotheses, which are depicted in the next chapter.

## 3 Results: On the Perception, Adoption and Implementation of Design Thinking in the IT Industry

We looked at three issues in our analysis: first, how is design thinking understood as such; second, how is it understood in respect of IT development; and third, how is it discussed within the scope of the implementation to organizational structures and the adoption to personal working routines. Dealing with these questions, we developed three hypotheses, which will be expounded below.

**Hypothesis 1.** The understanding of design thinking is more aligned when it comes to describing its general goals and principles; differences however increase when it comes to describing design thinking on a more applied level.

This hypothesis is related to the question of how design thinking is understood as such. We found out that there are no contradictory differences of opinion on what design thinking generally is, albeit the ways to express this vary. One interviewee stating that design thinking is *"willingness to ask (. . .) 'am I really solving the right problem'; and then to try out what the right ways are to solving this problem"*, stresses another aspect than an interviewee stating design thinking is "*that the usability of the product and acceptance of the end user determines the design of a product*".[1] Another quote combines both messages: "*design thinking is a way to get out of your narrow view of what your problem is and (. . .) look broader and take everything from your environment in a view that kind of helps; (. . .) solve the problem that you need to solve, so most of the time it's going out and talking to users and talking to customers and anybody who is (. . .) associated with that problem*." Generally spoken, the interviewees emphasize either one or both of the following aspects, namely (a) finding the viable solution to the fairly understood problem, and (b) both the viable solution and the fairly understood problem are delimited by the user's point of view. Both aspects are deeply complementary, so that we do not see any confusion of language games when it comes to a general explanation of design thinking.

However, when it comes to applied explanations of design thinking, we discerned two divergent views. On the one hand, design thinking is explained as a methodology with a strong focus on a prescriptive process model, supportive tools and an underlying team structure. This can be exemplified with one interviewee distinguishing three levels of information about design thinking: first, "*specific tools and techniques, which are things like how to run a brainstorming workshop or how to do user interviews or the very specific tangible activities and tools that you do*"; second, *"the group dynamic piece" of (. . .) teams working on problems (. . .) and how do you get them to (. . .) come up with new ideas*"; and third, "*the overarching categories*" (he uses 'categories' instead of 'phases' as he wants to avoid the image of sequential process). On the other hand, design thinking is seen rather as a mind-set from which people draw their actions without relying on instructions from a formalized method. One quote shows this transition quite clearly: *"On the one hand, (. . .) (design thinking) is a method that I associate mostly with the whole process and its phases; and on the other hand it is a sort of mindset. (. . .) And I think you don't have to go through the whole process when you have this attitude (. . .). You just should have the intention in mind and try to live it."* We see in both views fundamentally different qualities. The first view regards design thinking as a bundle of methods that can be realized by means of organizational arrangement; the second regards it as a way of thinking that has to be

[1]German quotes are translated to English by the authors.

internalized by means of education. Thus we assume that this causes a juxtaposition of language games that can make it difficult to agree on the concrete purpose of design thinking in a company: Is it a meta-disciplinary attitude that people should learn, or is it an organizational technique that people should stick to?

**Hypothesis 2.** Design thinking is understood as a learning approach contributing to IT development rather than a development approach in itself.

Our second hypothesis is connected with the question how far the understanding of design thinking is related to IT development. This is a central aspect as it entails, whether design thinking *competes* conceptually with existing IT development techniques, or if it is regarded as contribution to those techniques. We found quite a clear picture. None of our interviewees sees design thinking as a clear-cut alternative to existing software development approaches, independent of the approval of agile approaches as SCRUM or sequential approaches such as the waterfall model.[2] Instead, the general focus is on the learning aspect of design thinking regardless of what development approach is in favor. The following quotes exemplify this: *"(Design Thinking) is imagining, understanding a problem space, and eventually the search for solutions, whereas one lets things drift at times, without any restrictions imposed upon oneself, but open for all possible kinds of ideas, then however making very quick steps to find out what is viable and what is not."* This interviewee, a software developer, emphasizes the value of design thinking in fast-track (and thus inexpensive) learning about problem space and potential solution paths outside the prearranged restrictions (that IT development altogether would entail). Another interviewee points out the difference between design thinking as a learning approach and developing itself: *"Design Thinking does not guarantee an outcome. That is completely in conflict with the idea of working with uncertainty. So, understand that forming fast, lean, simple, even with prototypes that are reflective of the end state, you are not moving forward to but make you smarter about how the end state should be. That in itself is a tool. It is not an alpha release (. . .). It's just a thinking tool to understand the problem."* We see that design thinking is regarded as a contribution to a certain notion on software development in general, namely 'how to build up a novel and viable design' – whereas the notion 'how to build up a functioning IT system in time and budget', which every IT development process has to tackle as well, does not play any role. This however suggests that the first notion has not been effectively addressed in IT engineering as otherwise design thinking would have been perceived rather redundant than contributing. As one lead IT developer stated about design and development in general: *"There is no specific statement on design in the building process, so that the build process doesn't say anything about design. But it is up to the individuals who implement the build process and then apply their*

[2]Albeit the majority of our interviewees showed preferences for agile development approaches, sequential approaches are favored when it comes to large-scale IT development projects due to its better planning reliability.

*design thinking based on their understanding."* This statement exemplifies the inherent 'technical bias' in IT development (see chapter "Tele-Board: Follow the Traces of Your Design Process History"). It indicates that the process of building software is more constituted than the process of designing software, so that the question how to build a viable design is likely to be subordinated to how to build a functioning system. The knowledge gap that this imbalance creates seems to be the reason why design thinking attracts developers.

**Hypothesis 3.** There are two groups of language games when it comes to implementing or adopting design thinking to IT development: the firsts treats design thinking and IT development as two separate worlds, and the second as an integrated one.

Our third hypothesis relates to both questions: how people speak about the organizational implementation and how they speak about the personal adoption of design thinking to IT development. We found equally two underlying language game patterns, namely that design thinking is treated (a) as an external, self-contained matter linked to but not into integrated to IT development, and (b) as an influence to change IT development itself. We call these patterns the 'two-worlds games' and the 'one-world games' respectively.

When it comes to implementaion, the 'two-worlds games' manifests in the idea that design thinking is realized in a project prior to the actual development process. Our interviewees describe this either as a service by an external 'task force', and/or as a form of workshop in which also some developers contribute substantially so that they can act as "design advocates" in the development process later on. However, the crucial transition between both worlds is generally a prototype as the outcome of the design thinking process that ought to serve as a starting point for the development process – and thus gets *"thrown over the fence"*, as two interviewees say. Against that background, discussions on adopting design thinking lead to controversies on how design thinking prototypes can be picked up by the developers later on. One interviewee sees different conceptions of prototypes as a hurdle between both worlds. He stresses that design thinking prototypes eventually embody completed concepts *"to which you can get down afterwards asking: how can we translate this to a real product?"*, whereas prototypes in IT development instead initiate a process of conceptualizing: "*You build software prototypes because you have otherwise nothing to look at when you discuss what you actually need – which would be extremely difficult. (. . .) It is easier to define requirements in the software world as a delta to something existing*." This discussion shows that there is a danger of misconception at the transition between both worlds. Developers are more used to treating prototypes as a form of tangible assistance for the development process and not as a non-technical blueprint for the final product. Connecting both worlds is regarded as critical as the following statement exemplifies: *"(. . .) design thinking people should learn that what they deliver is not enough for that what developers need. On the other hand, also developers should learn that sticking post-its casually and creatively, filling them with writings and permanently rearranging them is also serious and valuable work that provides results that the developers need afterwards."*

We found many perspectives on how to merge both worlds to a 'one-world game'. The most general of these views treats design thinking as an *imperative*, or rather as a kind of 'wake-up call' for developers to alter the way they work. Implementation happens in this sense through people who take this up and change their mindsets and problem solving routines. As one of our interviewees states: *"It is cultural change. The people just have to learn to change their views."* This is rather a symbolic approach of implementing, as it is about demonstrating the benefits of design thinking-led problem solving and asking people to internalize and to apply it. Yet, as one interviewee stated about the workshop experience: *"Many were enthusiastic about it. Many said: 'I want to adopt it somehow, I just do not know how,' or: 'how can I tell my boss?"*

As this quote exemplifies, we were able to identify a strong tendency that when design thinking is communicated as an appeal to developers, they appreciate the general idea but doubt being able to apply it within the tight frames of a development organization. Moreover, we found out that there is severe risk perception involved. Many had problems aligning the openness and the 'explorative detours' in design thinking with common performance measurement systems for IT development projects that rely on project plans, milestones and punctual shipment: *"When you are under time pressure and have to finish your tasks, then you refer to what you are assessed by and what you have to fulfill. Those things where everybody would say, 'yes, that would make sense' are skipped anyway."* Against the background of those organizational practices, design thinking is perceived as an uncertain method, which may be helpful to come up with innovations, but also entails a high planning (and justification) risk. The willingness to apply design thinking in daily work therefore ends when superiors ask for results without explicitly backing the use of a design thinking approach.

We found two views that try to overcome this dilemma. Some suggest to implement design thinking in the form of an obligatory phase of development processes, others intend to translate design thinking to an adaptive toolbox that can be applied by developers depended on what kind of problem they are faced with (instead of in which phases they are). The first would treat the learning effect of design thinking itself as an objective that has to be achieved; the second would make the use of design thinking more flexible so that requests of using design thinking methodology could be formulated very specifically. However, within the frame of our study, both ways of implementing design thinking were not realized so that we could not gain further insights about them.

## 4 Conclusion

In summary, this discussion confirms the observation that the range of perceptions of design thinking diverge as more applied people think and speak about design thinking. Table 2 summarizes the variety of views on how to implement and adopt design thinking in the IT industry.

**Table 2** Language games on implementing and adopting design thinking

| | Implementing | Adopting |
|---|---|---|
| Two-worlds games | DT as a foregoing project; DT as a service | 'Picking up what is thrown over the fence' |
| One-world games | DT as a 'wake-up call'; DT as a process phase; DT as an adaptive toolbox | 'DT is appealing, but needs backing by the organization'; 'Choosing DT tools when it helps' |

Drawing on the image of 'languages games', we showed that there is no single way of how meaning is created about design thinking in IT development, but rather an evolving variety of ways. This was substantiated in the hypotheses 1 and 2 pertaining to the range of understandings of design thinking in general and in IT development in particular, as well as in hypothesis 3 as for matters of implementing and adopting. We regard an initial incongruity between design thinking and IT development as a basic cause for this juxtaposition of language games. Design thinking is not a concept that seamlessly infixes as a further development approach to the IT world. Instead, it is a self-contained methodological field that can serve as an example to tackle shortcomings of established IT development approaches, i.e. the technical bias, by suggesting further attitudes towards knowledge and categories of knowledge for IT development. As a result, it remains fuzzy what exactly the overlaps between design thinking and IT development are like. Applying design thinking to IT development thus presupposes strong *translation efforts* that set off – as shown in our study – the emergence of divergent and partly incongruent language games.

We regard the resulting juxtaposition of language games as both helpful and destructive. It is helpful when it stimulates reflection and awareness of the constraints and limitations of established IT design approaches. It can be destructive when it comes to implementing and adopting design thinking, as there is a danger that parallel meanings weaken the communicability of the concept, and dissolve it in the end within a 'semantic nirvana'. As implementation and adoption require clear-cut concepts, it is not surprising that our interviewees tried to wipe out the fuzzy overlaps between both worlds either by separating them into two distinct worlds, or by merging them into one integrated world. With both ways, our interviewees sought to bring clarity to the rules determining the language games on design thinking in IT development. This shows quite plainly that promising attempts to implement or adopt design thinking presuppose clear images of how design thinking can be thought within IT development. On the basis of our study, we can distinguish different models connecting design thinking and IT development processes:

– In the **split project model**, design thinking is handled as a separate process performed by a specialized "design thinking team" before the IT-development process starts (→ design thinking as a service). Its main purpose is to map out potential directions in terms of user needs and to inform the IT development process with an initial "package" that is handed over to the subsequent development process.

- In the **overlapping teams model**, an initial design process is likewise used to inform the subsequent development process. But instead of "throwing the package over the fence", one or more project members of the development team participate in the design thinking process to be able to act as an communication agent to explain and maintain the gained design knowledge throughout the development process.
- In the **unified project model**, design thinking is a central technique for the front-end of the development process itself. The overall process is changing from a design thinking to an IT development process when the conceptions of problem und solution are specified enough to translate them to development tasks. This implies that there is a strong overlap of personnel and management responsibilities between both the design thinking and the development phases.
- In the **toolbox model**, design thinking is not regarded as a distinct project or process phase, but as a bundle of methods developers can draw on to solve certain design problems they could not solve by means of common IT development methods. In this case, design thinking is narrowed down to a well-defined box of tools for adaptive support.

However, also this range of models carries some inherent contradictions due to the fact that the respective implementation strategies imply different conceptualizations of design thinking itself. A design thinking toolbox, for instance, focuses rather on handy and selective techniques, while a split project focuses also deliberately on a coherent design thinking process carried out by skilled personnel. This observation is supported by our first hypothesis, stating that design thinking is more apparent as a general concept than as an applied one. Implementing design thinking seems to be thus strongly connected to a conceptualizing process – and a wide range of potential design thinking implementation models seems to be an inevitable consequence. To further explore and to develop the range of models in greater detail will be an important task for both future research and management practice.

## References

Beck, K.; Andres, C. (2004) Extreme Programming Explained – Embrace Change, Addison Wesley-Longman, Amsterdam.

Beckmann, S. L. and Barry, M. (2007) Innovation as a Learning Processes - Embedded Design Thinking, *Californian Management Review*, 50(1), 25–56.

Bogner, A.; Littig, B. and Menz, W. (eds.) (2005) *Das Experteninterview – Theorie, Methode, Anwendung*, VS-Verlag für Sozialwissenschaften, Wiesbaden.

Brown, T. (2008) Design Thinking, *Harvard Business Review*, June, 84-92.

Buxton, B. (2007) *Sketching User Experiences: Getting the Design Right and the Right Design*, Morgan Kaufmann, San Francisco, CA.

Cross, N. (2007) Designerly Ways of Knowing, Birkhäuser, Basel etc.

Dorst, K. (2006) Design Problems and Design Paradoxes, *Design Studies*, 22(3), 4–17.

Dorst, K.; Dijkhuis, J. (1995) Comparing paradigms for describing design activity, Design Studies 16, 261–274.

Krippendorf, K. (2006) *The Semantic Turn – a New Foundation for Design*. Taylor & Francis, Boca Raton etc.

Kuckartz, U. (2007) *Einführung in die computergestützte Analyse qualitativer Daten*, VS-Verlag für Sozialwissenschaften, Wiesbaden.

Lawson, B. (2006) *How Designers Think – the Design Process Demystified*, Architectural Press, Oxford.

Lindberg, T; Meinel, C. (2010) Design Thinking in the IT Industry?, Electronic Colloquium on Design Thinking Research, Report No. 1.

Lindberg, T.; Noweski, C. and Meinel, C. (2010) Evolving Discourses on Design Thinking – How Design Congnition Inspires Metadisciplinary Creative Collaboration', *Technoethic Arts*, 8(1), 31–37.

Mandel, T. (1997) *Elements of User Interface Design*, John Wiley & Sons, Hoboken etc.

Owen, C. (2006) Design Thinking – Notes On Its Nature And Use, Design Research Quarterly, 1:2, December, 16-27.

Plattner, H.; Meinel, C. and Weinberg, U. (2009) *Design Thinking*, mv-verlag, Munich 2009.

Pichler, R. (2008) Scrum – Agiles Projektmanagement erfolgreich einsetzen, dpunkt.verlag, Heidelberg.

Rittel, H.W. J. (1972) On the Planning Crisis - Systems Analysis of the First and Second Generation. Bedrifsokonomen No. 8, 390–396.

Schön, D.A. (1983) The Reflective Practitioner – How Professionals Think in Action. Perseus Books, New York.

Strauss, A.L. (1998) *Grundlagen qualitativer Sozialforschung*, Wilhelm Fink Verlag, Munich.

Vredenburg, K.; Isensee, S. and Righi, C. (2002) *User-Centered Design – An Integrated Approach*, Prentice Hall, Upper Saddle River.

Wittgenstein L. (1984) *Tractatus logico-philosophicus*, Suhrkamp, Frankfurt a.M.