Design Thinking: A Fruitful Concept for IT Development?

Tilmann Lindberg, Christoph Meinel*, and Ralf Wagner

Abstract In our research project *Collaborative Creativity of Development Processes in the IT Industry*, we pursue the question how design thinking can help to enhance the innovativeness in IT development and which individual and organizational factors facilitate or encourage this. In this chapter, we outline what the contribution of design thinking to engineering thinking can be, how it is related to akin IT development approaches (e.g. agile development), and what our initial insights on the didactic and organizational implications are.

1 Introduction: On Problem Solving in Design and Science

> "Most outsiders see design as an applied art, as having to do with aesthetics, unlike a solid profession unto itself, with technical knowledge, skills, and responsibilities to rely on. Insiders to design, by contrast, talk of innovative ideas, coordinating the concerns of many disciplines, being advocates for users, and trying to balance social, political, cultural, and ecological considerations."
>
> Klaus Krippendorf (2006, 47)

It seems to be the nature of design that it is not always easy to understand what this term actually means. As the initial quotation exposes, people in design themselves have not only a broader view on their discipline, they also see themselves in a different role than outsiders do: not merely in the position of dealing with aesthetical aspects of forms and products, but of taking on a general role of coordinating disciplines, stakeholders, and the manifold environmental matters within product development processes. The following argumentation reduces this point to central differences in problem solving in design as well as in science and technology.

T. Lindberg (✉), C. Meinel, and R. Wagner
Hasso-Plattner-Institut, PO-Box 900460, 14440 Potsdam, Germany
e-mail: tilmann.lindberg@hpi.uni-potsdam.de; meinel@hpi.uni-potsdam.de

* Principal Investigator

H. Plattner et al. (eds.), *Design Thinking: Understand – Improve – Apply*,
Understanding Innovation, DOI 10.1007/978-3-642-13757-0_1,

Comparing problems tackled in well-established sciences as astronomy, quantum mechanics, or computer science with those in design, it seems obvious that design problems are closer to the everyday life. What does a backache reducing office chair look like? What form should a computer interface have to be accessible for elderly people? What do we have to do to avoid injuries through battery acid in developing countries? The motivation to find answers to those questions is not to gain scientific knowledge or to discover new technical possibilities (even if design clearly takes advantage of both). Rather, it is the need to create ideas and find solutions (products, services, systems), which are as viable as possible for certain groups of users. By that, design intends to offer a very concrete solution to a complex problem that is socially highly ambiguous and hence neither easy nor certain to comprehend. Design problems thus – using a term by Horst Rittel – are close to *wicked problems*, blurred in character and not definitely definable (Rittel 1972).

People in design have developed problem solving skills that allow them to deal with such kind of problems successfully. This however calls for a dissimilar mode of thinking than taught in the curricula of the established sciences. Solving wicked problems does not acquire the analytical inductive/deductive scheme pursued in science that follows an epistemological logic to achieve knowledge about scientific truth, since designers only strive for enhanced viability and novelty of products (Dewey 1997, 79; Martin 2009, 63). Scientific problems generally will be answered by theories, concepts, taxonomies, or models, and only become finally accepted when they depict a problem analytically in all its dimensions. Still, this demands not only long-term efforts to put into research and analysis but also to reduce the complexity of a stated problem to such an extent that it is finally *non-wicked*, thus entirely describable. Yet for design problems, this would be a misleading approach: first, as designers have to deal with problems pragmatically and come up with solutions in much shorter periods of time than scientists do; second, as designers do not have the possibility to reduce the complexity of a problem because design problems are made up of exogenous perspectives that finally decide about the solution's viability: the user's, the client's, the engineer's, the manufacturer's, the law-maker's, the environmentalist's, the employee's, etc. (Lawson 2006, 83f).

It becomes obvious that design unavoidably has to take on a coordinating role within this context of multiple stakes, because they have to rely on the knowledge of others as long as they want to approach general viability of solutions. To do so, there emerged professional learning strategies in design: cognitive patterns to grasp multiple knowledge and multiple perspectives of others for the purpose of synthesizing and creatively transforming the knowledge to new service or product concepts. In contrast to analytical thinking in science, we call those strategies *design thinking* (Brown 2008; Dunne and Martin 2006; Lindberg et al. 2009).

To understand how those strategies work, it is useful to work with two fundamental pairs of terms: the problem and solution space on the one hand, and diverging/converging thinking on the other hand.

The distinction between the problem and solution space elucidates the dualistic approach of design thinking.[1] Whereas in science the focus lies in general on exploring the solution while the initial problem is given, design treats both the problem and the solution as something to be explored. This indeed characterizes design thinking as an approach for professional learning. The distinction between diverging and converging thinking however shows how designers approach both spaces (Lawson 2006, 142f). Learning alone, certainly, is not enough in design as the knowledge acquired is at the same time a means to come up with viable as well as novel solutions. Thus design thinking is always an interplay between diverging exploration of problem and solution space and converging processes of synthesizing and selecting. Contrary to thinking styles predominant in science, the knowledge processed in design thinking has to be neither representative (as in inductive thinking) nor entirely rationalized (as in deductive thinking), rather it serves to obtain an exemplary but multi-perspective comprehension in order to deal creatively with the ambiguity of wicked problems. Building upon this argumentation, design thinking can be put down to three basic characteristics (see also Fig. 1).

- *Exploring the problem space*: When exploring a problem space, design thinking acquires an intuitive (not fully verbalized) understanding, mainly by observing exemplary use cases or scenarios, as opposed to formulating general hypotheses or theories regarding the problem; and synthesise this knowledge to point of views.
- *Exploring the solution space*: Design thinking asks for a great number of alternative ideas in parallel and elaborates them with sketching and prototyping

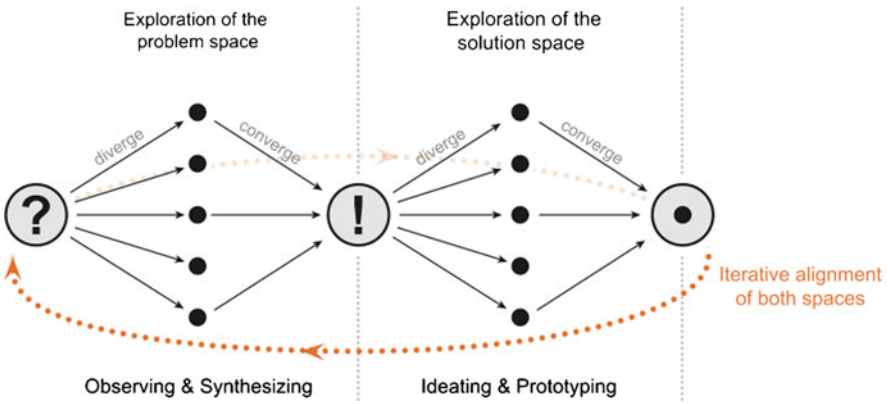

Fig. 1 Problem and solution space in design thinking

[1] A concept of the "problem space" was first introduced by Newell et al. (1967), although with a different meaning, as they locate the representation of possible solutions in the problem space without regarding a separate solution space. This conceptualization does not appear sufficient for our purpose, as the distinction between learning about the problem and learning about the solution in design thinking could not be depicted.

techniques. In this manner, ideas are being consciously transformed into tangible representatives.
- *Iterative alignment of both spaces*: These representatives of ideas and concepts facilitate communication not only in the design team, but with users, clients and experts as well. Thus, design thinking helps to keep in touch with the problem-relevant environment and can use this information for refining and revising the chosen solution path(s).

Accordingly, design thinking engenders a system of checks and balances to ensure that the conclusive solution will be both innovative and suitable for the social system that the design problem addresses.

2 Understanding the Problem: Overcoming the Dilemma of Analytical Thinking in IT Development by Design Thinking?

As outlined in the preceding section, there are crucial differences concerning the quality how problems are characterized and approached in design and in science. The paradigms of problem solving in science originated in epistemology, the studies of finding out what is true or not, and led to a strong focus on analytical thinking. The paradigms of problem solving in design though originate in finding out what novel solution fits best in a social or technical system. The discussion about how to bring both paradigms together is particularly valuable in areas with a strong tradition in analytical thinking but which are nevertheless related to design responsibilities.

In our research project *Collaborative Creativity of the Development Processes in the IT Industry*, we explore the capabilities of design thinking to broaden the problem understanding and problem solving capabilities in IT development processes. We pursue in particular the question in which cases the application of design thinking to IT development is valuable (or not) and what the individual and organizational implications for teams and organizations are. Hereby, we draw upon an in-depth case study research with international IT companies. In our first year, we conducted 36 qualitative guideline interviews with IT experts on the one hand and design thinking experts on the other who were involved in design thinking-based IT projects both in the United States and in Germany.

In the following section, we will discuss the background and the peculiarity of our research focus.

2.1 Why IT Development Tends to Take Place in an Engineering Expert's World

Within the IT industry – comparable to any other industry dealing with technical products and services – a technical perspective plays a central role in problem solving and solution development.

This seems to be necessary as the development process itself asks for highly trained professionals who are able to deal with complex technical issues, such as programming languages or software and hardware architecture. Competencies in engineering-centered areas used to be not only a condition for participating in the IT development process, but also in the developing process of the actual design of the software product itself. Getting a comprehensive understanding of what the product will look like, what solutions will work or not, and how the conditions of interaction between user and software can be shaped, generally presupposes the ability to communicate about those questions in technical terms. This is due to the situation that every decision about the software design unavoidably manifests at the level of architecture or code and, thus, cannot be solved without expert knowledge. Consequently, the educational background of hardware and software engineers has strong influence on mind-set building and decision-making and, as a result, IT development has the tendency to take place within an "exclusive" expert's world. Thus, in past and present times, these circumstances lead 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, in fact, one of the few design terms that are almost exclusively associated with technical issues.

2.2 The Dilemma of a Predominantly Technical Perspective

Pursuing a dominant technical perspective in IT development however comes with its own set of problems. One basic problem, for instance, is that functionalities and user interfaces, albeit technically perfect, may shape up as incomprehensible or inappropriate from the user's point of view. Other features considered as meaningful and essential from a user's perspective may not be addressed. This problem, which is in these days e.g. approached by the research field of human-computer-interaction, can cause serious drawbacks: inefficiency and loss of effectiveness for the user, rejection of the product, and a loss of innovation prospects for the producer. Furthermore, those times in which the IT market grew mainly driven by technology push dynamics became a thing of the past. From home entertainment to web 2.0 applications and ubiquitous computing, IT products deeply depend on social life dynamics, which are primarily not the concern of engineering but of design. Since IT solutions became more and more part of people's everyday life, not only the demands on usefulness and usability have been growing continuously, but IT engineers

must also learn to develop for highly competitive consumer markets, in which successful innovations are rather defined by the users point of view than by technical perfection. An isolated technical perspective entailing isolated analytical thinking can thus lead into an innovation trap: while spending much effort in the development of technically novel or reasonable solutions, the clients do not really see the solution's distinctive value.

2.3 Design Thinking as a Complementary Approach?

Overall, the challenges that IT development is faced with exceed the established focus of an expert's world and ask for the integration of further perspectives on problem understanding and solution finding in IT development. This problem actually is the focal point of the present debate of applying design thinking to IT. The thinking of IT engineers is mainly influenced by the deductive-rationalist approach as taught in mathematics and informatics – subsequent to the logic of having a given problem and deducing the right solution in accordance with rational rules of logic. Design thinking, by contrast, teaches to treat problems as wicked problems, thus more openly, with the purpose of embracing the blurred space of social ambiguity through which a successful design process should pass as well. Thus, the idea behind applying design thinking to IT development asks for setting up a complementary thinking style, which extends the problem solving abilities of IT development teams with the purpose of making their outcomes more innovative.

However, when we talk about design thinking, we do not use the term in a sense of how designers (may) think, but of how anyone "should" think while dealing with design problems (Lindberg et al. 2009).

In consequence, we do not look upon design thinking as a profession-bound concept of designer's cognitive strategies, but as a comprehensive meta-disciplinary concept that broadens disciplinary reasoning and helps for example engineers to forget about the 'drawers' 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. This understanding of design thinking actually characterizes the d.school's educational philosophy as well as the main focus of the HPI-Stanford Design Thinking Research Program (Plattner et al. 2009). Thus our project focus is how to relate design thinking to the structures, cultures and processes of IT development – in particular with regard to the following aspects:

- *Building on Diversity*: Facilitating strong team diversity and frequent interdisciplinary communication and collaboration throughout the design process
- *Exploring the Problem Space*: Supporting a comprehensive shared understanding of the problem addressed before the actual development process starts, in particular by learning about the user and its social context from different perspectives

- *Exploring the Solution Space*: Promoting a creative ideation and conceptualization process by pursuing many alternative ideas on a rough-sketch level in order to learn about the most viable solution path
- *Iterative Alignment of Both Spaces*: Enabling a highly iterative development process with an early and continuous integration of user feedbacks based on comprehensible prototypes

To deal with this issue in detail, we will consider three questions. First, what approaches do exist that try to solve similar problems like design thinking and how is design thinking related to them? Second, how can design thinking in IT development be conceptualized on an operational level? Third, what implementation hurdles are there and are there ways to overcome them? In the subsequent section, we outline existing IT development approaches and their relation to design thinking.

3 Discussing the Context: Waterfalls, Agility, and New Design Professions

There have been various models to organize IT development projects and each of them tackles the issue of problem solving and knowledge processing in a different way. "Old school" IT approaches used to follow linear process logic as known from milestone-based project management: The process passes through several phases in a predefined order, separated through milestones that describe particular preliminary goals and timelines, and require an extensive phase documentation. The prime example of those models is the "waterfall model" that divides IT development generally in the following basic phases: *Requirement and specification analysis*, *program design*, *coding*, *testing* and *implementation* (whereas different descriptions of the waterfall phases vary depending on how detailed they are) (Royce 1970; Davis et al. 1988). The waterfall model seeks to build up knowledge about problem and solution very systematically, so that each step can build upon the outcome of the preceding ones. Knowledge processing is rather formalized via an in-depth documentation as a core completion criteria of each stage, not only with the purpose of recording knowledge for future maintenance and changes in the software code, but also to advance knowledge in an explicit condition to the subsequent phases. Overall, the waterfall process facilitates strong process and resource control as well as a strong clarity of that what is being developed. It allows project planning which is very comprehensible from an ex ante perspective and thus easy to communicate to clients and customers.

Nevertheless, the waterfall model does not always have a good reputation due to several reasons. As Boehm points out, "document-driven standards have pushed many projects to write elaborate specifications of poorly understood user interfaces and decision-support functions, followed by design and development of large quantities of unusable code" (Boehm 1988, 63). Even if feedback loops between the stages are not excluded, their effect on product improvements is very low due to

the rigid demand for documentation that slows down the feedback communication between the stages and makes the process, albeit potentially iterative, inflexible. Another problem is that cross-functional collaboration and mind-set building happens only on a poor level as there are mainly mono-disciplinary teams specialized in the respective stage's task. As a result, the waterfall model pursues a predominant technical perspective. The user perspective is considered only at the edges of the process: either by means of analyzing user or market demands and translating them into specifications *before* the actual development process starts, or by evaluating and testing the almost finished product by user feedbacks towards the end of the process. It has been shown that both ways of interacting with the user do not help to extend technical thinking, rather they have the opposite effect. While the translation of user or market demands into technical specifications helps to maintain a focal technical perspective throughout the development process, user feedbacks in the final stages mainly contribute to the elimination of discrete software errors. Changes of fundamental shortcomings in the general architecture would be extremely expensive at this late stage of development.

3.1 Overcoming the Waterfall with Agile Development

Further developments of the waterfall model pursue a more detailed view on software implementation in order to guarantee a better match between the initial specifications and the final system (V-model, see e.g. Höhn and Höppner 2008), or they integrate specific iteration phases that help to evaluate the progress continually throughout the process (Spiral model, see e.g. Boehm 1988). Yet, those concepts try to overcome the problems of the waterfall model by making the processes more complex and thus more difficult to handle. In contrast, with *agile development* (in particular *Scrum* and *Extreme Programming*), alternative approaches came up that bring in a fundamentally new perspective on organizing IT development (Beck 2003; Pichler 2008). The basic idea behind agile is to manage IT development not by rigid milestone-based process roadmaps that the team has to follow, but by a set of obligatory rules and roles in which the team can act flexibly in order to sustain learning about the project and adaptability to unexpected events. The requirement analysis is not a preceding step, but a parallel process to the actual development. The reason for this is one main feature of agile: the division of development circles into short iterative steps while the goal of each step is to produce an incremental intermediate solution that serves to generate feedback by users and specialists. These feedback loops, again, are the drivers of further development and refinement steps. Another feature is the strong focus on team collaboration and communication. For instance, in *extreme programming* there is (at least) an emphasis on programming in pairs, and *SCRUM* asks for an all-embracing one-team approach in which all disciplines involved in the development process (architects, developers, tester, documentation experts, …) pool their resources all the way through.

In comparison with design thinking, agile shows some strong parallels: core features like "user-centricity", "iterative learning and development processes", and "extensive team communication" seem to suggest that design thinking methodology has been already introduced to IT development. On closer examination, however, one can see crucial differences. Primarily, agile rather concentrates on continuous incremental refinements than on exploring and comparing radically new solution paths. As one of the interviewees, a software designer, stated, this can lead to highly contradictory ways of progressing:

> "In agile, you downsize the problem so that they're actually small enough that people can deal with it and make progress and don't get lost. But that's a very constraining technology. (…) Agile is always looking to remove options from the table. Design thinking is always trying to keep options on the table a long as possible."

According to this, agile seems to have a tendency to avoid divergent thinking in order to maintain the overall view on what to do next. This also means that the whole aspect of problem understanding in design thinking is limited down to the trial and error approach of iterative prototyping. This is why the focal goal of design thinking to put divergent options on the table will hardly be achieved. One other difference is that there is less emphasis on interdisciplinary creative collaboration than in design thinking. Although there is a strong emphasis on team collaboration, the people involved still are technically trained thinkers. A real expansion of thinking styles within the team does not occur. Thus, the major matter of concern in agile development is to reconfigure the way in which IT development projects are managed. It is not the aim to diverge the disciplinary composition of the development team – the focus still lies on engineering teams developing the software from start to finish.

3.2 Adding New IT Design Specialists

There have been serious efforts to introduce new design specialists to IT development in order to overcome the mentioned restrictions and encourage more interdisciplinary collaboration. In general, those specialists are assumed to take on the role of the "user's advocate" within the development team. The precise role of those specialists however varies. For instance, while *user-interface designer* mainly work on user-friendly digital graphic interfaces (Mandel 2009), *interaction designer* look also on the dynamic aspects of human computer interaction (Dix et al. 2003). Designers following a *user-centered approach* intend to both generate and validate IT design decisions on the basis of comprehensive user research (general information as well as specific feedback) (Vredenburg et al. 2002), but *user-experience designers* try to design the whole experience a product conveys in a such way that current users may not even be able to imagine (Buxton 2007).

This variety of concepts shows that the possible roles of IT designers professionally dealing with user perspectives in the development process are highly divergent. For instance, in case of the user-interface or interaction design the role is clearly

limited to what the software front-end actually should look and feel like. The role of user-centered design however is about both translating observable user needs into the software design and validating the software design through observable user feedbacks – as a kind of "coverage for user-friendliness" for the development team. In contrast, user experience design is much more wide-ranging because it embraces the whole software design process from specification to architecture. Accordingly, the demands on team collaboration highly vary. Whereas in user interface and interaction design, tasks can be clearly separated and carried out individually, user-interface and user experience design demand stronger collaboration between IT designers and IT engineers. In the last case, the designer would even gain a leading role in the whole process.

Those entire professional IT design concepts, and in particular user-experience design, show strong parallels to design thinking. They also emphasize learning about the user context in order to gain insights on novel solutions. They also pursue iterative and feedback-based learning, and they even apply similar techniques (e.g. observation techniques, ideation techniques, prototyping techniques). However, they differ from design thinking in so far as they concentrate on adding a new profession to the IT development team with the intention of bridging the gap between the potentials of technical thinking and a viable product by more manpower. It does not tackle the concept of collaboration or process management of the team itself. Having a user experience designer in the team would not mean that there is a guarantee for a shared problem understanding or a collaborative ideation process between design specialists, architects, and programmers. Thus, though concepts like user experience or user-centered design show strong similarities, those concepts still engender new *professional* disciplines added to the established team setting, whereas design thinking aims at influencing people meta-disciplinarily.

3.3 Design Thinking in the Context of IT Development Approaches

In sum, when comparing design thinking with the approaches to IT development discussed above, we see elementary differences:

- *Building on Diversity*: New IT design professions specialized on the user perspective mainly extend disciplinary diversity in IT development, whereas strong team-based collaboration is a core feature of agile development. Yet, a collaborative approach that tries to implement differing thinking styles in development teams as well as design thinking on a meta-disciplinary level is not explicitly addressed.
- *Exploring the problem space*: Understanding the user context has been professionalized by means of new design specialists. IT engineers still deal with the user's voice in a translated form: via specifications, which may be validated by user insights, but do not deliver the "full picture" needed for creative ideation.

Also, a team-based approach that uses a collaborative understanding of the user context to come up with radically new ideas has not been taken into account.

- *Exploring the solution space*: The whole issue of creative ideation is not explicitly included in IT development models. There could be a tendency in agile if the approach would not focus that much on incremental progress, which limits divergent thinking.
- *Iterative Alignment of both spaces*: There is a strong parallel between design thinking and agile concerning continuous iteration of user feedbacks based on prototypes throughout the process. However, a clear difference is that design thinking supports the iterative exploration of both spaces much more extensively than agile does.

4 Discussion: On the Challenges of Translating Design Thinking into Action

On a theoretical level, we have seen that design thinking can make valuable contributions to IT development and enhance the innovativeness of IT development. The question, however, how to apply it successfully to IT development is still an unsolved challenge and a central part of our ongoing research. In what follows, we will outline which elemental strategies to translate design thinking into action we have observed up to this point and will discuss essential conflicts and implementation hurdles.

Within the preceding discussion, we have conceptualized design thinking with four aspects: *exploring the problem space*, *exploring the solution space*, and *iterative alignment of both spaces* explain the workflow of design thinking from a knowledge-based viewpoint, while *building on diversity* aims predominantly at the team carrying out the workflow. One central insight of our research is that translating those aspects into practical action patterns can be carried out from two basic perspectives: the didactic perspective on the one hand, and organizational perspective on the other hand.

The guiding question of the didactic perspective is: How can one educate groups or individuals (in particular non-designers) in design thinking? We call those people who are both specialists in certain disciplines and trained in design thinking *t-shaped*, which means that one is able to look both horizontally on problems (going broad, looking for options, and being able to review the viable role of specialists within the development process) as well as vertically (bringing in the own disciplinary knowledge, solving aspects of the design problem which can be tackled through one's own expertise). In contrast, the guiding question of the organizational perspective is how one can shape those processes and structures in which people interact so that "designerly" product development will be supported. Thereby, we talk in particular about procedures and structures. While procedures in particular

include the workflow triassic "observe & synthesize, ideate & prototype, and revise & refine" (see introduction), structures refer to the team diversity itself as well as to those multilayered organizational aspects that either enable or support the process, including as diverging facets as corporate cultures, reward and control systems, and supportive tools and techniques. Summing up, whereas the didactic perspective intends to foster design thinking abilities by educating people, the organizational perspective tries to adjust the contextual and managerial conditions of product development projects. Although we pursue in our research project an organizational focus and thus look mainly at the organizational factors and strategies supporting or obstructing the implementation of design thinking, we have learned that we cannot exclude the didactic perspective due to the fact that both perspectives strongly presuppose each other: An effective design thinking strategy needs well-trained people as well as an organization that supports them instead of slowing them down.

4.1 The Didactic Perspective: Educating Design Thinking Competencies

The basic methodology of design thinking didactics has been elaborated at Stanford's School of Engineering. The basic philosophy of design courses for instance at ME310 and the d.school pursues a strong project-based learning approach – following the rule: training people in problem solving beyond scientific thinking asks for an education without theory lessons. Thus, design thinking students learn in interdisciplinary teams how to tackle a given design problem by exploring its problem space with hands-on research (e.g. by inquiries, interviews, observations, self-experiments), exploring its solution space with various ideating techniques (e.g. by brainstorming, sketching, prototyping), and aligning the ideas with the reality through repeated feedback that helps to refine or revise the selected paths towards a solution. The didactic goal of this kind of training is not only to develop practical insights into the essentials of dealing creatively with wicked problems, but also to learn to build up a shared team-based understanding of the problem that is not bound or disrupted by disciplinary thinking. Design thinking education, thus, teaches to generate a mutual knowledge and experience pool that helps to facilitate team communication on a meta-disciplinary level. In the course of our first year's research, we interviewed a group of IT professionals who attended design thinking workshops within a company training program. In those workshops, the participants were asked to go into a problem space that is close to their everyday work experiences from an explicitly non-professional perspective (i.e. mainly from the user's perspective) and brought into a situation in which they had to leave technical thinking behind for a while in order to find a viable solution concept. We intended to gain several insights into the acceptance and effectiveness of design thinking trainings for IT specialists from them. We asked them how they experienced those trainings, how they value design thinking in IT development in

general, and what they think about applying design thinking to their daily work routines. Conclusions from these interviews can be summarized as follows:

First, the majority of our interviewees were more skeptical about the benefit of design thinking before the workshop than afterwards. Most of them regarded the workshop experience as inspiring for their future personal work. In some cases, however, there was a tendency of reacting negatively to the use of too "flowery and cloudy" language when dealing with ambiguous situations. Also, some interviewees appreciated the creative openness of design thinking, but nevertheless doubted its viability as soon as a project is under time pressure, due to the fact that too much time would be spent *before* the development process for problem understanding. Second, almost all of our interviewees were interested in particular in the diverging aspects of design thinking, i.e. the broadening exploration of problem and solution space as a basic step of product development. They explain this interest in so far as this aspect is addressed only rudimentarily in their ordinary working background. Reasons for that lie in particularly strong tendencies towards converging activities due to established engineering-based problem solving patterns as well as common corporate pressure on coming up with results very quickly. Third, it was a striking insight that albeit most interviewees showed strong personal interest in design thinking methods, very few were able to apply it in their daily work routines. We rather gained much information about how an organization can hinder the application of design thinking. In sum, we learned that translating design thinking into action from a didactic perspective can work well on an individual or group level through a learning-by-doing approach, but it may come up against limiting factors when applying it within an organizational context.

4.2 The Organizational Perspective: Design Thinking as a Front-End Technique or as an Integrated Development Philosophy?

When it comes to the organizational perspective, we are confronted with two typical approaches: first, implementing design thinking at the very front-end of a development process, as a methodology to support the development of concepts for future products or the definition phase of a development project, and second, implementing design thinking as a comprehensive development philosophy with strong implications for organizational processes and structures. However, deciding about what design thinking strategy would work better has a lot to do with the (still open) question of how to bridge the gap between a company's need for reliable control of their processes and resource flows and an open and entrepreneurial approach to new product development. Applying design thinking in an organization presupposes good answers to this question, as it stands in contradiction to common management techniques such as stage gate innovation management that rely strongly on predefined workflows and standardized quality gates, anticipating and selecting solution paths in a restrictive manner so that explorative and creative solution paths become

rather constrained. Also, employees who are generally evaluated – apart from their ability of scientific reasoning – by punctual shipment and budgeted resource plans and thus work in tight and fixed time schedules with scarce resources, get in conflicts when they deal with product development approaches that entail the uncertainty of extensive divergent thinking. One core finding of our interviews is that *those employees who report to higher hierarchy levels perceive design thinking as a risk.* For them, it was more secure to plan milestone-based development processes (with a main focus on converging thinking), even if the results may be not as innovative as they could be. Against that background it is not surprising that design thinking-centered organizations have been built up first in university labs and agencies while established companies have been dealing with fundamental change management issues. When it comes to the decision to integrate design thinking within a company's internal organizational structure, it is rather likely to be regarded as a front-end technique. The intervention into the organization is in both cases rather small as design thinking workflows are either outsourced to external design agencies (such as IDEO), semi-autonomous research labs (such as the T-labs of the Deutsche Telekom), or are restricted to selected working methods which are integrated into the established organizational workflows.

Our research has led to the insight that the more design thinking is limited to fuzzy front-end matters, the easier is its implementation because there are hardly any conflicts with established development processes and the corporate reward, reporting and controlling systems. An IT development company following a waterfall approach, for instance, could apply design thinking techniques to the specification phase, and likewise, a company pursuing an agile approach can start with design thinking-inspired concept development before beginning the actual agile development process. We could find both cases in our case studies and in the latter case the question came up whether this is the ideal form of connecting design thinking with IT development because the agile logic would meet the aspect of aligning problem and solution space through strong team communication, continuous integration of user feedback and a self-organizing process roadmap.

However, we identified one problem that arose in both cases: *the risk of a fundamental disruption of the knowledge flow between front-end design thinking and subsequent development stages* due to dissimilar communication media used in design thinking and IT development. This can be traced back to conflicting usage of prototyping: Prototypes in design thinking generally are mock-ups that support the elaboration and evaluation of product concepts with the goal of finding out which ways are right or wrong. This means that they can be very experimental and consist of any material that allows achieving information about the ideas behind the concept (and not so much about its technical specifications). Consequently they have to get handed over to the development phase and must still be translated into technical specifications and task definitions. Design thinking prototypes have the main purpose of supporting learning about the underlying product concept. In contrast, software prototypes are generally made of the same material as the final product, and are – in the case of agile – continuously iterated into the final product. Hence, they focus less on learning about the general product idea but on finding a smooth way

towards a final solution without running into the wrong direction for too long. As a result of that difference, there is a serious danger of misconception regarding the use of prototypes between the protagonists of the design thinking phase and the development phase, so that a design thinker, on the one hand, does not know how to transfer their acquired knowledge to later stages, and a developer, on the other hand, does not know how to continue his work with the sort of information that is passed on to him. Therefore, the central problem of locating design thinking solely at the front-end of a development process is to find a modus operandi how to transfer design knowledge to the succeeding development stages without loosing it at the interfaces.

This is one major problem to which an integrated design thinking strategy seems to have better answers. The problem, for instance, to support a better flow of knowledge throughout the process could be answered through a comprehensive one-team approach, which means that (at least) core members of the team, who represent all available disciplines, are involved throughout the development. This would support – besides an enhanced *building on diversity* – a better mutual understanding as well as a general tacit design knowledge base in all parts of the process. This again, however, would presuppose a general project-centered organization that allows its employees to concentrate on one development project at the time instead of participating in numerous projects simultaneously. Also, there would be a call for new quality and controlling measures that do not cut off divergent thinking but still deliver an effective understanding for a project's steps forward. In particular a company's middle management would have to rely on those measures, as no manager would take on the risk of getting bad evaluations when they are a structural result of spending resources on design thinking.

5 Outlook

Overall, it is yet an open question, which ways of applying design thinking to IT development, apart from specialized agencies or research labs, are promising, and which are not. There still is a lack of good practice from which we can derive reliable insights, for instance with regard to the following questions: How do appropriate combinations of design thinking with common IT development models look like? What are the didactic and organizational implications? In which areas in IT development would design thinking be congruous? In our second year's research we will elaborate concepts responding to those questions, building on our accomplished and future case study research.

References

Beck, K.: Extreme Programming – Das Manifest. Munich, Addison-Wesley, 2003

Boehm, B.W.: A Spiral Model of Software Development and Enhancement. IEEE Computer, 21(5) May (1988) 61–72

Brown, T.: Design Thinking. Harvard Business Review June (2008) 84–92

Buxton, W.: Sketching User Experiences – Getting the Design Right and the Right Design. San Francisco, Morgan Kaufmann, 2007

Davis, A.M.; Bersoff, E.H.; Comer, E.R.: A Strategy for Comparing Alternative Software Development Life Cycle Models. IEEE Transactions On Software Engineering 14(10) October (1988) 1453–1461

Dewey, J.: How We Think. Mineola, Dover Publications, 1997

Dix, A.; Finlay, J; Abowd, G.D.; Beale, R.: Human-Computer Interaction. Harlow, Prentice Hall, 2003

Dunne, D.; Martin, R.: Design Thinking and How It Will Change Management Education: An Interview and Discussion. Academy of Management Learning and Education 5(4) (2006) 512–523

Höhn, R.; Höppner, S.: Das V-Modell XT – Grundlagen, Methodik und Anwendungen. Berlin Heidelberg New York (Springer), 2008

Krippendorf, K.: The Semantic Turn – a New Foundation for Design. Boca Raton, Routledge Chapman & Hall, 2006

Lawson, B.: How Designers Think. Oxford, Architectural Press, 2006

Lindberg, T.; Noweski, C.; Meinel, C.: Design Thinking: Zur Entwicklung eines explorativen Forschungsansatzes zu einem überprofessionellen Modell. Neuwerk, Zeitschrift für Designwissenschaft (2009) 47–54

Mandel, T.: Elements of User Interface Design. Hoboken, John Wiley & Sons, 2009

Martin, R.: The Design of Business. Boston, 2009

Newell, A; Shaw, J.C.; Simon, H.A.: The Process of Creative Thinking. In: Gruber, H.; Terrel, G.; Wertheimer, M. (eds): Contemporary Approaches to Creative Thinking. New York (1967) 63–119

Pichler, R.: Scrum – Agiles Projektmanagement erfolgreich einsetzen. Heidelberg, d.Punkt Verlag, 2007

Plattner, H.; Meinel, C.; Weinberg, U.: Design Thinking. Munich, mi-wirtschaftsbuch, 2009

Royce, W.: Managing the Development of Large Software Systems. Proc. of IEEE WESCON 26, (1970)

Rittel, H.W. J.: On the Planning Crisis – Systems Analysis of the First and Second Generation. Bedrifsökonomen 8 (1972) 390–396.

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