Design theory

Product Design and Development (ME30294). Lecture 10.
Jérémy Bonvoisin, Dept. Mech. Eng., University of Bath

Last update: July 24, 2018

Abstract

This lecture introduces some descriptive models of the design activity attempting to picture the essence of design. These descriptive models are put into perspective with more practical aspects of design and are used to explain the relevance of some content delivered in this module.

Contents

1 Prologue: why this lecture?

Along previous lectures, this module strove to provide an overview of good design practices. It aimed at providing methods to build up own's "designer's toolbox". It gave tips to create novel design concepts (lecture 1 and 6), to structure design processes (lecture 3). It mentioned some of the contextual aspects which are relevant to take into account in today's design practice (lecture 2 and 9).

While focusing on how to design, it left out the question of *what design is*. This lecture takes a step backward and reviews relevant theories in order to picture the essence of design. It provides explanatory frameworks to represent what happens when somebody designs something. These frameworks provide a basis to explain some of the oddities of design and justify why some aspects discussed in previous lectures are of relevance. Through this lecture, we hope providing a more thorough understanding of design, hence helping students to make connections between the concepts addressed in this module and therewith to ground their life-long design learning process. redhttps://www.slideshare.net/Leursism/design-theory-lecture01

2 Basic characteristics of design

Without taking any risk of saying something wrong, we can say at a very high level of abstraction that design is an *intellectual*, *teleological*, and *universal* process—three fundamental characteristics which are detailed hereafter. Any further attempt to be more precise about what design is requires making adventurous assumptions. The next sections provide different more or less conflicting theories of design, providing more or less compatible definitions, and shedding different lights on what design is.

Intellectual. Design is an intellectual activity in the sense it is virtual, it is a "pen and paper"-thing [Slide 1]. It is about creating normative information in the form of definitions, instructions, projections, representations (e.g. CAD models and drawings, assembly instructions, nomenclatures). This normative information is ultimately meant to be executed by physical processes to affect the real world. The distinction between physical and virtual is what distinguishes design and production. Nonetheless, while industrial production is a physical process of execution, it also embed a part of design, since the normative information it gets delivered is rarely precise enough to be executed *as is* and need to be refined by further detailed design. Also, design may require physical processes in the form of prototyping and testing in order to *validate* information.

Teleological. Design is teleological in the sense it is about *purposefully* "devis[ing] courses of action [in order to change] existing situations into preferred ones" [1, p. 111] [Slide 2]. Design is what makes the difference between natural sciences and professions (including engineering): while "[natural] sciences are concerned with how things are [..., a profession] is concerned with how things ought to be, with devising artifacts to attain goals" [1, p. 114]. In that sense, design is about defining a way to change a part of the reality to reach a given purpose. In engineering and industry, design is about defining artefacts, that is, artificial things which wouldn't exist without our intervention, such as products or processes.

Universal. So defined, design appears to be universal in the sense it is not only the remit of designers but of all professions [Slide 3]. “The intellectual activity that produces material artifacts is not different fundamentally from the one that prescribes remedies for a sick patient of the one that devises a new sales plan for a company or a social welfare policy state” [1, p. 111]. In that sense, design is common to engineering, architecture, business, medicine, and even arts like painting and music. This universality makes it difficult to distinguish design from other types of problem solving [2]: do I *design* when I define which route I will take to go and grab a coffee at the cafeteria?

3 Design as heuristic problem solving

Design can be seen as a procedure to solve an optimization problem. This amounts to recursively build a decision tree where each node is a move towards a solution and each tree level is a further level of specificity in defining the solutions [Slide 4]. In design like in optimization, a lot of real-life problems are beyond the reach of exact algorithms such as *brute force search* or the *minmax decision rule* [Slide 5]. For these problems, it is neither possible to list all the solutions nor to prove the absolute superiority of one of them. For example, while it is possible to define exact algorithms predicting the best move to do next in chess or to solve the travelling salesman problem, the volume of operations required to terminate these algorithms is beyond the reach of computational power. In other words: ‘Only in trivial cases is the computation of the optimum alternative an easy matter. [In] the real world we usually do not have a choice between satisfactory and optimal solutions, for we have only rarely a method of finding the optimum ” [1, p. 118-120]. Those non-trivial problems are only in the reach of heuristic algorithms which only guarantee to find fairly good solutions, in contrast to exact algorithms which guarantee to find the optimal solution.

3.1 Design by analogy

In design, the identification of possible “next moves” towards a solution is fueled by the designers’ experience, which can be represented as a database of connections between the problems and solutions they encountered in practice. Examples a designer has been exposed to are stored in their memory as ‘cases’ to be used in *analogical reasoning**. “Analogical reasoning is based on the idea that problems or experiences outside the one we are currently dealing with may provide some insight or assistance. [...] Analogy is a way of recognizing something that has not been encountered before by associating it with something that has. [It] can be used in common situations, where the previous experience is directly applicable, or in unique or creative situations, where the previous situation shares something with the new situation, but the differences are just as interesting as the similarities” [3, p. 1]. More generally, analogy is a mechanism we instinctively use in our efforts to understand the world around us or to explain how we understand it. For example, explaining the working principles of electricity often involves a comparison with water flow through a system of pipes [Slide 6].

The way analogical reasoning works is that the designer “is reminded of the previously solved problem because it has some relevance to the new problem. After the person recalls a previously solved problem, certain aspects of the previous problem’s solution are used in the new context and others are not [Slide 7]. [While designing a 18-meter long pedestrian bridge over a busy street, the previous example of] a pedestrian bridge with a span of 15 meters may be recalled [...]. The same design for the superstructure, such as the steel arch, may be used, but the span and sizes of the steel members will change.” [3, p. 1-4].

3.2 Practical implications

This suggests that the more cases a designer has memorized, the more they are likely to find design alternatives to feed the decision tree. For example, generating “a design for a bridge requires not only an understanding of the analysis of bridges, but exposure to examples of several bridge designs.” [3, p. 1]. This also explains why designers commonly seek for external stimuli even when they are not actively solving a problem [4], for example by browsing design magazines [Slide 8].

Also, experienced designers intuitively infer *design heuristics** out of their memorized cases. Design heuristics are “directives [...] which provide design process direction to increase the chance of reaching a satisfactory but not necessarily optimal solution” [5] [Slide 9]. These (first) implicit heuristics can be elicited and collected to be communicated to other designers. Example of heuristics bases are the Iowa State University’s 77 heuristics [Slide 10] or the 40 principles of invention involved in TRIZ [Slide 11]. The use of these heuristics in controlled as well as real design settings has been shown to increase the variety of idea generated and hence to support innovation [6]. Exposure to a variety of heuristics also proved to help developing expertise in design [7].

Take-aways of this section:

- *Most design problems cannot be computed to find an optimal solution.*
- *Solution search leans on designer’s experience.*
- *Designer’s experience grows with exposure to either personally experienced or observed examples.*
- *Designers instinctively infer out of examples design heuristics which guide their action.*
- *To increase experience: be curious, read design magazines, analyze existing designs, tinker around.*

4 Design as problem solving without a goal

One of the reasons why it is not possible to find all the solutions to a design problem is that design problems are ill-defined, like equation systems where there are more unknowns than equations. Design amounts to "problem solving without a goal" [1, p. 106] [Slide 12]. Ill-defined problems are those that have less specific criteria for knowing when the problem is solved, and do not supply all the information required for solution. The distinction between ill- and well-defined problems is based on the amount of information guiding the search of solution given in the task environment. Ill-defined problems are moreover subject to 'discoveries' potentially challenging the formulation of the problem: To "discover gold, one does not even have to be looking for it (although frequently one is), and if silver or copper shows up instead of gold, that outcome will usually be welcome too. The test that something has been discovered is that something new has emerged that could not have been predicted with certainty and that the new thing has value or interest of some kind." This aspect makes of design a rather unpredictable and opportunistic activity.

4.1 Co-evolution of problem and solution spaces

Instead, the design activity involves a co-evolution of the problem and the solution [8] [Slide 13]. "[The] formulation of the problem at any stage is not final [...]. As the design progresses, the designer learns more about possible problem and solution structures as new aspects of the situation become apparent and the inconsistencies inherent in the formulation of the problem are revealed. As a result, [...] the problem and the solution are redefined" (*ibid.*, p. 5). Consequently, it appears that design involves an "iterative interplay to 'fix' a problem from the problem space and to 'search' plausible solutions from the corresponding solution space" (*ibid.*, p. 1).

In summary, design is about refining *both* a problem *and* a compatible solution to this problem. Consequently, the design activity does not start with precise product specifications but produces them out of rough "hopes and aspirations", eventually gained from a study of customer needs [9, p. 73] [Slide 14]. The exit condition of this interactive process does not only lie in the consistency between the problem and the solution (when the product design is said to "meet" the specifications or "fulfill" the requirements), but in external factors: for example, time and money went out, or the team is satisfied with the degree of precision achieved in the formulation of both problems and solutions.

Attempting to define the best moment in the product development process for defining product requirements, Ulrich and Eppinger's reference lecture on product design and development says: "In an ideal world, the team would establish the product specifications *once* early in the development process and *then* proceed to design and engineer the product to *exactly* meet those specifications." [9, p. 73, emphases are not in the original text]. This sentence begins with "in an ideal world" because it acknowledges that problems do not precede solutions in design, although it would be way easier to handle product development as if it would be the case.

4.2 Practical implications

The co-evolution of the problem and solution space is illustrated by the difficulty to implement in practice rigid stage-gate process models such as the Waterfall model of engineering design [Slide 15] or the Pahl & Beitz design process [10]. These sequential design approaches start with defining the requirements. The design then only starts once the requirements are defined. While this process may work in cases where design has low chances to lead to unpredicted outcomes (e.g. slight redesign of a product based on a well-managed technology), it may not be beneficial in radically new product design. Indeed, in these cases, the refined understanding of the problem yielded by the design activity may lead to challenging the requirements, an event which requires iteration and which does not fit with the logic of stage-gate processes.

In reaction to these difficulties, the software branch came out in the 90's with new project management principles under the umbrella of "agile software development" [11] which then spread to other engineering disciplines like engineering design. Among the key principles of agile are a "close collaboration between the development team and business stakeholders" in order to make the users participate in the incremental refinement of requirements. This requires delivering "working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale", discussing the results with users in "face-to-face conversation" and being open for "changing requirements, even late in development". Examples of agile management methods in software engineering are Extreme Programming (XP) [Slide 16] which is based on short development cycles and Scrum which is based on weekly design sprints.

Considering design as a co-evolution of both problem and solution spaces also helps understand why customers may have difficulties to express their requirements in practice. A famous (but apocryphal) quote attributed to Henry Ford says: "If I had asked people what they wanted, they would have said faster horses" [Slide 17]. This iconic case illustrates that requirements may emerge only after a solution has been proposed. This problem is also a part of communication problem the widely encountered in software engineering and humorously referred to as the 'Tree Swing Cartoon' [Slide 18]. End users may be delivered with a solution which does not match their need even if they have been asked what they wanted at the beginning of the process. If they do not take part in the whole product development process, they don't participate in the iterative refinement of the problem and the solution. They don't profit from the better understanding gained by the development team, which may converge towards a direction which is not useful for the users. Avoiding this pitfall is another benefit of agile project management methods.

Take-aways of this section:

- *The word of saying "clearly stating the problem is half the way to solve it" also apply to design.*

- *In most cases it is not realistic to expect specifications can be all set at the beginning of the design process.*
- *Stage-gate processes are relevant for design projects with low probability of making discoveries.*
- *Agile processes are relevant for design projects with low probability of making discoveries.*
- *The process of refining the requirements may need to involve the users so the designers don't converge towards an unwanted solution.*

5 C-K theory

The "C-K theory" introduced by Hatchuel and Weil [2][1] defines design as the parallel expansion of two ensembles: the knowledge and concept space [Slide 19].

The "knowledge space" K is defined as the "space of propositions that have a logical status" for a given designer or team of designers. In other words, K is the set of all propositions a designer or a group of designers know to be true or false. The elements of this space are propositions designers can rely on and don't have to explore because they know these to be true or false. For example, a designer may know that knives are made to cut and airplanes can fly, they can take these propositions for granted.

The "concept space" C is defined as the space of propositions that have no logical status. In other words, C is the set of propositions a designer or the group of designers do not know whether they are true or false. For example, a designer may never have experienced a knife that can fly or a plane that can cut. The propositions "a knife that can fly" and "a plane that can cut" are neither true or false for them as long as they don't experience these or get proven it is not possible to do such things. Propositions of C are hypothetical ideas thrown in a design process waiting to be validated and transformed into knowledge. These are the creative bits of design.

From this, design is defined as a sequential process of cross-fertilization between C and K spaces through four types of operations building the so-called "design square" [Slide 20]:

- $K \to C$: this operator creates elements in C (new concepts) from elements of K (bits of knowledge). A concept can be generated by combining elements of knowledge. A designer knowing the proposition "airplanes can fly" and "there is a planet called Venus" could come to the proposition of an "airplaine flying on Venus". This proposition remains a concept (that is, an element of C) as long as he has not been proven whether it is possible to make an airplane fly on Venus or not (the answer here).

[1]All quoted sentences in this section are taken from this source ([2])

- $C \to K$: this operator creates elements in K (bits of knowledge) from elements of C (concepts). Transforming concepts into knowledge corresponds to what is generally called 'concept validation'. This can be done in various ways, such as "consulting an expert, doing a test, an experimental plan, a prototype, a mock-up". An element of C is transformed in an element of K when it is given a logical status (true or false).
- $C \to C$: this operator creates concepts out of concepts. It amounts to jump onto an idea to create a new one.
- $K \to K$: this operator creates knowledge out of knowledge. This brings us back to the conventional rules of deductive reasoning used for proving mathematical theorems or in syllogistic reasoning such as: "All men are mortal / Socrates is a man / Therefore, Socrates is mortal."

5.1 Application example

The case of a "nail holder avoiding to hurt one's hand while hammering" reproduced here is reported in [12]. The reasoning process starts with the concept {safe knocking a nail} [Slide 21], which is a combination of the elements of knowledge {safe} and {knocking a nail}. The designer or the design team know about {Knocking a nail} and can rephrase it as {hammer in right hand, nail in left hand, energy given by shocks}. Having recalled the element of K {hammer}, the designers can choose to challenge it and explore two refined concept alternatives {Safe knocking a nail with a hammer} and {Safe knocking a nail without a hammer}. As well, having recalled the element of K {nail in left hand}, they can challenge it and explor two further refined concept alternatives {Safe knocking a nail with a hammer and with the nail in the left hand} and {Safe knocking a nail with a hammer and without holding the nail in the left hand}. The designer team also knows about {safe} meaning {without knocking on fingers}. From this, they can derive alternative concepts to hold the nail without having the risks of knocking it with the hammer, such as {avoid the hammer to derive from the desired trajectory} or {protect the hand from the hammer}. These concepts can be refined into more precise concepts {a trajectory control device} and a {hilt} or {protecting glove}.

The same tree-shaped recursive search algorithm can be continued from any element from C or K, such as {safe hammering with hammer in right hand and left hand not holding the nail} [Slide 22]. The process stops when the leaves of a given branch of interest in the exploration tree are validated, that is, they are fed back in K using the $C \to K$ operator.

5.2 Practical implications

The C-K theory can be used to log the outcomes of a creativity session. Therewith, it offers a way to structure further creative processes and to systematically explore new alternatives by going back where a branch may have been under-explored. If further provides research with a formalism to record and analyze design processes.

Beyond this, the C-K theory underlines the important role played by prototyping in design and explains why companies have integrated R&D departments. The design process creates concepts that need to be validated and fed back into the knowledge space so they can be put into practice. While the validation of some concepts can be done with quick prototyping or mock-up techniques, the validation of some concepts may takes years of efforts. These more ambitious and visionary concepts requiring technological development are the job of the 'R' part in R&D. Engineers working in the 'R' of 'R&D' perform research in the sense that they intend to expand the knowledge space in large design endeavors. This research is 'applied' in the sense it remains teleological, that is, it is part of the design process (of the 'D' of 'R&D') its finality is "bring a concept to some form of "reality"".

The C-K theory also helps avoiding the difficulty to differentiate problems and solutions. It avoids the necessity to postulate that the aim of the design process is to make a product definition 'meeting' product specification—and we have seen in section **??** that we should not postulate that specifications precede design. Instead, the C-K theory does not make the difference between specifications and product definition which are both considered as 'concepts' as long as they haven't been validated.

Take-aways of this section:

- *Design is a process of expansion of the designer's knowledge and available concepts.*
- *Design terminates when concepts of interest are turned into kwowledge, that is, their feasibility has been proven and they can be turned into reality.*
- *Concepts are created by recombination of elements of knowledge.*
- *Research (in the sense of technological development) is a part of design.*

References

[1] H. A. Simon, *The Sciences of the Artificial.* MIT press, 3 ed., 1996.

[2] A. Hatchuel and B. Weil, "A new approach of innovative Design: An introduction to CK theory.," in *DS 31: Proceedings of ICED 03, the 14th International Conference on Engineering Design, Stockholm*, 2003.

[3] M. L. Maher, M. B. Balachandran, and D. M. Zhang, *Case-Based Reasoning in Design.* Psychology Press, 2014.

[4] M. Gonçalves, C. Cardoso, and P. Badke-Schaub, "Inspiration choices that matter: The selection of external stimuli during ideation," *Design Science*, vol. 2, Jan. 2016.

[5] K. K. Fu, M. C. Yang, and K. L. Wood, "Design Principles: Literature Review, Analysis, and Future Directions," *Journal of Mechanical Design*, vol. 138, pp. 101103–101103, Aug. 2016.

[6] S. Yilmaz, S. R. Daly, J. L. Christian, C. M. Seifert, and R. Gonzalez, "Can experienced designers learn form new tools? A case study of idea generation in a professional engineering team," *International Journal of Design Creativity and Innovation*, 2013.

[7] S. Yilmaz, S. R. Daly, C. M. Seifert, and R. Gonzalez, "How do designers generate new ideas? Design heuristics across two disciplines," *Design Science*, vol. 1, Nov. 2015.

[8] M. L. Maher, J. Poon, and S. Boulanger, "Formalising Design Exploration as Co-Evolution," in *Advances in Formal Design Methods for CAD*, IFIP — The International Federation for Information Processing, pp. 3–30, Springer, Boston, MA, 1996.

[9] K. Ulrich and S. Eppinger, *Product Design and Development.* New York: McGraw-Hill Higher Education, 5 ed., Aug. 2011.

[10] G. Pahl, W. Beitz, J. Feldhusen, and K.-H. Grote, *Engineering Design: A Systematic Approach.* London: Springer-Verlag, 3 ed., 2007.

[11] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, and others, "Manifesto for agile software development," 2001.

[12] A. Hatchuel, P. Le Masson, B. Weil, and others, "CK theory in practice: Lessons from industrial applications," in *DS 32: Proceedings of DESIGN 2004, the 8th International Design Conference, Dubrovnik, Croatia*, pp. 245–258, 2004.