Design Science as Nested Problem Solving

Roel Wieringa
Department of Electrical Engineering, Mathematics, and Computer Science
University of Twente, The Netherlands
roelw@cs.utwente.nl

ABSTRACT

Design science emphasizes the connection between knowledge and practice by showing that we can produce scientific knowledge by designing useful things. However, without further guidelines, aspiring design science researchers tend to identify practical problems with knowledge questions, which may lead to methodologically unsound research designs. To solve a practical problem, the real world is changed to suit human purposes, but to solve a knowledge problem, we acquire knowledge about the world without necessarily changing it. In design science, these two kinds of problems are mutually nested, but this nesting should not blind us for the fact that their problem-solving and solution justification methods are different. This paper analyzes the mutual nesting of practical problems and knowledge problems, derives some methodological guidelines from this for design science researchers, and gives an example of a design science project following this problem nesting.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Design science, regulative cycle, nested problems, practical problems

1. INTRODUCTION

Following Simon [34], March & Smith [20, page 253] introduced design science as attempts to create things that serve human purposes. Later, Hevner et al. [13] provided a methodological context for design science in which (1) business needs motivate the development of validated artifacts that meet those needs, and in which (2) the development of justified theories about these artifacts produces knowledge that can be added to the shared knowledge base of design scientists. This framework clarifies the interface of design science with its social environment and with the scientific knowledge base, and it makes clear that design and research are closely related activities. However, the interaction of these activities is not analyzed, and this may create methodological problems. In particular, practical problems and knowledge problems in design science may be identified, and this can create methodological problems.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DESRIST'09, May 7-8, 2009, Malvern, PA, USA.
Copyright 2009 ACM 978-1-60558-408-9/09/05...$5.00.

Practical problems call for a change of the world so that it better agrees with some stakeholder goals. *Knowledge problems* by contrast do not call for a change the world but for a change our knowledge about the world [39, 42]. These two problems are both first class citizens in design science, in that they can both be equally challenging and rewarding to solve, but they do have important methodological differences. Answers to knowledge problems are propositions claimed to be true; solutions to practical problems are changes in the world claimed to meet the goals of some stakeholders. As a consequence, solving practical problems involves investigating the goals of stakeholders, and evaluating solutions involves applying stakeholder criteria; but answering knowledge problems involves applying stakeholder-independent criteria of truth. The distinction is defined in section 2 and elaborated in section 4.

Teaching problem-solving methodology to undergraduate and graduate students in information systems engineering[1] has shown me that students often state practical problems in the form of knowledge questions. For example, (1) *"Design an architecture for exchanging information between hospitals and insurance companies"* is a practical problem, because it calls for a design of a change in the world. However, to make it look like a research problem, students often state in the form (2) *"What is an architecture for exchanging information between hospitals and insurance companies?"* This is exactly the same practical problem, but stated in a way that makes it very hard to distinguish it from the knowledge question (3) *"What is the architecture for exchanging information between hospitals and insurance companies?"* Question (2) differs in only one word from question (3). However, (3) is a knowledge question, that asks for information about the currently existing information exchange architecture between hospitals and insurance companies, whereas (2) asks for the specification of a new artifact. Question (2) is a design question and this question is more clearly formulated in (1). To answer this design question, the problem solver must identify stakeholders and their goals, derive design criteria from these, investigate how a proposed solution would

[1] `http://wwwhome.cs.utwente.nl/~roelw/DesignScienceMethodology-handout.pdf`

achieve a goal in a given context, analyze trade-offs between alternative solutions, and assess the sensitivity of a solution to changes in context; tasks that are not present in answering a knowledge question like (3). To answer knowledge question (3), the researcher should investigate existing artifacts and describe them, and then validate the description. Expressing problem (1) in the form of problem (2) is bound to create methodological trouble.

The goal of this paper is to clarify the methodology of design science by first distinguishing practical problems and knowledge questions as sharply as possible (section 2) and then identifying some important knowledge questions that occur within the practical problem solving cycle (section 3). This allows us to rationally reconstruct the logic of design science as mutually nested problem solving cycles, in which solving a practical problem may lead the problem solver to ask knowledge questions, and answering these knowledge questions leads the problem solver to new practical problems (section 4). Reflection on this nested structure leads us in section 5 to a discussion of the terminology of design science, of scientific research methods, of levels of theory, and of the role of conditions of practice in design science. In section 6 I compare the proposal of this paper with other work. Section 7 concludes by extracting from the paper some guidelines for the aspiring design scientist.

2. KNOWLEDGE PROBLEMS VERSUS PRACTICAL PROBLEMS

In this paper the view is taken that design science projects are problem solving projects. The first distinction to be made then is between kinds of problems to solve. I define a **practical problem**[2] as a difference between the way the world is experienced by stakeholders and the way they would like it to be, and a **knowledge problem** as a difference between current knowledge of stakeholders about the world and what they would like to know.

Examples of practical problems are the problem of getting electricity from suppliers to consumers [14], increasing the altitude and speed of airplanes [6], providing a national infrastructure for the exchange of patient data between hospitals and health insurance companies, and developing a web site to sell last-minute airplane tickets. In practical problems there is someone who wants to change something in the world in the near or more distant future. Examples of knowledge questions are the questions when a rigid beam breaks under weight [38], why light bulbs get blackened by ductile-tungsten filaments [28, pages 121–127], what the performance of a database under different workloads is, and what the cheapest ticket to Sidney is. Answering these questions tells us something about the world without changing it—but our knowledge about the world changes.

The reason for making the distinction is that practical problems are solved by changing the world in accordance with stakeholder desires, but knowledge problems are solved by formulating propositions about the world [41, page 52]. This different problem-solving approach in turn implies a difference in evaluation criteria for proposed solutions. The evaluation criterion for solutions to practical problems is found by analyzing the problem, namely by identifying solution criteria, often called requirements [10], based on stakeholder goals. The evaluation criterion for answers to knowledge problems does not depend on stakeholder goals but is found by evaluating its truth value with respect to the subject domain of the knowledge.

For example, the practical problem *"improve the architecture for exchanging information between this hospital and these insurance companies"* calls for an improvement of the world, viz. of some IT architecture. To solve the problem, the problem-solver must identify stakeholders and their goals, derive criteria from these in order to find out what kind of improvement is called for, propose an architecture that he or she thinks will meet these criteria, validate this proposal against the criteria and then, if the sponsor of the project approves, implement the selected proposal. By contrast, the knowledge question *"What is the architecture for exchanging information between this hospital and these insurance companies?"* asks for information about something that exists. This could be a subproblem of the above practical problem but to answer this subproblem, the problem solver should study existing documentation, possibly analyze source code, verify his or her understanding by interviewing maintenance personnel etc. The result is not a changed architecture but a report about the existing architecture.

Henceforth, I will talk of practical *problems* and knowledge *questions,* which have *solutions* and *answers,* respectively. Their problem domain, the part of the world where the practical problem exists or the part of the world that the question is about, may be very particular or very generic. The practical problem "paint this house" leads to the knowledge question "what color do you like?", both having a particular problem domain, namely this house at this moment, and your color preferences at this moment, respectively. There is no deep research interest served by answering these problems. The practical problem to deliver propulsion technology for airplanes at high altitudes and high speeds has a very generic problem domain, namely the propulsion of airplanes of different sizes at different (high) altitudes at different (high) speeds under different weather conditions. And the question what the cause of blackening of a light bulb is, has as domain all light bulbs constructed of this material. The scientist who answered it, Irving Langmuir, received a Nobel prize for this work [28]. The following two sections analyze the logic of practical problem solving in both particular as well as generic cases; we return to the classification of the genericity of knowledge in section 5.

3. THE REGULATIVE CYCLE

3.1 Tasks in the regulative cycle

March & Smith [20, page 253] have defined design science as attempts to create things that serve human purposes. Here

[2]We called this a world problem earlier [42], because this makes clear that the outcome of practical problem solving is a change of the world, just as the outcome of answering a knowledge question is a change of our knowledge. However, this is not quite right because both kinds of problems concern the world, but with the purpose of changing it resp. of learning about it. The term "action problem" is more accurate than "world problem", but to many people this suggests incorrectly that we are talking about action research only. The term "practical problem" fits nicely with the term "conditions of practice" introduced later.

I generalize this to also include attempts to improve *existing* things to serve human purposes better. This makes a design science problem a practical problem as defined above. We will structure a design science project as a set of nested problems in which the top level problem is always a practical problem. A logical structure for solving practical problems is the regulative cycle, shown in figure 1.[3]

The **regulative cycle** [35, 41] starts with an investigation of a practical problem, itself the outcome of solving earlier practical problems; then specifies solution designs, validates these, and implements a selected design; the outcome of which can then be evaluated, which could be the start of a new turn through the regulative cycle. Researchers in such various fields as product development [7, 15, 29], systems engineering [12], architecture [1], mechanical engineering [25] and psychological practice [35] have identified, with some minor variations, the regulative cycle as the logical structure of rational practical action. The regulative cycle is the general structure of a rational problem solving process [21, 33]: Analyze the current situation and current change goals, propose possible changes to meet those goals, evaluate possible changes and select one, apply the change and then start all over again.

We will use the regulative cycle as our conceptual framework for the logic of practical problem solving. So we decompose practical problems into knowledge subproblems and practical subproblems (figure 2). As illustrated later, the practice of solving practical problems is messier than the simple structure in figure 2, because rationality is not the structure of the real world but of our attempts to deal with the real world. However, we will see that the regulative cycle provides useful guidelines for solving practical problems. We now look at the structure of the regulative cycle in more detail.

Problem investigation. Problem investigation is a knowledge question, because it asks for information about and understanding of the given problem, without yet changing it. The goal of problem investigation is to describe the problem, to explain it, and possibly to predict what would happen if nothing is done about it.

It is useful to distinguish four non-exclusive reasons for investigating practical problems, namely problems, goals, and impacts. Each of these leads to different emphases in the problem investigation process.

- In **problem-driven investigation,** stakeholders experience problems that need to be diagnosed before solving them. For example, a company may experience problems with supplying its different branches, located in different cities. Problem investigation then starts with a description and diagnosis (explanation) of these problems. Important tasks in problem-driven investigation are describing problematic phenomena, formulating and testing hypotheses about their causes, and identifying priorities for problems to be solved.
- **Goal-driven investigation** considers a situation in which there may be no problem experienced but where there are nevertheless reasons to change the world in agreement with some goals. For example, hospitals and insurance in the Netherlands experienced no problem in exchanging data, but a new law requires them to exchange new kinds of data for new kinds of goals, and so a new IT communication infrastructure needed to be designed. Problem investigation in this case starts with an analysis of goals to be achieved rather than with problems that stakeholders experience. Important tasks in all cases of solution-driven problem investigation are describing stakeholder goals, defining and operationalizing them, and identifying priorities of goals.
- In **solution-driven investigation,** technology is in search of problems that can be solved with it. For example, from a consumer point of view, new technology, such as mobile information technology or cloud computing may bring new goals within reach of potential users of this technology. Problem investigation in this case would start with an investigation of the properties of the new technology; and solution design would be an exploration of ways in which it could be used to achieve new goals. From a manufacturer's perspective, new technology could be developed to open up new markets, such as the development of video recording technology for the consumer market in the 1970s. Problem investigation in this case consists of making an inventory of goals and of current technology, and in identifying functionality and performance requirements for new technology.

  More innovative even is the search for new technology that can solve problems not yet experienced. The development of jet propulsion technology in the early 20th century [6], was triggered by the expectation of some engineers that in the future, contemporary aircraft propulsion technology would reach performance limits. In this case there was no experienced problem, but a possible future technical possibility. Problem investigation consisted of clarifying performance goals for the new technology at the same time as the technology was being developed.
- In **impact-driven investigation,** also called evaluation research, we focus on the outcome of past actions rather than preparing for the design of future solutions. For example, we may evaluate the cost savings realized by an ERP implementation or we may evaluate the performance of the current generation of video recorders on the market. Important tasks in evaluation research are describing solutions implemented earlier, identifying their impacts and explaining these impacts in terms of properties of the implemented solutions, identifying relevant stakeholder goals, translating these into criteria and applying these to the impacts.

These categories are not mutually exclusive but to the extent

[3] I have called this the *engineering cycle* elsewhere [42, 39] There is nothing wrong with this term but to avoid the impression that in this paper I restrict myself to technical design science I use the more general term "regulative cycle", borrowed from Van Strien [35], which refers to exactly the same cycle.

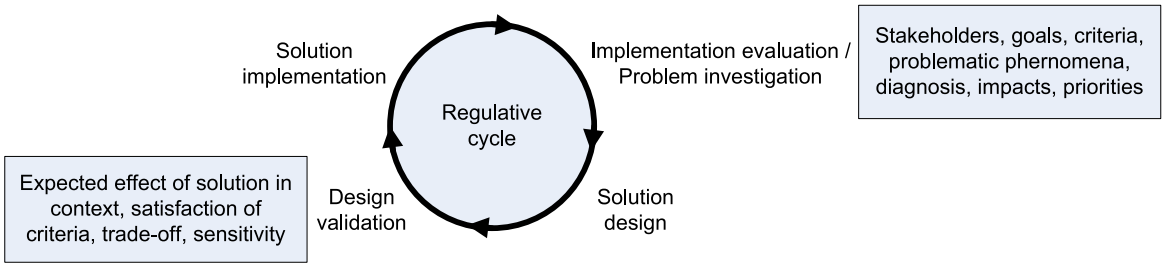


Figure 1: The regulative cycle.

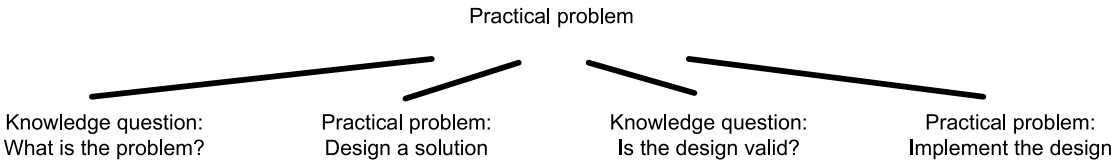


Figure 2: A simple decomposition of practical problems into subproblems.

that each of them is present, certain aspects of the problem will be emphasized and other aspects de-emphasized.

Solution design. Calling this task *solution design* is optimistic because it may blind the reader to the possibility that "solutions" may be designed that after implementation will turn out to have made things worse, at least for some stakeholders. "Improvement" has the same problem; "artifact" may blind the reader for organizational solutions, and "treatment" and "intervention" have the opposite problem. "Provision" [2] has none of these problems but is rarely used. So I will use the term "solution" but use some of the the other terms if they fit the context better.

I would like to avoid the suggestion that a solution is completely specified before being validated or implemented—often it is not. Even when not specified completely, though, solutions are *designed*. I follow the Merriam-Webster dictionary by viewing design as a plan in which the means to an end is laid down.[4] The *end* to be reached consists of one or more stakeholder goals. The *means* to reach this end is the solution designed by the problem solver, and in information systems may consist of a new or improved technique, notation, instrument, device, algorithm, process, business structure, etc. to reach the desired stakeholder ends. The plan to reach the end is always *specified* to some extent, because it must be communicated to others. And this specification may take many forms, ranging from natural language descriptions to formal specifications, diagrams, sketches, blueprints, mathematical models, scale models, prototypes, etc. or any combination of these.

The reason for classifying solution design as a practical problem in figure 2 is that designing is the formation of a commitment by stakeholders to improve the world in a certain way. The practical problem solver help stakeholders form this commitment by offering possible solution designs. Forming the design commitment is a significant change in the world that reduces the gap between what stakeholders experience ("we are uncertain about what to do") and what they would like to experience ("we are sufficiently certain about what to do"). The resulting solution specification is not a description of the world, for the specified solution does not exist yet; nor is it an explanation, because it is a specification of a possible future change rather than an explanation of what happened; it is not even a prediction, for the decision to apply this change must still be taken, and after it is taken, the specification may be implemented incorrectly. A solution specification does not have a truth value because it does not describe a past, current or future state of the world. Rather, the specification is the expression of a possible commitment to act in a certain way.

Later, after the solution design is implemented, we may ask wether the implementation is correct. Even then, the solution specification is not a description of the implementation, because if the implementation is incorrect with respect to the specification, we conclude that the implementation is wrong and not that the specification is false.

In terms of speech acts, creating a specification is a directive speech act [31]. By specifying a solution, the design scientists influence the behavior of stakeholders; and by accepting a specification, the stakeholders commit to behaving in a certain way.

Design validation. Design validation is a knowledge task in which we ask whether the specified design, if implemented correctly, would indeed bring stakeholders closer to their goals. There are three important knowledge questions in design validation:

V1 **Internal validity.** Would this design, implemented in this problem context, satisfy the criteria identified in the problem investigation? This contains two subquestions.

[4] `http://www.merriam-webster.com/dictionary/design`.

E1 **Causal question:** In problem domain D, would solution S have effects E?;

E2 **Value question:** Do E satisfy stakeholder criteria C?

V2 **Trade-offs.** How would slightly different designs, implemented in this context, satisfy the criteria?

V3 **External validity (a.k.a. sensitivity analysis).** Would this design, implemented in slightly different contexts, also satisfy the criteria?

These are knowledge questions. Their answers are propositions, claimed to be true. Importantly, and typical of design research, they are all predictions: What would happen in the future if this design were implemented in that context? The object of interest, an implemented solution, does not exist yet. This sets it apart from evaluation research, where we ask questions like E1 and E2 about an implemented change. We return to these questions when we discuss the design argument below.

Implementation. The term "implementation" is overloaded. In this paper I take the view that what constitutes an implementation depends on what solution has been designed. If the goal was to bring a video recorder on the consumer market, and a process to reach this goal has been designed, then an implementation of this design is the execution of this process; but if the goal was to try out a new video recorder design, and series of tests has been designed to do this, then an implementation of these designs is an execution of these tests according to these plans.

We have now identified a large number of knowledge questions in the regulative cycle, that have already been briefly summarized in figure 1. Note that some of the knowledge questions in a regulative cycle can be answered without starting a research project but by taking the answer from an existing knowledge base. However, other knowledge questions, such as problem diagnosis or solution validation, may require research projects to answer them. So an instance of the regulative cycle may or may not contain research problems.

3.2 Rational reconstruction of a practical problem solving process

The process of solving a practical problem is a sequence of activities and events as they could be written down in the diaries of the problem solvers involved in it. This process does *not* necessarily follow the regulative cycle. Already in 1971 Bruce Archer remarked that a design process does not start with a statement of requirements, leading up to a specification of a solution unknown at the start of the process [2]. Rather, the practical problem solving process starts with obscurity about the requirements, or about the the practicability of a possible solution, or about the match between requirements and solution; and it leads to a state with a requirements/solution match that contains an acceptably small amount of residual misfit and obscurity [2, page 17]. This view has been corroborated by empirical studies of designers, which show that experienced designers start with matching an incompletely specified solution to an incompletely understood problem and then jointly elaborate their solution specification and problem understanding [8]. We also find this view in systems engineering, where the "doctrine of successive refinement" says that the problem solvers iterate through the regulative cycle at increasingly detailed levels of understanding and design specification, starting with incomplete and vague problem/solution descriptions, and ending with a sufficiently clear set of problem/solution descriptions [23]. And in industrial product development this phenomenon is known too, where it is called "concentric development" [29, pages 26, 112].

So what does the regulative cycle represent, if not how the project actually evolves? It is a template for *rational reconstruction* of a practical problem solving process that lumps together what problem solvers did to understand the problem, to design an improvement, to validate it and to implement it. A rational reconstruction of a problem solving process is a chain of tasks from problem to improvement implementation that we construct to justify the improvement to others [41, pages 368, 372–373].

The concept of rational reconstruction has been used to discuss the relationship between history of science, which studies knowledge acquisition processes in history, and the philosophy of science, which studies the logical structure of justification of scientific knowledge [17]. The logical justification is a rational reconstruction of what actually happened. The same concept has also been identified by Parnas and Clements [26] in software engineering, who call it *faking* a development process. And in the ethnographic study of administrative processes, Suchman [36] showed that in offices, a lot of activity goes into rationally reconstructing administrative events such as receiving payments, paying out refunds, and sending bills, as if they happened according to the official procedures. These reconstructions are not falsifications of history but constructions of accountability, according to the rules of accountability applicable to the reconstructed process.

3.3 The design argument

Crucial for being able to justify a solution design is the ability to justify its internal validity, which is the claim that a correct implementation of the design solves some practical problem for a stakeholder. This argument occurs both in design validation and in problem evaluation, and I state it here in time-neutral way.

E1 **Causal claim:** In problem domain D, solution S has effects E;

E2 **Value claim:** E satisfies stakeholder criteria C

The *problem domain* D consists of the part of the world with which the solution will interact, such as the phenomena experienced as problematic by stakeholders, or the entities with which the solution will interact in order to achieve certain effects. After introduction of the solution in this domain, D has become the *context of use* of S.

The effects mentioned in the causal claim only result if the solution interacts with the domain: The new video recorder has to be used by consumers in their home to record a TV program from a cable TV provider (but it may fail to produce the desired effects in certain countries), the new ERP package has to be used by procurement officers to buy new material (and may reduce or increase the procurement process, depending on the context of use); the medicine will create a certain effect in John Smith (and possibly another in Jay Jones), etc. E1 is a claim about the effect of the solution in a context; Different contexts of the same solution may lead to different effects. The value claim then compares the effects with criteria derived from stakeholder goals.

The design argument occurs in various forms in problem investigation and design validation in the regulative cycle. If the problem investigation is *impact-driven,* then the above design argument precisely is the structure of realist (non-positivist) evaluation, in which outcomes are identified and attributed to the interaction between solution and problem domain, and then evaluated against stakeholder goals [27]. The realist evaluator is interested in the mechanisms that make the interaction between S and D create effects E. For example, an ERP implementation (solution) implemented in a particular company (problem domain) may turn out to fail to reduce costs (effects), which violates the stakeholder goal of reducing costs. Investigation of the implementation reveals the mechanism that causes this unexpected effect: In this company, employees received training in the use of the ERP package, which raised their competence level, and this caused these employees to look for and find a better paid job elsewhere, forcing the company to spend more money on hiring and training new employees. In other companies, this effect may be absent.

If problem investigation is *problem-driven, goal-driven* or *solution-driven,* part of the argument may be omitted because it is not needed in order to justify possible solutions. For example, in goal-driven problem investigation only the operationalization of goals into criteria C may be performed, and in problem-driven investigation, only description and evaluation of effects E may be performed.

As we saw, internal validation of a design follows the design argument too. Here we are looking to a future action instead of evaluation the outcome of past actions. In software engineering this is known as the "reference model for requirements and specification" [11], where in terms of this paper the *requirements* are stakeholder goals, and *specifications* are specifications of solution designs.

External validation of a solution design, finally, is analyzing whether the design argument remains valid if we would change the term D in it; and trade-off analysis consists in varying the term S.

4. PROBLEM NESTING

4.1 Problem classes

In order to analyze problem nesting it is helpful refine our classification of problems. Figure 3 shows the classification of problems that we will use. We have already distinguished two kinds of practical problems, namely design problems, in which we construct a specification of a solution to a practical problem, and implementation problems, in which we implement a specification (figure 2). We have also encountered four kinds of knowledge questions, namely (1) description and (2) explanation, which are likely to occur in problem investigation to find out what problems need improvement and what their causes are; (3) prediction, which is done in design validation in order to estimate what the effects of an implementation would be; and (4) evaluation, in which observed, diagnosed or predicted facts are compared against criteria (figure 2).

- Practical problems
  - Design problems. How to construct or improve something?
  - Implementation problems. Implement an improvement design.
- Knowledge questions
  - Conceptual questions
    - Conceptual modelling: Which concepts do we use?
    - Conceptual analysis: What are the conceptual relationships among these concepts?
  - Empirical questions
    - Description: What are the facts?
    - Explanation: What are their causes?
    - Prediction: What are their effects?
    - Evaluation: How do the facts/causes/impacts compare to criteria?

Figure 3: A classification of problems.

In figure 3 we group these knowledge questions into the class of empirical questions and contrast these with a second group of knowledge questions, namely conceptual questions. Where empirical questions ask for facts, their causes and their impacts, conceptual questions ask for the concepts by which we describe and classify these facts. For example, when developing a personnel information system we need concepts such as employee, contract and organizational unit to define the meaning of the data in the system. This is conceptual modelling. At a higher level of aggregation, we may make a reference model of the subject domain of personnel information systems in an industry branch, and at a yet higher level we may make a model of concepts needed to describe the role of personnel information systems in realizing company mergers. At each of these levels, concepts are needed to describe the problem and the effort to define these concepts is conceptual modelling. The source of the definitions may be the stakeholders in the domain, or a discipline such as HRM management. Conceptual modelling thus may range from fixing a set of concepts that define the meaning of data in an information system to adapting concepts developed elsewhere to the domain at hand.

In information system development, conceptual modelling is more than developing a dictionary to be used in a particular domain. When developing a particular information system for a particular company, the developer may discover that there is no such a dictionary in this company, or that dif-

ferent stakeholders may have different dictionaries—as in company mergers. Since the domain contains stakeholders who use concepts to structure their domain, conceptual modelling involves a dialog with these stakeholders, in which relevant concepts are jointly identified and defined.

The same is true if we are investigating rather than developing information systems. Information systems are subsystems of organizations that contain people who interpret their world, and whose world partly consists of interpretations. The researcher then tries to understand these interpretations, a situation called "double hermeneutics" in social science [30, pages 29 ff.]

In both information systems development and information systems research, interaction between the developer/researcher and the social system that is studied may lead to a change of concepts used in the domain, a phenomenon shared with all social studies. Social systems have a history [27, pages 72 ff.]. This does not invalidate the classification of figure 3, but it does indicate that it should be used with care because in reality some problems may have features of more than one problem class. In particular, empirical questions may contain conceptual questions and vice versa. This is in line with the claim of this paper that problems are nested.

Nesting should not blind us for the difference between a problem and its subproblems. The classification of figure 3 is presented here because the information systems developer/researcher needs to do different things in order to solve different problems, even if a problem of one kind is nested inside a problem of another kind. I will illustrate how this works out in practice in the next section.

4.2 Example: Problem-driven research

Figure 4 shows the nested problem structure of the Ph.D. thesis of Bela Mutschler [22]. The large transparent box is the decomposition of the practical problem problem how to improve effort estimation of process-aware information systems (PAIS) in an industrial company X. This problem was tackled by performing the first three tasks of the regulative cycle, namely problem investigation, solution design, and design validation. The transparent box contains three knowledge questions that inquire into the problem, namely Which problems are experienced (box (2) in figure 4), What effort estimation techniques are available (box 5) , and How do these fare on solving these problems (9). Next, a design problem (10) calls for an alternative technique that avoids some of the problems of existing techniques, and finally the designed technique is internally validated (14). Having specified and validated a solution to the design problem, the researcher looked back and drew lessons learned about the problem-solving process itself (18). This resulted in some improvement suggestions that could help researchers who want to do a similar project later.

We can make a number of interesting observations of this decomposition. First, a glance at the table of contents of Mutschler's thesis [22] would reveal that this problem decomposition has been mapped onto the table of contents of the thesis. This is a general feature of this kind of problem decomposition: They help the researcher structure their report about the research project. This is in line with the role of the regulative cycle in the production of accountability.

Second, knowledge questions can contain practical problems as subproblems. For example, the knowledge question what the current problems experienced in company X are (2) is decomposed into a number of problem-driven questions about problems and stakeholders (3). To answer these questions, a practical problem must be solved, namely how to get the relevant information. This problem has been solved by conducting an internet survey in company X and by interviewing some project managers (4). We can generalize this: To answer a knowledge question, we have to do something, for example ask someone, or look it up in a knowledge base, or do original research. Answering a knowledge question is thus itself a practical problem.

Third, problems can be decomposed forever, but in practice, decomposition can stop when we reach a subproblem for which we know what to do to answer it. For example, conducting an internet survey and interviewing project managers (4) have not been decomposed further because they are well-known research tasks which have more or less standard decompositions. By contrast, making an inventory of existing effort estimation approaches (5) turned out to be unexpectedly complicated, because the researcher wanted to classify existing approaches, and there turned out to be several conflicting taxonomies. The new problem then was to make a taxonomy first (6). This too was solved following the regulative cycle (7), which contained, among others, a literature review of existing taxonomies. Only then the existing effort estimation techniques could be classified according to the taxonomy designed by the researcher (8). This involved another literature review.

Fourth, design problems can contain knowledge questions too. For example, using causal loop diagrams (CLDs) to model qualitative relations among cost factors (10) was part of the design of a solution. It included knowledge elicitation sessions with project managers (12).

Fifth, sometimes, answers are found to knowledge questions that were not asked. For example, the research collected modelling guidelines for CLDs, although this answers a question not asked by company X (13).

Sixth, Mutschler encountered knowledge questions that required original research to answer, and no research method is banned from being used to answer these questions.. Mutschler used survey research combined with interviews (4, 12), meta-research (structured literature reviews) (7, 8), experiments (15), and case study research (17). The experiment in (15) was done using a student project, the pilot project in (17) was a real project in company X where the manager used the effort estimation models developed by Mutschler.

Seventh, although the regulative cycle is recognizable in the leftmost stack of boxes in figure 4, the problem decomposition is a lot messier than the neat cycle of figure 1. This illustrates the point made earlier about rational reconstruction. Actual research projects contain the logic of the regulative cycle but in addition contain many factors that one may call *conditions of research practice,* unforeseen conditions regarding available knowledge and research resources

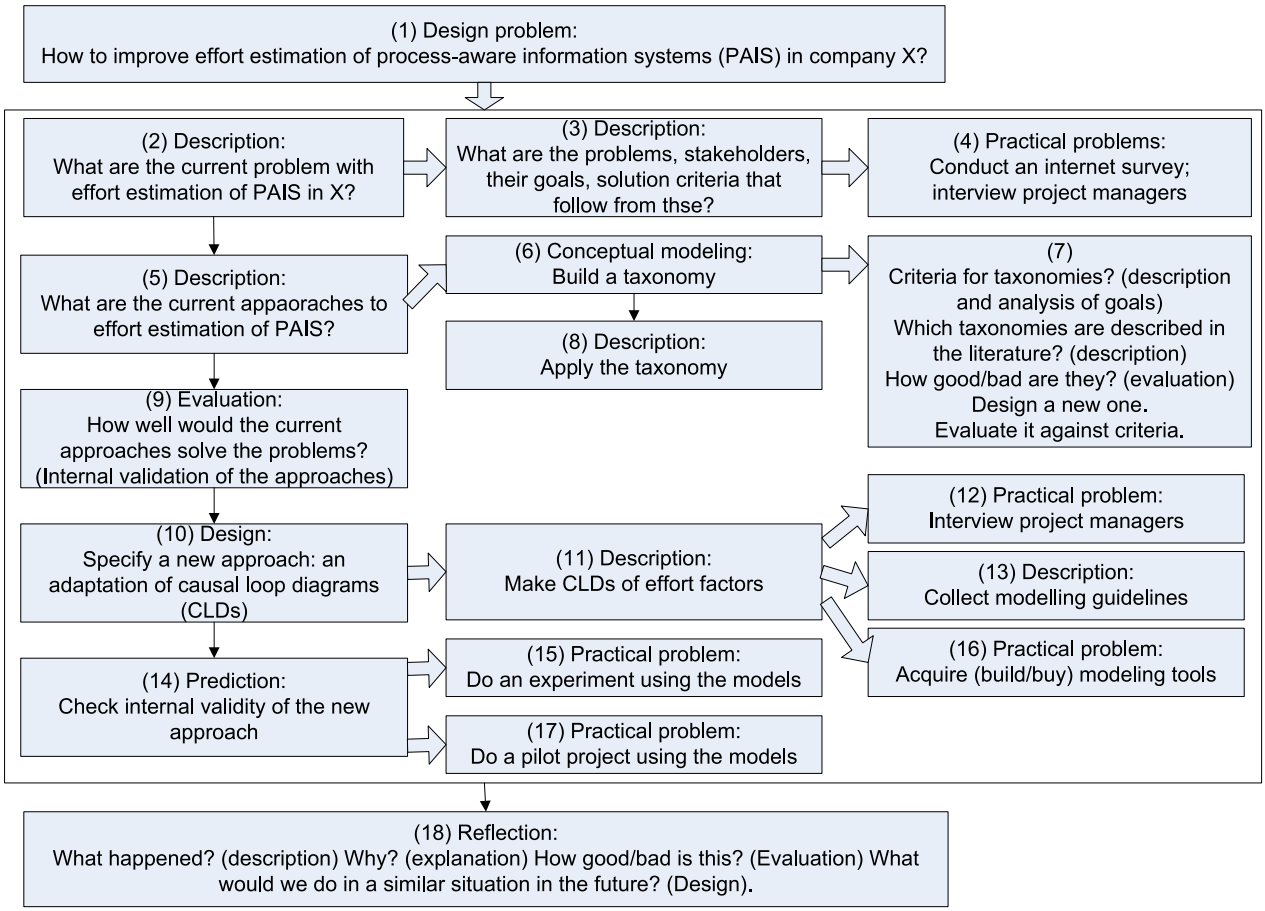


Figure 4: Nested problem structure of a Ph.D. project. Hollow arrows denote problem decomposition, single arrows denote temporal sequence. The numbers do not indicate sequence but are for ease of reference.

that have to be dealt with. Figure 4 is a rational reconstruction too, but now of the problem nesting of Mutschler's project. It is closer to what actually happened, but a day-to-day log of his project would reveal many iterations and false starts not reported in the thesis nor visible in the problem decomposition of 4.

Eighth, one deviation from the regulative cycle is that trade-off analysis was part of design (10) rather than of validation (14). The literature review of existing methods revealed a number of weaknesses that made some methods less suitable for the projects studied by Mutschler, and he positioned his CLD method as a complement to existing methods that would avoid these weaknesses. Sensitivity analysis has not been done because so far only one pilot project has been performed. More pilot projects in different contexts would reveal more about the sensitivity of the CLD technique to differences in project context.

5. DISCUSSION

5.1 Design science

In section 3 we identified design science problems as practical problems, and then in sections 3 and 4 avoided the term "design science" as much as possible. We now return to an analysis of its meaning in terms of the framework of this paper. In terms of the regulatory cycle, *design science* as defined by March & Smith [20, page 254] and Hevner et al. [13, page 77] is the design and validation of solution proposals to practical problems. This is how I use the term henceforth in this paper.

Hevner et al. suggest that design science differs in two aspects from other branches of science: (a) it is concerned with artifacts rather than facts of nature or social structure, and (b) it is concerned with a search for prescriptive rules for design, rather than a search for descriptions, explanations and predictions, as other branches of science are. This goes back to Simons's remark that natural science is concerned with the way things are but that design science, or the science of design, is concerned with how things ought to be, with devising artifacts to attain goals [34, page 132].

However, property (a) of design science, the study of artifacts rather than of nature, is not a distinguishing feature. Even though it may be possible to distinguish nature from human-made artifacts, in the 20th century industrial-scientific complex it is impossible to distinguish a *science* of nature from a *science* of artifacts [5, 9, 28]. In fact, even in 17th century science it was a matter of debate whether experimental philosophers were studying artifacts such as barometers and air pumps [32] or natural phenomena such as air pressure. Of course the processes in artifacts are natural processes. But even if the distinction between nature and artifacts is maintained, research methods used to *study* nature may be used to study artifacts and vice versa. I conclude that distinction (a) of design science drops out.

Distinction (b), that design science yields prescriptive rules for design, is philosophically suspect: prescriptions do not follow from observations. However, during practical problem solving we need to show the internal validity of a design (questions E1 and E2). The internal validity argument E1 does imply a prescriptive rule, namely

P1 In order to produce E, one can place S in context D.

However, human choice is involved, as there may be other solutions S' that in context D produce the same effects. Most importantly, the rule does not say it is a *good* thing to do. It simply tells us how an effect can be produced. Whether the effect is good depends on stakeholder goals and on the truth of E2. So distinction (b) drops out too. What we are left with is an iteration over designing and validating solution proposals, which is how I view design science.

So what justifies our use of the term "design *science*" rather than "design"? This requires looking again at the genericity of knowledge questions, as we did at the end of section 2.

5.2 Levels of theory

The regulative cycle contains two important kinds of knowledge questions, namely *problem investigation,* in which we investigate currently existing problems, goals, and outcomes, and *design validation,* in which we try to predict future outcomes when a design would be implemented (figure 1).

Van Strien [35] makes a useful distinction between what I here will call *practice-oriented theories*[5] and *N=1 theories.* A practice-oriented theory is a theory of a class of practical problems. Van Strien distinguishes two kinds of practice-oriented theories, diagnostic theories and treatment theories. To bring this terminology in line with what we said earlier about possible triggers of problem investigation (section 3.1), we refine this into four kinds of practice-oriented theory: A *diagnostic theory* [35, page 692] explains the causes of problematic phenomena; A *solution theory*[6] predicts the effect of a solution design, following the schema E1 mentioned earlier; A *goal theory* conceptualizes and operationalizes goals; and an *impact theory* describes and explains the impact that an implemented solution has brought about. Impact theories are made after a design has been implemented and effects have actually occurred in a particular case. These theories may be generalizable to other cases where similar mechanisms occur [27].

In order to apply a practice-oriented theory to the particular, concrete problem at hand, the practitioner needs to deal with the unique features of this problem, and the role of a practice-oriented theory is to help the practitioner to formulate an *N=1* theory of the problem at hand [35, page 692]. For example, a practice-oriented theory of trade-offs between efficiency and flexibility in ERP implementation can help a practitioner to formulate a theory of trade-offs in a particular company.

Practice-oriented theories are contrasted by Van Strien with *nomothetic theories,* in which general laws are formulated. For example, social network analysis can provide a nomothetic theory of alliances and gatekeepers in social networks. A theory of cooperation patterns in global software development would be a practice-oriented theory. A project manager could use such a practice-oriented theory to formulate an N=1 theory about the communication in her project.

[5]Called problem-oriented theories by Van Strien.
[6]Treatment theory in Van Strien.

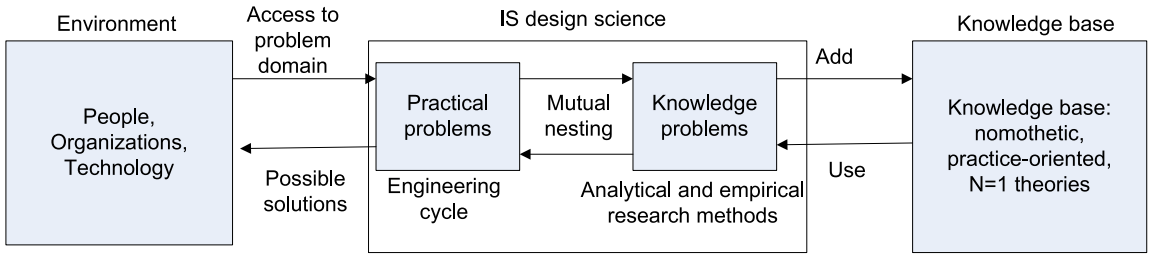


Figure 5: Refinement of the framework of Hevner et al. [13].

This gives us three levels of theory, N=1, practice-oriented, and nomothetic, with increasingly large scopes. The view taken in this paper is that design-science research aims at practice-oriented theories. It aims to produce diagnostic theories of classes of problems, solution theories of classes of solutions, operationalizations of classes of goals, and explanations of impacts. Practice-oriented theories correspond to what Walls et al. [37, page 41] call *design theories*, except that I do not think that they are prescriptive, as Walls et al. say, but that are operationalized in the form E1, as I explained above. A practice-oriented theory may use a nomothetic theory as background knowledge, which then functions as Walls et al.'s *kernel theories*. But in my view this is not a requirement for being a practice-oriented theory.

5.3 Research methods

Above I made the claim that no research design is banned *a priori* from design science research. Methods to investigate nature may be used to investigate artifacts and vice versa. This claim needs to be qualified, because there is a preference for certain kinds of research designs in validation research.

In validation research we ask questions about a thing that does not exist yet: the implementation. Hence we make a prototype or scale model and investigate that. For example, when the jet engine was developed in the 1930s and 1940s, scale models of it were constructed and investigated in the laboratory, and these were increased in size and investigated in increasingly realistic conditions until a test pilot tested one in a plane [6]. Similarly, if one wants to test a new business process modelling technique, never used before, the researcher can (1) set up a simulation with students using the technique in a laboratory experiment, and then scale up by (2) having professionals use it in a field experiment and then include even more conditions of practice by (3) using the technique in a pilot project. These experiments, simulations and pilots are done under increasingly less controlled conditions. In general, design validation researchers have a preference for research methods that involve modelling and simulation, so much so that in some versions of the regulative cycle, validation is called "simulation" [29].

In addition, in validation research there is often a sequence of investigations starting in a lab where conditions of practice are almost all absent, and leading up to field tests where conditions of practice are all present. *Conditions of practice* is a concept introduced in the history of science and technology to indicate the numerous variables that exist in concrete artifacts and that can impact their behavior [18, 19]. Design-science research studies practical problems or their solutions, and therefore cannot abstract from conditions of practice as basic science can [4]. This forces the validation researcher to slowly scale up his or her models to conditions of practice that exist in the field [16]. This explains the preference of engineering researchers for field research methods such as field experiments and case studies in the form of pilot projects to test out a solution under conditions of practice.

Technical design-science research typically proceeds by an endless iteration over solution design and design validation using models, starting in the lab end ending with pilot projects in the field. Such an endless iteration is not possible if the investigated solution is a social system, such as an information system when construed as subsystem of an organization. It simply too expensive, or not moral, or not valid to iterate over successive versions of a solution design involving people as many times and in the same way as this can be done with inanimate material. Field experiments using the intended users of the technique may be too expensive too, and in addition pilot projects involving a technique proposed by design-science researcher may be a bridge too far for the intended users of the technique. An alternative road open to the information systems design-science research is then to use a technical action research (TAR) approach, where the researcher validates a technique by using it herself in a real project under conditions of practice [3].

This discussion indicates a preference for some kinds of research methods in validation research but this does not invalidate the point that no research method is banned from design science *a priori*. In design science, as anywhere else, a research design should be driven by the research questions asked, the properties of the subject that is investigated, and the available resources [40].

6. COMPARISON WITH OTHER WORK

We have now refined the framework of Hevner et al. [13] in three ways (figure 5). First, the regulative cycle elaborates problem investigation to elicit business needs. This aspect does not receive much attention in the framework of Hevner et al. but problem investigation is an important task on the road to possible relevance. Second, the knowledge base is stratified into the levels of N=1 theories, practice-oriented theories and nomothetic theories. Design science is primarily concerned with the intermediate level. Third, IS design-science research is decomposed into a nested problem structure, where the two major problem classes are practical problems and knowledge questions. Knowledge questions may be answered applying knowledge from the knowledge

[G1] Distinguish practical problems from knowledge questions	Section 2. In practical problems stakeholders desire to change the world, in knowledge questions the researcher desires to change his/her knowledge of the world.
[G2] Solve practical problems by the regulative cycle	Figure 1.
[G3] Distinguish problem investigation from design validation	Section 3. In problem investigation existing phenomena are investigated, in design validation the effects of an unimplemented design are predicted
[G4] Problem investigation may be problem-driven, solution-driven, goal-driven, or impact-driven	Section 3.1. In problem investigation, one or more of these tasks needs to be done: diagnose a problem, operationalize goals, check the validity of the design argument, or investigate the impact of realized implementations.
[G5] When designing a solution, maintain the design argument	Section 3.3. The causation part of the design argument says that the solution in a context will have certain effects, the valuation part says that these effects satisfy stakeholder criteria
[G6] When validating a design, consider trade-offs and sensitivity	Section 3.1. In trade-offs we vary the solution, in sensitivity analysis we vary the environment.
[G7] When validating a design, aim to incorporate conditions of practice	Section 5.3. Scale up from controlled conditions to realistic conditions.
[G8] When solving a knowledge question in the regulative cycle by means of research, no research method is banned.	Section 5.3. Research design must be justified, as anywhere else, in terms of research questions, the investigated domain and available resources to do the research.

Figure 6: Guidelines for design science.

base or by doing original research, using conceptual analysis or empirical methods such as experiments, case studies, field research or modelling and simulation. Practical problems are solved by matching problems and solutions in a regulative cycle.

Four of the seven design-science research guidelines of Hevner et al. are supported by the analysis in this paper, the other three are elaborated a bit. The ones that are adapted are dashed: (1) design yields artifacts; (2) problems must be potentially relevant but this is context-dependent, as we saw above; (3) solution proposals must be validated; (4') design research must produce practice-oriented theories; (5) these must be validated using scientific methods; (6') design research iterates over designing and validating, as part of the regulative cycle; (7) research results must be communicated to researchers as well as practitioners.

The proposal in this paper is very close to the proposal by Nunamaker and Chen [24], who present a systems development research process in the form of a regulative cycle. The differences are that in this paper I distinguish practical problems from research questions and that I give problem investigation and problem nesting more attention.

As pointed out above, the design theories of Walls et al. [37] correspond to my practice-oriented theories, which can be solution theories, diagnostic theories, goal theories or impact theories. Like Hevner et al., Walls et al. introduce prescriptive (normative) theories, which I think is a philosophically unsound concept, that is not needed to introduce the concept of design theory. Walls et al. also require a design theory to refer to a kernel theory, corresponding to a nomothetic theory. I do not require nor prohibit that a practice-oriented theory (design theory) refer to a nomothetic theory (kernel theory). Any practice-oriented theory produced by a sound research method is OK in my view. More generally, Walls et al. and also March & Smith [20] provide detailed prescriptions for what a theory should contain; I avoid such prescriptions.

7. CONCLUSIONS

Figure 6 lists the claims of this paper in the form of guidelines for the design science researcher. They are consistent with and can be added to the guidelines provided by Hevner et al. discussed above. Each of the guidelines in figure 6 has been argued in a section of the paper, as indicated in the table. [G6], [G7] and [G8] are additionally justified because they are sound engineering practice; [G2] and [G5] are injunctions to be rational; [G1], [G3] and [G4] are definitions, which I claim help the design scientist become aware of what he or she is doing. Using these guidelines in our own research, such as the thesis of Mutschler discussed in this paper, helps researchers clarify what goals they want to achieve and which methods are appropriate for reaching these goals.

8. ACKNOWLEDGMENTS

I would like to thank Hans Heerkens for the numerous fruitful discussions we have had about the interaction between research and design. Thanks are due to the anonymous reviewers for constructive comments on an earlier version of the paper.

9. REFERENCES

[1] B.L. Archer. The structure of the design process. In G. Broadbent and A. Ward, editors, *Design methods in Architecture*, pages 76–102. Lund Humphries, 1969.

[2] L.B. Archer. Whatever became of design methodology? *Design Studies*, 1(1):17–18, July 1971.

[3] R.L. Baskerville. Distinguishing action research from participative case studies. *Journal of Systems and Information Technology*, 1(1):25–45, March 1997.

[4] M. Boon. How science is applied in technology. *International Studies in the Philosophy of Science*, 20(1):27–47, March 2006.

[5] W.B. Carlson. Innovation and the modern corporation. From heroic invention to industrial science. In J. Krige and D. Pestre, editors, *Companion to Science in the Twentieth Century*, pages 203–226.

Routledge, 2003.

[6] E.W. Constant. *The Origins of the Turbojet Revolution*. Johns Hopkins, 1980.

[7] N. Cross. *Engineering Design Methods: Strategies for Product Design. Second Edition*. Wiley, 1994.

[8] N. Cross. Design cognition: Results from protocol and other empirical studies of design activity. In C. Eastman, M. McCracken, and W. Newstetter, editors, *Design Knowing and Learning: Cognition in Design Education*, pages 79–103. Elsevier, 2001.

[9] M.J. de Vries. *80 Years of Research at the Philips Natuurkundig Laboratorium*. Pallas Publications, 2005.

[10] D.C. Gause and G.M. Weinberg. *Exploring Requirements: Quality Before Design*. Dorset House Publishing, 1989.

[11] C.A. Gunter, E.L. Gunter, M.A. Jackson, and P. Zave. A reference model for requirements and specifications. *IEEE Software*, 17(3):37–43, May/June 2000.

[12] A.D. Hall. *A Methodology for Systems Engineering*. Van Nostrand, 1962.

[13] A.R. Hevner, S.T. March, Park J, and S. Ram. Design science in information system research. *MIS Quarterly*, 28(1):75–105, March 2004.

[14] T.P. Hughes. *Networks of Power. Electrification in Western Society, 1880–1930*. Johns Hopkins university Press, 1983.

[15] J.C. Jones. *Design Methods: Seeds of Human Futures*. Wiley, 1970.

[16] G. Küppers. On the relation between technology and science—goals of knowledge and dynamics of theories. The example of combustion technology, thermodynamics and fluid dynamics. In W. Krohn, E.T. Layton, and P. Weingart, editors, *The Dynamics of Science and Technology. Sociology of the Sciences, II*, pages 113–133. Reidel, 1978.

[17] I. Lakatos. History of science and its rational reconstructions. In *PSA: Proceedings of the Biennial Meeting of the Philosophy of Science Association*, pages 91–136. The University of Chicago press, 1970.

[18] E. Layton. Mirror-image twins: The communities of science and technology in 19th century America. *Technology and Culture*, 12(4):562–580, October 1971.

[19] E.T. Layton. American ideologies of science and engineering. *Technology and Culture*, 17:688–701, 1976.

[20] A.T. March and G.F. Smith. Design and natural science research on information technology. *Decision Support Systems*, 15(4):251–266, December 1995.

[21] J.G. March. *A Primer on Decision-Making*. Free Press, 1994.

[22] B. Mutschler. *Modeling and simulating causal dependencies on process-aware information systems from a cost perspective*. PhD thesis, Univ. of Twente, Enschede, January 2008. `http://eprints.eemcs.utwente.nl/11864/`.

[23] NASA. *Systems Engineering Handbook*, June 1995. SP-610S `http://human.space.edu/old/docs/Systems_Eng_Handbook.pdf`.

[24] J.F. Nunamaker, M. Chen, and T.D.M. Purdin. Systems development in information systems research. *Journal of Management Information Systems*, 7(3):89–106, Winter 1990–1991.

[25] G. Pahl and W. Beitz. *Konstruktionslehre. Handbuch für Studium und Praxis*. Springer, 1986.

[26] D.L. Parnas and P.C. Clements. A rational design process: How and why to fake it. *IEEE Transactions on Software Engineering*, SE-12:251–257, 1986.

[27] R. Pawson and N. Tilley. *Realistic Evaluation*. Sage Publications, 1997.

[28] L.S. Reich. *The making of American industrial research: Science and business at GE and Bell, 1876–1926*. Cambridge University Press, 1985.

[29] N.F.M. Roozenburg and J. Eekels. *Product design: Fundamentals and Methods*. Wiley, 1995.

[30] A. Sayer. *Method in Social Science: A Realist Approach*. Routledge, 1992 (2nd edition).

[31] J.R. Searle. *Speech Acts. An Essay in the Philosophy of Language*. Cambridge University Press, 1969.

[32] S. Shapin and S. Schaffer. *Leviathan and the air pump: Hobbes, Boyle and the experimental life*. Princeton University Press, 1985.

[33] H.A. Simon. A behavioral model of rational choice. *Quarterly Journal of Economics*, 69:99–118, 1955.

[34] H.A. Simon. *The Sciences of the Artificial*. The MIT Press, 1981. Second edition.

[35] P.J. van Strien. Towards a methodology of psychological practice: The regulative cycle. *Theory & Psychology*, 7(5):683–700, 1997.

[36] L. Suchman and E. Wynn. Procedures and problems in the office. *Office: Technology and People*, 2:135–154, 1984.

[37] J.G. Walls, G.R. Widmeyer, and O.A. El Sawy. Building an information system design theory for vigilant EIS. *Information Systems Research*, 3(1):36–59, 1992.

[38] L. White. Pumps and pendula: Galileo and technology. In C.L. Golino, editor, *Galileo Reappraised*, pages 96–110. University of California Press, 1966.

[39] R. Wieringa, N. Maiden, N. Mead, and C. Rolland. Requirements engineering paper classification and evaluation criteria: A proposal and a discussion. *Requirements Engineering*, 11(1):102–107, March 2006.

[40] R. J. Wieringa and J. M. G. Heerkens. Design science, engineering science and requirements engineering. In *16th IEEE International Requirements Engineering Conference, Barcelona, Spain*, pages 310–313, Los Alamitos, 2008. IEEE Computer Society Press.

[41] R.J. Wieringa. *Requirements Engineering: Frameworks for Understanding*. Wiley, 1996. Also available at `http://www.cs.utwente/nl/~roelw/REFU/all.pdf`.

[42] R.J. Wieringa and J.M.G. Heerkens. The methodological soundness of requirements engineering papers: A conceptual framework and two case studies. *Requirements Engineering Journal*, 11(4):295–307, 2006.