Design Science Research as Movement Between Individual and Generic Situation-Problem–Solution Spaces

Ilia Bider, Paul Johannesson and Erik Perjons

Abstract Design science is an emerging research paradigm in the Information Systems area. A design science project typically includes the activities of problem analysis, requirements definition, artifact development, and evaluation. These activities are not to be seen as sequential but can be carried out in any order. The purpose of this paper is to propose a conceptualization and formalization of design science research that show the possible ways in which a design science project can be carried out. The proposal is based on the state oriented view on business processes and suggests that design science research can be viewed as movements in a space of situations, problems and solutions.

Keywords Design science · Information systems · Business process

1 Introduction

1.1 Motivation

Design science is an emerging research paradigm in the Information Systems (IS) area. To make design science more widely accessible, there is a need for clear

I. Bider · P. Johannesson (✉) · E. Perjons
Department of Computer and Systems Sciences (DSV), Stockholm University, Stockholm, Sweden
e-mail: pajo@dsv.su.se

I. Bider
e-mail: ilia@dsv.su.se

E. Perjons
e-mail: perjons@dsv.su.se

R. Baskerville et al. (eds.), *Designing Organizational Systems*,
Lecture Notes in Information Systems and Organisation 1,
DOI: 10.1007/978-3-642-33371-2_3,

guidelines on how to use it, which are targeted at students and young researchers as well as experienced researchers accustomed to other ways of conducting research. One of the main issues that research students need to understand is that a research process can be conducted in many alternative ways while still staying within the design science paradigm.

The main way of presenting results of research of the design science type is starting from a situation, presenting the identified problem, setting the requirements, describing the solution, presenting the results of tests whether the solution solves the problem in the given situation, or/and a set of similar situations [1]. If the test shows that the solution does not solve the problem, it is redesigned, and tested again. Even when it is not explicitly stated that this sequence was the actual order in which the research was conducted, this kind of presentation may create an impression that this is how the research actually was done in real-life, which is often far from the case.

One problem to be solved when creating guidelines for conducting design science research is how to determine where in the process the researcher is at a given moment, and what are the alternatives to proceed. The goal of this chapter is to propose a solution to this problem. More exactly, we propose a kind of a map which helps the researcher to identify his/her position in the design science process, and explicitly shows the possible ways that are open for the researcher to continue. This "map" is aimed to serve as a foundation for creating practical guidelines on design science research in the future.

1.2 Design Science in Information Systems Research

Design science research [1–3] is related to finding new solutions for problems known or unknown [4]. To count as related to research, a solution should be of a generic nature, i.e., applicable not only to one unique situation, but to a class of similar situations. A generic solution for real-life problems cannot be proven in a formal way, but requires testing via implementation of the solution in one or more situations and investigation whether it solves the intended problem or not. The solidness of the verification, of course, depends on the number of implementations, and the diversity of situations in which the solution has been tested. However, the first step always is to implement and verify a generic solution in at least one situation. In the literature, this is called demonstration [1].

There is a substantial difference between design science on one hand and wide spread qualitative and quantitative empirical methods [5] on the other. The latter are aimed to investigate real life situations as is, or as they were at some point in the past, and find commonalities between them which can give rise to a theory that explains the current or past state of affairs. Focusing on the present and past also allows employing statistical methods as there is a possibility to gather information on many similar real-life situations, insuring that the size of a sample is sufficiently large for relying on statistical methods.

Focusing on the present and past in such a dynamic area as IS has a major drawback. Such a focus means that research follows the industry/practice and explains its successes and failures rather than showing new ways to proceed. Design science research with its focus on generic problem solving tries to overcome this drawback. This kind of research can be considered as an activity aimed at generating and testing hypotheses about future solutions. Therefore, demonstration is a critical part of design science, as it shows whether a hypothesis is worth adoption or needs to be discarded or improved.

Design science research, on its own, cannot provide sufficient evidence in favor of a hypothesis. It can only demonstrate that it could work in one or several specific situations. The real proof comes when and if the industry/practice adopts the solution, which generates sufficiently many examples of its usage in real life, so that standard empirical methods can be employed to prove or disprove the hypothesis generated by design science. Therefore, design science research cannot be placed in the same category as standard empirical methods, but should be regarded as complimentary to them.

Summarizing the above deliberations, we agree with [2] that design science represents a distinct research paradigm—generating hypotheses on how the future could look like and making initial filtering of them in order to remove hypotheses not worth of pursuing. IS is not the only field within the social sciences where design science research is applicable. Marxist theory, when it first emerged, can be said to have followed the design science approach by proposing a solution for all the evils of capitalism, though it failed to produce even one demonstration where this solution would work. The difference of IS from other fields in the social sciences is the greater dynamics of the IS industry/practice, where new solutions are introduced and tested all the time. This gives a greater opportunity for design science research to take a leading position in this field.

1.3 Artifacts in IS Design Science Research

As was mentioned in the previous subsection, design science concerns generic solutions, not the individual ones. A generic solution is always a template that can be adjusted to a specific situation to which the solution is to be applied. This template can be defined in different ways, e.g., as a model, or as a method that shows how to transform the original situation to a new and better one. In design science literature such a template is often referred to as to an (scientific) artifact. We call this artifact a primary artifact.

In addition to the primary artifacts, design science research in IS may include secondary artifacts inherent to the field of Information Systems. In essence, IS investigates technology enabled human organizations and practices, to which nowadays belong any private company, public administration, interest organization, or a bunch of youngsters playing an interactive game with each other. Thus, IS is tightly connected to the technological artifacts (e.g., an ERP system or an e-

service) used to tie together technologically enabled organizations and practices. IS deals with both development and use of such technological artifacts. A generic solution in IS is often built around some technological artifact, e.g., the solution includes the development and/or introduction in organizational practice of some type of an IT-system. We refer to such technological artifacts as secondary artifacts.

In this paper, we do not investigate various types of primary and secondary artifacts. Instead, the discussion is conducted in terms of individual and generic solutions, while artifacts are referred to only in the explanatory examples. An individual solution concerns a specific situation which disappears as soon as an attempt to solve it has been made, and, normally, cannot be restored. A generic solution can be applied to many real-life situations, and thus can be tested more than once.

1.4 Moving Between the Two Worlds

As explained in Sect. 1.2 , design science research aims at generating and testing generic solutions for various problems. To do that, a researcher needs to move in and between two worlds: the world of real situations and real solutions, and the world of generic (abstract) situations and solutions. The movements in these two worlds cannot be totally independent of each other; movement in one world needs to be coordinated with movement in the other world.

There is no fixed way in which the movement inside and between the two worlds should be accomplished. One can start with designing a new solution for a known problem and then design a generic solution based on it, using the individual solution as an example. One can also start from the opposite pole—design a generic solution for the problem unknown and search for a situation where such generic solution can be applied to transform this situation to a better one.

The goal of this paper is not to prescribe the way to conduct research, but define a "map" that includes both worlds. The map is aimed at helping the researcher to plan his/her movements inside and between the worlds, and find a way out when a kind of dead end is reached. For example, if a generic solution does not work very well in a situation where it should, the researcher does not need to rush to discard the solution, or try to modify it. He/she can instead try to find some other type of situations, or other problems where the solution fits better.

The rest of the paper is structured in the following way. In Sect. 2, we introduce a number of quasi-formal notions, with the help of which we can define the meaning of design science research and the ways it could be conducted. In Sect. 3, we discuss in which circumstances individual problem solving gives opportunities for design science research. In Sect. 4, we discuss examples of a research that aims at finding a generic solution from the beginning. Section 4.1 discusses a case where a researcher moves to the world of generic situation/solutions in order to find a solution for a problem in a particular real-life situation. The individual

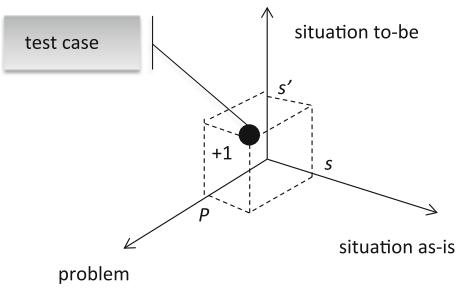

Fig. 1 Individual SPS-space

solution is then obtained by instantiation of the newly designed generic solution. In Sect. 4.2, we discuss a case where research starts from finding a generic solution and proceeds to finding a situation where it could be applied. Section 5 reviews related works. Section 6 is devoted to the discussion of the material presented. Section 7 presents plans for future work.

2 Formalizing Design Science Research

In this section we present a set of notions that leads to a quasi-formal definition of design science research. These notions will be used in the next sections to explain the ways of how design science research could be conducted. The formalization is done along the following lines. As the first step, we introduce a space for individual (specific) problem solving as consisting of three dimensions: *situation as-is*, *situation to-be*, and *problem*, see Fig. 1. A point $< s, s', P >$ in this space is called a test case. A weight of $\pm$ 1 is assigned to a test case according to the following rule:

- +1 is assigned if transforming situation s into s' solves problem P. In this case s' is said to be a solution for problem P in situation s.
- -1 is assigned otherwise.

As the next step, we introduce a space for generic problem solving as consisting of three dimensions: *generic situation as-is*, *generic situation to-be*, and *generic problem*, see Fig. 2. We consider that a generic situation (as-is or to-be) is represented by some kind of a template (t or t') that defines a set of similar situations, the template serving as an extension of this set. A point $< t, t', GP >$ in this space is called a hypothesis.

As the third step we establish correspondence between the two spaces based on the following rule test case $< s, s', P >$ serves as representation (or instantiation) of hypothesis $< t, t', GP >$ if situation as-is s belongs to the generic situation as-is defined by template t, situation to-be s' belongs to the generic situation to-be defined by template t', and problem P belongs to the generic problem GP, see Fig. 3.

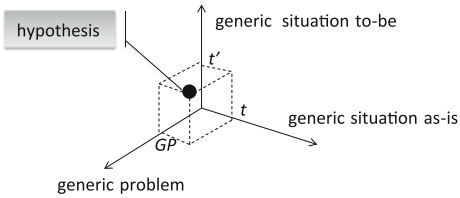

Fig. 2 Generic SPS-space

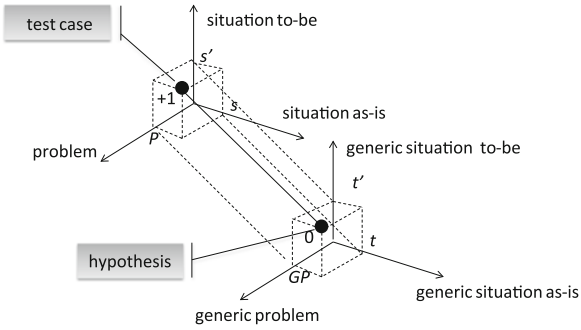

Fig. 3 Correspondence between the spaces

Based on the correspondence between the two spaces, we can assign a weight to a hypothesis according to the following rule:

- -1, if there is a test case $< s, s', P >$ with weight -1 that represents the given hypothesis $< t, t', GP >$. In this case, we say that t' cannot serve as a generic solution for generic problem GP in generic situation t.
- 0 if there is one or few test cases $< s, s', P >$ with weight +1 that represent the given hypothesis $< t, t', GP >$. In this case, we call t' a potential generic solution for generic problem GP in generic situation t.
- +1 if there are many test cases $< s, s', P >$ with weight +1 that represent the given hypothesis $< t, t', GP >$. In this case, we call t' a verified solution for generic problem GP in generic situation t.

In the following Sects. 2.1–2.3, we will explain the three steps above in more details.

2.1 Individual Problem Solving Space

We define a *situation* as a "state of affairs or combination of circumstances" in a particular part of the world. A situation has some structure, but we are not interested in details of it in this paper. As an example of a situation consider the following:

"activities at the sales office of SoftMotors in Stockholm", where SoftMotors is an imaginary company.

Denote the set of all possible situations s as S. S includes all situations that are in effect now, the ones that happened in the past, and the ones that could be imagined in the future.

Transformation of situation s (situation as-is) into situation s' (situation to-be) is called *implementation* of s' in s, and is denoted $s' \leftarrow s$.

As soon as the transformation happens, the original situation becomes part of the past, and the new one changes the status of being imaginary (possible) to becoming part of the present. We consider that this transformation is, normally, not reversible, i.e., the old situation cannot be restored, and thus no other transformation can be applied to it. For example, if SoftMotors fires all their sales staff and hire engineers to conduct sales activities instead, we consider the old situation as not reachable. Even if the old staff is rehired, probably, their trust in SoftMotors management will not remain the same.

Completing implementation is associated with *costs* in effort and time. The closer situation s' is to s, the smaller is the cost of transformation. It is impossible to define the concept of closeness in strict terms for a general case. In this paper, we just consider that the more s and s' share the same elements and internal structure, the closer they are, which potentially leads to lower costs of implementing s' in s. For example, the result of introducing a CRM (Customer Relationship Management) system at the sales office of SoftMotors (without changing its staff) can be considered as relatively close to the original situation in which there was no CRM system in place.

We also consider that there is some limitation on the "length" of a possible transformation). For example, we may consider that a situation in which SoftMotors sales office changes their sales people to engineers as reachable, but the sales office becoming the engineering department as unreachable. Note, that a situation not reachable in one transformation may be reachable is several consecutive transformations over time. Denote the set of situations reachable from s as $R(s)$.

A *problem* P is a set of situations ($P \subset S$) considered to be undesirable from some point of view. As an example, suppose that in the sales office of SoftMotors, each sales person works on his/her own and never coordinates his/her work with the colleagues. Management considers this situation as a problem, as prospects could be lost when a sales person pursuing it becomes temporarily (sick) or permanently (leaves the company) unavailable. Formally, this problem can be defined as a set of situations in which

the progress of pursuing a prospect in the sales office of SoftMotors in Stockholm is visible only to one sales person.

Let P be a problem, then for each situation s that belongs to P ($s \in P$) we say that situation s has problem P, or there is a problem P in situation s.[1] Denote the set of all possible problems P as Π .[2]

A *solution* for a problem P in situation s ($s \in P$) is a situation s' such that s' does not have problem P, i.e. $s' \notin P$. An example of a solution to the problem above is a situation in which:

> all activities related to pursuing a prospect in the sales office of SoftMotors in Stockholm are registered in CRM system *Oracle on demands* and are made visible to all members of the staff

We call a solution s' *justifiable* if the cost of the implementation is reasonable with respect to the seriousness of the problem P in s that the solution is aimed to solve.[3]

We call a solution s' *acceptable* if it does not contain any other major problem that is not present in s. For example, the CRM system to be implemented is so difficult to use that all current sales staff will not be able to handle it, and everybody leaves.

Testing s' as an acceptable solution for a problem P in s means implementing s' in s and verifying that:

- s' is justifiable for P in s,
- s' is really a solution for s, i.e., $s' \notin P$, and
- s' is acceptable, i.e., does not include other major problems.

A space with dimensions S (situation-as-is), S (situation-to-be), Π (problem) of the kind in Fig. 1 is called a Situation-Problem–Solution space, or *SPS-space* for short. A point $< s, s', P >$ in the SPS-space is called a *test case*. Testing s' as a solution for problem P in situation s is considered as assigning a weight to the test case $< s, s', P >$ in the SPS-space according to the following rule:

- Weight +1 is assigned in the case s' showed to be *justifiable* and *acceptable* solution for problem P in situation s,
- Weight -1 is assigned otherwise.

[1] The problem can be defined in very broad terms, however, for all practical purposes only the intersection between the problem P and a set of situations reachable from s, $R(s)$ is important for any given situation s.

[2] $\Pi \subset P(S)$, where $P(S)$ is the set of all subsets of set S.

[3] Obviously, the first sign of "justifiable" is being "reachable", i.e. $s' \in R(s)$.

2.2 Generic Problem Solving Space

A set of situations with similar internal structure is called a *generic situation* or *g-situation* for short, and is denoted as *GS*. As an example of g-situation, consider "Activities at sales offices of all companies in Stockholm", or "Activities at sales offices of all companies all over the world". The set of all possible g-situations is denoted as Σ.

As a criterion of similarity for situations in a g-situation, we consider existence of a *pattern/template t* that can be found in the internal structure of each situation that belongs to the given g-situation. Denote all possible templates as *T*. We consider a pattern/template to be a sentence in some formal language that includes variables and constants. Continuing the example from Sect. 2.1, we can create several templates of situations:

1. Activities at sales office of company x located at Stockholm
2. Activities at sales office of company x located at y
3. Activities at sales office of company x located at y such that the progress of pursuing a prospect is visible only to one sales person
4. Activities at sales office of company x located at y not using any CRM system
5. Activities at sales office of company x located at y using *CRM on demand* from Oracle to make the progress of pursuing a prospect visible to all members of staff
6. Activities at sales office of company x located at y using CRM system from Salesforce to make the progress of pursuing a prospect visible to all members of staff
7. Activities at sales office of company x located at y using CRM system c to make the progress of pursuing a prospect visible to all members of staff

We consider that all templates are constructed from a universal set of variables *V*. For example, in template (1–8) above, variable x stays for a company.[4]

Assigning each variable $v \in V$ a value is called a mapping. Denote a set of all possible mappings as *M*. Let $m \in M$ be a mapping and $t \in T$ a template, then situation s obtained from t by substituting variables in t with values according to m is called instantiation of t via m, and is denoted as $s = t(m)$.

The example considered in Sect. 2.1 is an instantiation of template (2) above, where map $m = \{x = SoftMotors, y = Stockholm\}$.

A class of situations defined by template t is denoted as $GS(t)$, $GS(t) = \{s = t(m) \mid m \in M\}$; in other words, $GS(t)$ includes all possible instantiations of template t.

A pair of templates t', t defines a class of transformation $\{s' \leftarrow s \mid s' = t'(m), s = t(m), m \in M\}$. We will call such class a generic transformation, or

[4] Existence of a template presumes existence of an algorithm that generates all situations that belong to a given template t by assigning values to the variables of the template.

g-transformation for short, and denote it as $t' \leftarrow t$. A g-transformation $t' \leftarrow t$ defines a transformation for each situation s from $GS(t)$ into a situation s' from $GS(t')$.

Using templates (1–7) above, we can define the following generic transformations:

(5) ← (4), (6) ← (4), (7) ← (4), (6) ← (5).

The first three transformations, (5) ← (4), (6) ← (4), (7) ← (4), refer to introducing a CRM system where it did not exist, the last one, (6) ← (5), refers to changing the existing CRM system from Oracle to Salesforce.

A problem that exists in all instances of some generic situation is called a generic problem and is denoted as *GP*. Formally, we can define a generic problem as follows: let $GS(t) \in \Sigma$ be a g-situation and $GP \in \Pi$ be a problem such that $GS(t) \subseteq GP$, then we say that g-situation $GS(t)$ has (generic) problem *GP*, or there is a (generic) problem *GP* in g-situation $GS(t)$.[5]

A *generic solution* or *g-solution* for short for a (generic) problem *GP* in g-situation $GS(t)$ ($GS(t) \subset GP$) defined by template t is a template t', such that $GS(t') \cap GP = \varnothing$. In other words, all situations that belong to $GS(t')$ do not have problem *GP*.

For example, templates (5), (6), (7) represent generic solutions for the generic problem defined by template (3) in situation (4).

A g-solution $GS(t')$ is called *justifiable* for $GS(t)$ if for each $m \in M$, $t'(m)$ is justifiable for situation $t(m)$. In the same way, a g-solution $GS(t')$ is called *acceptable* if for each $m \in M$, $t'(m)$ is acceptable solution for problem *GP* in situation $t(m)$.[6]

A space with dimensions T (template for g-situation as-is), T (template for g-situation to-be), Π (generic problem) of the kind in Fig. 2 is called a generic Situation-Problem–Solution space, or generic *SPS-space* for short. A point $< t, t', GP >$ in the generic SPS-space is called a *hypothesis*.

We say that test case $< s, s', P >$ corresponds to the hypothesis $< t, t', GP >$ if there exists a mapping $m \in M$ such that $s = t(\mathrm{m})$ and $s' = t'(\mathrm{m})$ and $R(s) \cap P = R(s) \cap GP$, which is illustrated in Fig. 3. We also say that test case $< s, s', P >$ represents hypothesis $< t, t', GP >$, or hypothesis $< t, t', GP >$ generalizes test case $< s, s', P >$, and denote this fact as $< s, s', P > \Leftrightarrow < t, t', GP >$.

Let $< s, s', P >$ be a test case that represents hypothesis $< t, t', GP >$, i.e., $< s, s', P > \Leftrightarrow < t, t', GP >$, to which weight +1 has been assigned. Then we call $< s, s', P >$ a *demonstration* for template t' to be an acceptable g-solution for generic problem *GP* in g-situation $GS(t)$.

Let $< t, t', GP >$ be a hypothesis, and r be a representative sets of corresponding test cases such that $r = \{< s, s', P > \mid < s, s', P > \Leftrightarrow < t, t', GP >$, and

[5] Using templates, $GS(t)$ has a problem *GP* if for each $m \in M$, $t(m) \in GP$.

[6] Note that our notion of solution template roughly corresponds to the idea of artifact widely used in the design science literature [2, 3].

for each pair $< s_1, s'_1, p_1 >, < s_2, s'_2, p_2 > \in r, s_1 \neq s_2\}$ and to each test case $< s, s', P > \in r$ weight +1 has been assigned. Then we call r a *verification* for t' being an acceptable g-solution for generic problem GP in g-situation $GS(t)$. We will not try to clarify the meaning of r*epresentative* set, just assuming that the larger is the set the more solid is verification.[7]

Now, we can assign weights to the hypothesis $< t, t', GP >$ from the generic SPS-space. This is done according to the following rule:

- Positive weight of +1 is assigned if there is a *verification* for template t' to be an acceptable g-solution for problem GP in g-situation $GS(t)$.
- Negative weight of -1 is assigned if there is a corresponding test case in the individual SPS-space $< s, s', P > \Leftrightarrow < t, t', GP >$ with the weight -1 assigned to it. As was defined in Sect. 2.1, negative weight means that solution s' has been tried for P in s and showed to be no solution at all, or an unacceptable solution.
- Weight 0 is assigned if there is at least one demonstration for $< t, t', GP >$ (and no negative examples have been found so far).

2.3 Definition of Design Science Research

In terms of the previous subsections *design science research* can be defined as a set of activities aimed at choosing a hypothesis in the generic SPS-space and assigning a weight to it, if it has not been assigned yet, or changing the weight already assigned to it. In other words, design science research deals with finding, demonstrating and verifying generic solutions. Assignments can be:

- From *nothing* to 0—demonstrating that a new g-solution can work,
- From 0 to +1—proving that g-solution should work for all situations described by the *as-is* template,
- From nothing, 0, or +1 to -1—finding that the generic solution does not work for some situations described by *as-is* template.

As we see from the list, demonstration and verification are essential parts of the design science research, which means that the research activities are not concentrated only in the generic SPS-space, but researchers also needs to move between generic and individual SPS-spaces. Here, we need to add one reservation. As was already mentioned in Sect. 2.3, it is not feasible for a design science

[7] Note that we can consider any individual situations s as a generic one GS, where set GS consists of one element only, $GS = \{s\}$. In this case, the template for GS would not contain any variables. Such interpretation could reduce the number of SPS-spaces to consider from 2 to 1. Though this solution could be preferable from the pure mathematical point of view, we will not pursue it in this paper, as we consider the two spaces construction as being easier to explain and visualized.

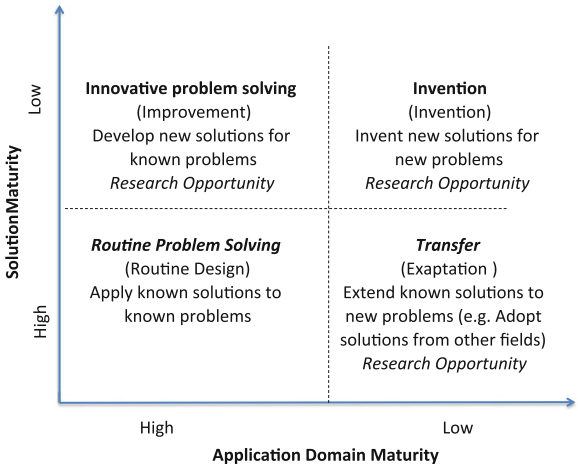

Fig. 4 The application domain maturity/solution maturity matrix adopted from [4]

researcher to test a generic solution in a number of cases sufficient to verify it. To provide conditions for sufficient verification, industry/practice should adopt the solution, so that there will be many cases available for investigation. Investigating these cases, carried out without the involvement of the original design science researchers, falls into the scope of empirical research (using qualitative and quantitative methods). It is this kind of research that can provide final verification of a generic solution.

3 Starting from Individual Problem Solving

Though not equal to individual problem solving, design science research can be initiated by finding a solution for a specific problem in a specific situation. Cases of problem solving that offers opportunities for research are discussed in [4], and are visualized in the Application Domain Maturity/Solution Maturity matrix. Below, we formalize the findings from [4] by introducing four different types of individual problem solving. The matrix, adapted to our terminology, is presented in Fig. 4. The main difference between our matrix and the original one from [4] is the names of the quadrants, the original names from [4] are presented in parenthesis in Fig. 4.

1. *Routine problem solving* (see Fig. 5)—finding a solution to a problem P in situation s via finding a corresponding hypothesis $< t, t', GP >$ in the generic SPS-space with weight +1 (verified generic solution) assigned to it and using this hypothesis for designing a solution for situation s.

Example: if we consider that introducing a CRM system is a verified generic solution for the problem of non-cooperation between sales staff, then introducing

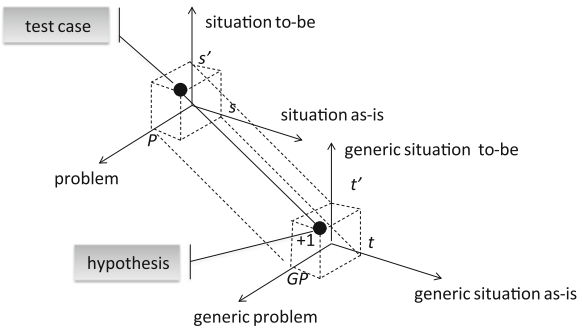

Fig. 5 Routine problem solving

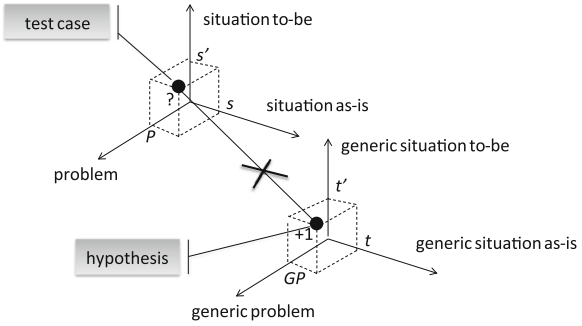

Fig. 6 Innovative problem solving

such a system in the sales office of SoftMotors in Stockholm belongs to the routine problem solving.

Formally, routine problem solving means finding $< t, t', GP >$ with weight +1 such that $R(s) \cap P = R(s) \cap GP$ and there is a mapping $m \in M$ for which $s = t(m)$, and using $t'(m)$ as a solution for problem P in s.

Routine problem solving in its pure form offers no opportunity for design science research.

2. *Innovative problem solving* (see Fig. 6)—finding a solution s' for a problem P in situation s that cannot be derived from an already known g-solution. Innovative problem solving concerns finding a new (or the first) solution for a known problem.

Example: Designing and introducing a system to support sales activities at SoftMotors in Stockholm that is not similar to any known sales support systems constitutes innovative problem solving. Here, we assume that the system is aimed to solve the known problem: "the progress of pursuing a prospect is visible only to one sales person".

Formally, innovative problem solving can be defined as finding a solution such that test case $< s, s', P >$ does not correspond to any known hypothesis $< t, t', GP >$ in generic SPS-space to which a weight of 0 or +1 has been assigned.

Innovative problem solving presents an opportunity for researchers to find a hypothesis $< t, t', GP >$ for which $< s, s', P >$ is a representation. If found, $< t, t', GP >$ gets the weight of 0, test case $< s, s', P >$ being regarded as a demonstration for it.

3. *Transfer*—testing a generic solution designed for another problem or the same problem in different class of situations, or both.

Example: Successful introduction of some Wiki software, developed to solve other problems, to support sales activities at SoftMotors in Stockholm, could be considered as an example of transfer.

Formally, transfer means applying hypothesis $< t, t', GP >$ (with weight of +1) to a test case $< s, s', P >$ such that $GP \cap R(s) \neq P \cap R(s)$ (different problem in relation to the current situation), or $s \notin GS(t)$ (different original situation) or both.

Problem solving of type transfer presents an opportunity for researchers to find a new hypothesis $< t_1, t', GP_1 >$ for which $< s, s', P >$ is a representation. If found, $< t_1, t', GP_1 >$ gets the weight of 0, test case $< s, s', P >$ being regarded as a demonstration for it.

4. *Invention*—testing a transformation to a new situation s' without clearly defined problem in original situation s to solve. This is a process of discovering new opportunities. After transformation has been completed one can discover which problem P has been solved by comparing situations s and s'.

Example: We can use the same example as for *Innovative problem solving* above, but assume that the sales office of SoftMotors in Stockholm functions very well, and there are no particular problems to solve. In this case, designing and introducing a system to support sales activities that is not similar to any known sales support systems constitutes an invention. The difference between the two cases is as follows. Innovative problem solving will require validating that the problem has been solved, while invention will require investigation of which previously unidentified problems have been solved.

Invention presents an opportunity for researchers to find a new hypothesis $< t, t', GP >$ in the generic SPS-space for which $< s, s', P >$ is a representation. If found, $< t, t', GP >$ gets the weight of 0, test case $< s, s', P >$ in the individual SPS-space being regarded as a demonstration.

Classifying problem solving in four types, as above, constitutes some idealization of reality. In practice, some mixture of these types is more probable than a pure adherence to one of them. For example, while following the route of routine problem solving, one can slightly change the recommended solution making it a bit innovative. Another example, after introducing a routine solution, one can discover that besides the original problem, this solution has solved another problem, therefore making the process a bit of invention.

4 Generic Problem Solving

Any of the of individual problem solving types presented in the previous section, except the routine one, can serve as a starting point for initiating design science research via generalizing a newfound solution and finding a point in the generic SPS-space that this solution represents. However, starting from the individual problem solving is not the only way of conducting design science research. The movement between individual and generic SPS-spaces can be done in many different ways. In this section, we present examples of generic problem solving from our practice. Note that these examples do not cover all alternatives of generic problem solving. Section 4.1 discusses a case where a researcher moves to the world of generic situation/solutions in order to find a solution for a problem in a particular real-life situation. The individual solution is then obtained by instantiation of the newly designed generic solution. In Sect. 4.2, we discuss a case where research starts from finding a generic solution and proceeds to finding a situation where it could be applied.

4.1 Innovative Generic Problem Solving

Term "innovative generic problem solving", or IG problem solving for short, is built in association with innovative (individual) problem solving introduced in the previous section. As the name implies, IG problem solving is aimed at finding an innovative generic solution for a known generic problem. A generic problem can be discovered in two ways, either:

1. An individual problem has been discovered in a real-life situation and has been generalized before an attempt to solve it has been undertaken.
2. Previous research has established the fact of a problem existence via employing qualitative or/and quantitative empirical methods.

It can also be both, first a researcher identifies a specific problem in a specific situation and then discovers that the generic problem related to it is already known.

Figure 7 illustrates steps in innovative generic problem solving. If research is initiated by *discovering an individual problem* (case 1 in the list above), the following seven steps could be applied:

1. Establish (refine) the problem P for the given situation s (arrow 1 in Fig. 7)
2. Generalize the situation by creating a template t for which s is instantiation (arrow 2a), and defining a generic problem GP that includes P (arrow 2b)
3. Design a solution template t' (arrow 3)
4. Instantiate a solution to a situation to-be s' (arrow 4)
5. Implement s' in s
6. Evaluate the result and assign a weight to test case $< s, s', P >$ in the individual SPS-space

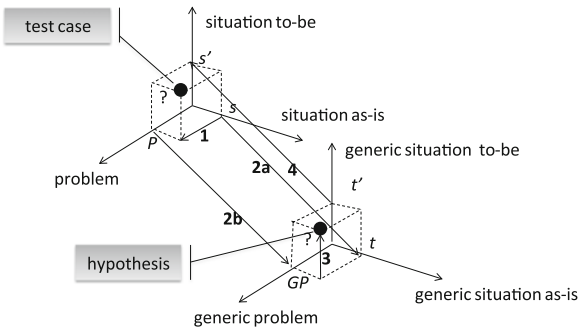

Fig. 7 Generic innovation

7. Promote the result to the hypothesis $< t, t', GP >$ in the generic SPS-space. If $< s, s', P >$ is assigned weight +1 then $< t, t', GP >$ gets weight 0 (see Fig. 3), otherwise −1.

If research start with a *known generic problem* (case 2 in the list presented in the beginning of this subsection), the process starts with step 3 (see Fig. 7), and will require additional step between steps 3 and 4 of finding a situation s to which a solution can be applied. This new step may also be required in a case when, due to some practical reasons, the generic solution cannot be tested in the original situation from which the search for a solution has been initiated. For example, the situation may no longer exist by the time a generic solution has been found.

To illustrate IG problem solving, we will shortly overview a case from our own practice. In 2003–2006, we were engaged in a research project, called INKA [6] aimed at investigating effects from the introduction of an integrated business process support and knowledge management system into operational practice of a non-profit interest organization. Soon after the first version of the system had been developed and put into operation, we discovered that very few used it, which made it impossible to investigate any effects. An investigation was conducted on the causes of the failure, which resulted in critique of the usability of the system, i.e., the system design was not sufficiently intuitive and user-friendly.

After this discovery, the user interface of the system was totally redesigned [7] and the users were once more invited to use the system. The result was negative this time as well, i.e., very few used the system. However, nobody was criticizing the system design any longer. The situation was plainly explained by a statement from one of the supposed users: “I am sure that the system is very good, but I do not know what I should use it for”. As nobody was “blaming” the system, the attention of our research group was moved to search for the reasons of failure elsewhere.

As the research focus in the project was not in the introduction process itself, we tried to apply routine problem solving first. However, our search for a generic solution in the literature gave no result. Solutions recommended for conducting an organizational change where related to planning it in the right way from the very beginning. This was not the case in our project. We were in the middle of not so

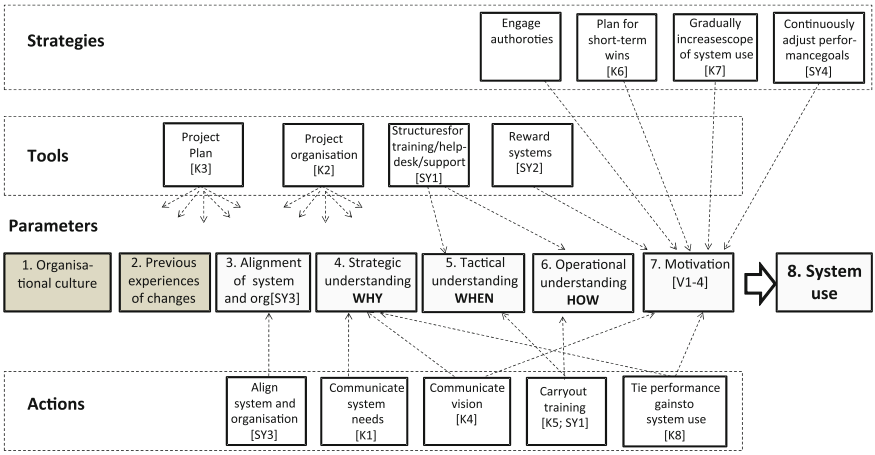

Fig. 8 Element of A^3

successful system introduction that we could not "rewind". The only way to proceed for us was to invent a solution ourselves.

A solution was designed as a general methodology for conducting a system introduction process that was possible to apply even in the middle of unsuccessful introduction [8] (In Swedish). The methodology was called A^3—Assess-Adjust-Apply. The main idea behind A^3 is a number of measurable parameters that can be assessed at any point of introduction process, and a number of means of control that can be employed for influencing the values of these parameters, see Fig. 8. The means are grouped into three categories: actions, tools and strategies. The process is driven by periodic assessment of parameters and choosing means of controls to proceed based on the values of these parameters.

A^3 was tested in the original situation with some success. We were able to higher the usage of the system to some degree. However, due to the reason outside our control, i.e. changes in the management of the interest organization with which we worked, we could not fully complete our testing.

Summarizing the above, we started with a situation with a problem—introduction process that "stammered". We generalized the problem and devised a generic solution for it—a way of conducting the introduction process that drives the system usage up. We (partly) applied this generic solution to the original situation. Right now, we are still looking for new situations where A^3 can be applied to subject our generic solution to additional tests.

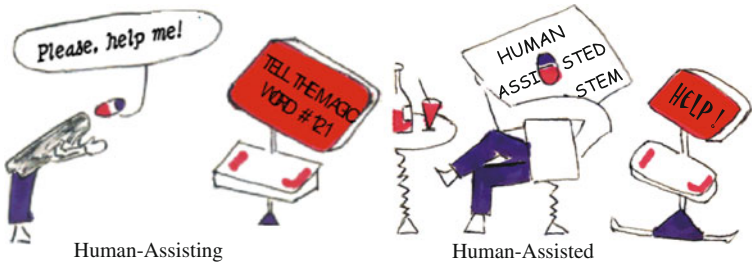

Fig. 9 Human-assisting vs Human-assisted systems

4.2 Generic Invention

We consider generic invention as a generalization of invention in the individual SPS-space described in Sect. 3, which amounts to testing a new solution in a given situation without having any specific problem to solve. Accordingly, generic invention means finding a generic solution for an unknown generic problem. What is more, generic invention can start without full understanding of in which situations the solution is to be applied, which should be found at a later stage. To give an example of a generic invention, we overview a case from our own practice described in details in [9].

In the middle of 1980th one of the authors with a number of collaborators was engaged in a research project, called CHAOS, aimed at developing a theoretical framework for designing of what we then called "human-assisted systems". The idea was explained with two pictures presented in Fig. 9, one representing traditional at the time human-assisting systems (left part of Fig. 9), the other one a new paradigm of "human-assisted systems (right part of Fig. 9). The project acronym CHAOS stood for Concurrent Human-Assisted Object Systems.

In a human-assisting system, a computer helps a human being to perform certain tasks, e.g., to write a letter, print an invoice, complete a transaction, etc. The relations between these tasks, and the aim of the whole process, are beyond the understanding of the computer, but are a prerogative of the human participant. In a human-assisted system, the roles are somewhat reversed, the computer has some knowledge about the process and keeps it running as long as it can. When the system cannot perform a task on its own or figure out what to do next, it will ask the human participant for assistance. The human-assisted system frees human beings from tedious, routing work, like searching for information, bookkeeping, reporting, allowing them to concentrate on things at which they are best, i.e. decision making.

The main idea of human-assisted systems was that the users and the system should work in a symbiosis. The symbiosis should be flexible which means such cooperation between the system and its users where the distribution of responsibilities between them may change in time. It means that the points of interaction between the system and its users may change, thus we need to have a model in

which such changes do not require substantial modifications. The latter requires having a model in which both human and system actions are represented uniformly on equal footing.

The project was of theoretical nature. We had one example in mind though, creating a computer programmer's secretary, a system that would help a programmer to managed his/her job, i.e., to ensure that the programmer does not forget to compile and test after making changes in the source code. The "secretary" should considerably extend the capability of the tools like make/build that existed at the time. The project continued for 2 years, from 1984 to 1986, and produced a model that consisted of the following components:

– A set of atoms,
– A set of objects
– A code of laws and
– A set of connectors, each connector hanging on a group of objects that must obey a certain law.

Objects have complex structure expressed by including in their "bodies" a set of connectors that hang on other objects making the latter sub-objects to the former. An object's body can also include a connector hanging on the object itself. The dynamics of the objects-connectors model can be defined by a machine in which a connector is regarded as a processing unit that monitors its operands. A connector:

– Awakes when one of its operands has been changed,
– Checks whether the law still holds by reading the condition,
– Restores it when it has been broken,
– Falls asleep.

A law can be fully deterministic, or not. Non-determinism can concern the condition of awakening, or rules of restoring the low, or both. A connector with a non-deterministic law is called a boundary connector. A boundary connector cannot do its job alone. Some help is needed, and here is where human being are introduced in the model. Humans are parts of boundary connectors to help when to awake, or/and how to restore the state of the objects entrusted to this connector.

As we mentioned above, a connector can both be included in the body of an object, and "hang" on this object. This allows an object to reconfigure itself based on changes in other objects. Such reconfiguration can include adding new connecters or removing the existing ones, including the one that completes the reconfiguration itself. The model itself was published later as a theoretical platform [10, 11].

It took several years to figure out in which situations the objects-connectors model could be applied. At that time the author who had participated in the CHAOS project worked for a small Swedish IT-consulting company. The company had developed a system to assist salespersons in the pharmaceutical industry, and the author's task was to support and further develop this system.

A salesperson in this industry was driving around the country, meeting doctors at various hospitals and leaving them samples of new drugs. The system was intended to keep order in his/her business, e.g., plan the trips, have a track of which samples where left in which hospital, and when it is time to follow up the previously made contact. It also helped to gather statistics and analyze sales potential. The business, hence the system, was built around such concepts as, activity, plan activity, report execution of activity, plan follow-up activities, like a phone call.

While working with the system, the author built a model of the sales business in the pharmaceutical domain in terms of objects-connectors model. In this model, a sales lead on which a sales-person is working is represented as an object. As sub-objects, the model includes a hospital, and a doctor to whom this particular lead is related. A planned activity is represented by a boundary connector that wakes up at the deadline point and asks the salesperson to complete it. While executing the activity, the sales person writes a report and plans further activities. In terms of the model, the boundary connector that represents a planned activity removes itself from the body of the lead-object adding at the same time some other connectors (i.e., new planned activities) to it. Part of these activities are calculated based on the fixed rules, others are added manually by the sales person behind the "steering wheel" of the boundary connector that represent an activity under execution.

Creating a model as above gave an idea of one generic situation for which the objects-connectors model could be applied. It could be used for building IT-systems that support business processes. This was tested in several areas with some successes and failures, details of which are reported in [9].

The case above represents a generic invention that consisted of creating a high-level model without having a well-formulated individual or generic situation in mind. An application area for this model was discovered later and more or less by chance. Note, however, that it was not actually a pure chance, as the authors were looking for a generic situation where the model could fit.

5 Related Research

Design science has been proposed as a distinct paradigm in information systems research. In [2], the design science paradigm is contrasted to the behavioral science paradigm. Behavioral science aims at creating theories that explain and predict human and organizational phenomena in the context of information systems. Design science, on the other hand, is the scientific study of artifacts within practices as they are developed and used by people with the goal of solving practical problems. Historically, design science originates from the engineering disciplines and the sciences of the artificial [2]. Its study object is the creation and use of artifacts that can advance individual as well as organizational and societal flourishing. A main tenet of design science is that knowledge and understanding of a practical problem and its solution can be acquired through the creation and use of an artifact, e.g., an IT system, a model or a method.

In design science, the artifact represents a general solution to a class of problems [3]. In this chapter we have introduced the concept of generic problem, generic solution as well as generic situation. We interpret design science research as a movement between two worlds: the world of real (individual) situations and real solutions, and the world of generic (abstract) situations and solutions. This movement between these two worlds is also described in the work of Lee and others [12], but the two worlds in this work are called the instance domain and the abstract domain. In [12], moving from an instance problem and an instance solution to an abstract problem and an abstract solution is called abstraction, and is seen as a way of theorizing in Design Science. Moving from an abstract problem and an abstract solution to an instance problem and an instance solution is called de-abstraction.

In comparison with our approach, [12] does not use the concept of situation. We believe that adding this concept extends the options for a researcher to proceed when a particular test case shows negative results. Besides an option to modify the proposed solution/artifact, the researcher can search for a different, in some respect, situation where the solution/artifact may work.

Designing artifacts can be viewed as a search process through a solution space, [13]. According to this view, the designer starts with setting up a solution space consisting of possible solutions to a given problem. The designer then makes a first design decision, thereby pruning away parts of the solution space. He/she continues and makes further design decisions, which further narrow the space. The process goes on and can be viewed as a systematic exploration of the solution space. At each step, the designer is guided in her decision making by some criteria and requirements on the artifact. This guidance can sometimes take the form of a goodness or utility function, which can be used to determine which design decisions to make. The author of [13] has even argued that the search process can be more or less automated using AI techniques. Though this has not proved feasible, the view of design as a search process is useful in supporting communication, planning and structuring of design work. Our work is akin to that of [13] but differs in that we introduce a space that is spanned by the dimensions of solutions as well as problems and situations.

In the work of Anderson and others [4], four cases of problem solving that give opportunity for research are presented. These four cases have been formalized in Section 3 according to our basic concepts. The chapter also introduces other types of movement between individual and generic SPS-spaces, such as innovative generic problem solving, see Sect. 4.1. The work by Sein and others in [14] points out that IT artifact are typically developed and shaped by their interaction with an organizational context. Therefore, design science research needs to interleave concurrently the activities of creating an artifact, introducing it into an organization, and evaluating it. The basic assumptions of this approach are reflected in our notions of moving between the individual and generic SPS-spaces.

Fischer and Gregor [15] argue for a more thorough investigation of reasoning logic of models for carrying out design science research. They present an idealized model of the hypothetico-deductive method, including the contexts of discovery and justification, and forms of reasoning. Our chapter can be interpreted as a attempt to investigate the reasoning logic in design science.

6 Discussion

In this section, we discuss how the notions introduced in this paper could be used for explaining and conducting design science research. We do not intend to analyze all aspects of design science research process, the aim being just to demonstrate how the proposed apparatus could be used in various stages of the process.

6.1 Tracking the Progress of Design Science Research

As it has been discussed in the introduction, design science research in IS is aimed at generating and testing hypothesis related to the future use of technology in organizations. In our conceptual quasi-formal model of Sect. 2:

- Generating a hypothesis equals to finding a new point $< t, t', GP >$ in the generic SPS-space that has not been investigated at all, or only partly investigated in the past.
- Testing the hypothesis equals to:

- Finding a test case $< s, s', P >$ in the individual SPS-space such that it represents hypothesis $< t, t', GP >$, i.e.: $< s, s', P > \Leftrightarrow < t, t', GP >$
- Implementing s' in s
- Evaluating whether s' is an acceptable solution for problem p in situation s.

As was discussed in Sects. 3 and 4, the order in which the steps above are completed can differ from case to case. For example, in one case, hypothesis generation can precede testing (Sect. 4), and in the next case, it could be vice versa (Sect. 3).

With the help of our notions, the final state of the design science research process could be represented as an octo-tuple $< t, t', GP, s, s', P, i, w >$, where i is a boolean variable that shows whether the solution s' has been implemented in s, w is the weight (i.e., +1, 0 or −1) assigned to $< s, s', P >$. If we allow any component of the octo-tuple to be undefined, (denote this fact as "?"), then the octo-tuple can be used to represent an intermediate state of the research process. Below, we present two examples of intermediate states along with descriptions of steps they warrant for continuing the research process:

– $< ?, ?, ?, s, s', P, true, +1 >$ —research that starts from the individual problem solving, e.g., invention, and requires generalization as its next step
– $<?, t', ?, ?, ?, ?, ?, ?, ? >$ –research that starts with discovering a generic solution for a problem unknown. This state warrants discovering a generic situation where the solution could be applied. This is the case with generic invention as described in Sect. 4.2.

Finding a way for representing the state of the design science research process allows us to consider this process as any other business process, at least from the state-oriented point of view [16]. The main concept of the state-oriented view on business processes is a state of the process instance. The state is defined as a position in some state space. A state space is considered multidimensional, where each dimension represents some important parameter (and its possible values) of the business process. Each point in the state space represents a possible result of the execution of a process instance. If we add the time axis to the state space, then a trajectory (curve) in the space–time will represent a possible execution of a process instance in time. A process type is defined as a subset of allowed trajectories in space–time. The goal of a process instance is defined as reaching a certain area (surface) in the state space called the set of final states. When a process instance is not in a final state, action should be planned aimed at moving nearer to the nearest final state.

The octo-tuple $< t, t', GP, s, s', P, i, w >$ introduced above can be used to represent a position of a research process instance in the process state space. The goal of an instance then can be defined as eliminating all question marks "?" from the tuple.

6.2 In Search for Solutions: Requirements

When design science research comes to a state where the original situation (specific or generic) has been specified, and the problem (specific or generic) has been identified, the next logical step is looking for a solution. In the terms of octo-tuples introduced in the previous section, this can be identified as:

– $< ?, ?, ?, s, ?, P, ?, ? >$ —individual problem solving
– $< t, ?, GP, s, ?, P, ?, ? >$ —generic problem solving that starts with an individual situation and problem, see Sect. 3
– $< t, ?, GP, ?, ?, ?, ?, ? >$ —generic problem solving that starts with a generic situation and generic problem, see Sect. 3.

Search for a solution is a creative process that is impossible to fully predict and/or regulate. However, some general steps in this process can be recommended, one of them being related to specifying requirements in terms of [17] or objectives in terms of [1]. In the terms of this paper, requirements can be considered as a way of

reducing the set of situations, or templates of situations among which the solution can be found. Such reduction can be achieved based on two considerations:

- *Reachability of solution*—taking into consideration whether it is possible to implement a solution in a given situation as-is. For example, moving the SoftMotors office from Stockholm to the Moon could be considered as not reachable, while moving it to India could be considered as reachable.
- *Unintended consequences*—taking into consideration other known problem that a solution might bring about. For, example, all sales staff may leave the office if a cumbersome CRM system is introduced to support sales activities.

Determining requirements of the first type, reachability of solution, could be facilitated by formalizing the idea of distance between situations. Though the concept of distance is practically impossible to define in general terms, it might be done in a specific domain to which the given problem belongs.

Determining requirements of the second type, unintended consequences, could be facilitated by constructing a generic situations-problems space in which all problematic points are marked.

Both types of requirements help to narrow down the search in the solution space in terms of [13].

6.3 Implementing Solution

When a design science research process has reached one of the following states:

– $< t, t', GP, s, s', P, ?, ? >$,
– $< ?, ?, ?, s, s', P, ?, ? >$,
– $< ?, ?, ?, s, s', ?, ?, ? >$,
– $< t, t', ?, s, s', ?, ?, ? >$,

it is time for implementing s' (solution) in s (original situation). Dependent on the distance between s and s', this could be a simple, or quite a complicated process. There can be a known method for implementation, or the method needs to be devised before the solution could be implemented. An example of situation where it needed to be devised is presented in Sect. 4.1. The designing of the implementation method described in Sect. 4.1 has been completed according to the design science approach. Here, we have an example of recursive decomposition where implementation of a solution creates a new problem to solve.

6.4 Evaluating Solutions

When a design science research process has reached one of the following states:

– $< t, t', GP, s, s', P, true, ? >$,
– $< ?, ?, ?, s, s', P, true, ? >$,
– $< ?, ?, ?, s, s', ?, true, ? >$,
– $< t, t', ?, s, s', ?, true, ?>$

it is time for evaluating the results [2]. With a well define problem, this step can be a relatively simple one. When the problem is loosely defined or unknown (marked by "?"), the evaluation could be quite a complicated process on its own, which can bring about a new problem to solve. An example of such a case is presented in [18], which discusses the research on measuring effects from introduction of business process support systems.

For evaluating purpose, one might need to apply empirical research methods as discussed in [2, 17].

6.5 Generalizing

When a design science research process has reached one of the following states:

– $< ?, ?, ?, s, s', P, true, +1>$,
– $< ?, ?, ?, s, s', P, true, ? >$,
– $< ?, ?, ?, s, s', P, ?, ? >$,
– $< ?, ?, ?, s, s', P, ?, ?>$

it could be time for generalizing the result going from a solution for a particular problem in an individual situation to solving a generic problem in a generic situation. This step is called abstraction in design science literature [12].

To complete generalization, one needs to find out whether the problem exists in other situations. The latter could be done via literature search, as we have done in the case described in Sect. 4.1. If the literature search produces no results, the question should be solved via field studies according to the methods of empirical research. The latter case constitutes another example (from the one discussed in Sect. 6.4) of usage of other methods in design science research as discussed in [2, 17].

6.6 Finding a Way Out

When a design science research process has reached state $< t, t', GP, s, s', P, true, -1 >$, the researcher needs to decide how to proceed. As we are dealing with research, not with problem solving as such, the choice is not limited to trying to find another generic solution t'' for the original generic problem GP. For example,

the researcher can investigate whether the solution will be satisfactory for another set of original situations by changing template t to some other template t_1. Alternatively, or at the same time, the researcher can search for other problems that the suggested solution can solve.

7 Conclusion: Plans for the Future

The authors had two personal motivations/goals for initiating research reported in this paper:

- Firstly, as we actively use design science approach in our own research, we were interested in a better understanding of it.
- Secondly, our university department encourages bachelor, master and PhD students to conduct their research and write their theses in accordance to design science. To do this effectively, they need guidelines that give them clear view on all alternatives that the design science paradigm offers.

While we feel that with this work we have achieved some success in pursuing the first goal, achieving the second goal needs additional work. We plan to continue our work in the direction of creating readable (not formal) guidelines for conducting design science research based on the ideas and quasi-formal notions presented in this paper. This can be considered as a test for the model of the design science research process presented in this paper.

References

1. Peffers, K., Tuunanen, T., Rothenberger, M.A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, *24*(3), 45–78.
2. Hevner, A.R., March, S.T., & Park, J. (2004). Design science in information systems research. *MIS Quarterly, 28*(1), 75–105.
3. Baskerville, R.L., Pries-Heje, J., & Venable, J. (2009). Soft design science methodology. DESRIST 2009, New York, NY: ACM, pp. 1–11.
4. Anderson, J., Donnellan, B., & Hevner, A. (2011). Exploring the relationship between design science research and innovation: a case study of innovation at Chevron. Communications in Computer and Information Science. New York, NY: Springer.
5. Neuman, W. L. (2006). Social research methods. Qualitative and quantitative approaches. Boston: Pearson.
6. Andersson, B., Bider, I., & Perjons, E. (2004). *Integration of business process support with knowledge management—a practical perspective. Practical aspects of knowledge management (PAKM, 2004), LNCS 3336*, New York, NY: Springer, pp. 227–238.
7. Andersson, T., Bider, I., & Svensson, R. (2005) *Aligning people to business processes. Experience report*. Software Process Improvement and Practice (SPIP), Vol. 10, No 4, pp. 403–413.

8. Bider, I., & Perjons, E. (2006). Accessing an ongoing introduction of an IT system in operational practice (In Swedish), Inka project report, available at (2012-07-01): http://www.ibissoft.se/projects/inka/INKA_rapport_II.pdf.
9. Bider, I. (2011). *In search for a good theory: commuting between research and practice in business process domain*. *LNBIP*, Vol. 81, New York, NY: Springer, pp. 16–30.
10. Bider, I., Khomyakov, M., & Pushchinsky, E. (2000). Logic of change: semantics of object systems with active relations. *Automated Software Engineering (ASE), 7*(1), 9–37.
11. Bider, I., & Khomyakov, M. (2003). New technology—great opportunities. How to exploit them. In: Filipe, J. (ed.), *Enterprise information systems IV*, Kluver, pp. 11–20.
12. Lee, J. S., Pries-Heje, J.,& Baserville, R. (2011). *Theorizing in Design Science Research. DESRIST 2011, LNCS 6629*, New York, NY: Springer, pp. 1–16.
13. Simon, H. (1969). The sciences of the artificial. MIT Press, Cambridge, Mass.
14. Sein, M. K., Henfridsson, O., Purao, S., Rossi, M., & Lindgren, R. (2011). Action design research. *MIS Quarterly, 35*(1), 37–56.
15. Fischer, C., & Gregor, S. (2011). *Forms of reasoning in the design science process. DESRIST 2011, LNCS 6629*, pp. 17–31.
16. Khomyakov, M., & Bider, I. (2000). *Achieving workflow flexibility through taming the chaos. OOIS 2000—6th international conference on object oriented information systems*, New York, NY: Springer, pp. 85–92.
17. Wieringa, R. J. (2009). Design science as nested problem solving. In: *Proceedings of the 4th international conference on design science research in information systems and technology*, Philadelphia: ACM, pp. 1–12.
18. Bider, I., & Perjons, E. (2007). Effects from introduction of business process support systems and how they can be measured. In B. Pernici & J. A. Gulla (Eds.), *CAiSE'07 workshop proceedings* (pp. 369–377). Trondheim, Norway: Tapir Academic Press.