Requirements Engineering (2021) 26:25–42
https://doi.org/10.1007/s00766-020-00330-4

ORIGINAL ARTICLE

Modeling functional requirements using tacit knowledge: a design science research methodology informed approach

Adrian Benfell[1]

Received: 31 October 2018 / Accepted: 9 March 2020 / Published online: 21 March 2020

Abstract
The research in this paper adds to the discussion linked to the challenge of capturing and modeling tacit knowledge throughout software development projects. The issue emerged when modeling functional requirements during a project for a client. However, using the design science research methodology at a particular point in the project helped to create an artifact, a functional requirements modeling technique, that resolved the issue with tacit knowledge. Accordingly, this paper includes research based upon the stages of the design science research methodology to design and test the artifact in an observable situation, empirically grounding the research undertaken. An integral component of the design science research methodology, the knowledge base, assimilated structuration and semiotic theories so that other researchers can test the validity of the artifact created. First, structuration theory helped to identify how tacit knowledge is communicated and can be understood when modeling functional requirements for new software. Second, structuration theory prescribed the application of semiotics which facilitated the development of the artifact. Additionally, following the stages of the design science research methodology and associated tasks allows the research to be reproduced in other software development contexts. As a positive outcome, using the functional requirements modeling technique created, specifically for obtaining tacit knowledge on the software development project, indicates that using such knowledge increases the likelihood of deploying software successfully.

Keywords Functional requirements modeling · Tacit knowledge · Design science research methodology · Structuration · Semiotics

1 Introduction

To improve software development, the research in this paper details how to utilize tacit knowledge during functional requirements modeling. The research also aligns to the recognized challenge associated with tacit knowledge when developing software [5, 21, 45, 50] and asks the following Research Question (RQ): how can tacit knowledge be obtained and managed in order to contribute to functional requirements modeling? To answer the RQ, the Design Science Research Methodology (DSRM) [6, 7, 24, 28, 39] empirically grounds the research undertaken [18, 27]. The DSRM incorporated input from end-users, a requirements engineer (also the researcher), and the Unified Modeling Language (UML) to change unwanted circumstances into better ones during a software development project for a client.

DSRM affords a process that guides researchers to initiate, develop and measure the impact of an artifact in a problem situation. The artifact must be evaluated to warrant its applicability to form a novel research contribution, either as problem solving or providing a more effective solution. Research rigor is central to the DSRM which includes a process model and a supporting knowledge base [18, 27, 28, 39]. The knowledge base in this research assimilated the development of a framework based upon structuration theory and semiotics [25, 40–42]. The framework first helped to analyze how end-users conveyed tacit knowledge, and second, assisted when making changes to functional requirements modeling to aid its capture and management. Also, structuration theory guided the use of semiotic concepts which formalized a functional requirements modeling technique specific to handling tacit knowledge during a repeat of the requirements analysis phase of the software development project.

✉ Adrian Benfell
a.benfell@uea.ac.uk

[1] University of East Anglia, Norwich Research Park, Norwich NR4 7TJ, UK

To present the research completed, the arrangement of this paper is based upon the stages of the DSRM. First, however, is a section that includes the theoretical background which discusses the presence of tacit knowledge when developing software and possible outcomes if this type of knowledge is not used. The theoretical background section also identifies mechanisms based upon structuration and semiotic theories that can be employed to capture and model tacit knowledge for functional requirements modeling, while also denoting the meaning of social system used in this paper. Section 3, research methodology, outlines the rationale and motivation for using the DSRM. Additionally, Sect. 3 includes the DSRM stages followed to provide a mental model, a template, so that other researchers can check the validity of the research [39]. This paper then follows the stages of the DSRM in Sects. 4–7. Section 4 identifies the problem, the motivation for the research, and the objectives adhered to. Section 5 includes the design and development of the artifact, and Sect. 6 comprises its demonstration. Section 7 has an evaluation of the artifact, the solution, set by the objectives in Sect. 4. Section 8 incorporates a short conclusion.

2 Theoretical background

To develop new software, functional requirements modeling helps to specify what is needed, structures communication, and provides tangible evidence that monitors how software development projects progress [1, 3, 29, 33, 36, 38]. Functional requirements modeling acts as the instrument between project stakeholders to facilitate communication. Hence, it is essential that stakeholders such as end-users communicate requirements unambiguously and requirements engineers interpret them correctly. To specify requirements, stakeholders normally use two different types of knowledge, tacit and explicit. Tacit knowledge has a personal quality that is difficult to formalize and articulate [32, 37, 44] and may also be defined as the skills, ideas and experiences that people own but which are difficult to express and codify [15]. Additionally, people are not cognizant about the tacit knowledge they own, and more importantly, how other people might make use of it. To share tacit knowledge with others normally requires extensive personal contact, regular interaction, and trust [26], as such knowledge can only be shared in specific contexts, for example when people participate in the same social systems [37].

Explicit knowledge refers to knowledge that is codifiable and transmittable using formal systematic languages [17]. Explicit knowledge is therefore easier to discover and model during software development when working with project stakeholders. The amount of tacit and explicit knowledge can fluctuate on a scale whereby one of them takes dominance depending upon the situation [44]. For example, the amount of tacit knowledge increases in conditions where people work or socialize in the same environment and use specificity in language to simplify communication [44]. The scale of fluctuation suggests that capturing and modeling tacit knowledge is difficult for requirements engineers in situations when they have not had extensive personal contact, regular interaction, or built trust. When functional requirements modeling, tacit and explicit knowledge are both needed [5, 21, 45, 50]. Also, many researchers have yet to agree on the best methods for its management, and additionally suggest that where tacit knowledge is owned by experts in a problem domain, requirements elicitation techniques are not guaranteed to capture tacit knowledge [8, 10, 12, 22, 29, 31, 45, 50].

Reviewing functional requirements modeling further indicates that it utilizes the combination of diagrams and narrative to represent proposed software [1, 4, 16, 19, 23, 43, 53]. Typically, functional requirements are captured during elicitation when examining documentation, carrying out observation, in meetings, doing interviews, or distributing questionnaires. For example, when applying Unified Modeling Language (UML) Use case diagrams, the symbols available become topics of conversation that can be asked, and when combined with recommended structured narrative in UML [51], help to frame questions and elicit answers from individuals. This process unfolds as the symbols available that represent actors, use cases, and associations in Use case diagrams, also form links to narrative dictionaries that must be populated for software development [30, 51].

Applying UML begins the process of modeling which relies upon explicit knowledge. For instance, Fig. 1 illustrates how the application of different symbols belonging to UML modeling techniques is used to create requirements models. Figure 1 shows a typical approach to functional requirements modeling starting with Use case diagramming to denote actors, use cases, associations, include, extend and generalization. Each element helps to detail use cases further with supporting narrative collected from end-users. The process continues by deconstructing each use case with other types of UML diagram, shown in Fig. 1, to help form completed requirements models.

When functional requirements modeling with UML using the elements in Fig. 1, this escalates the difficulty of capturing and modeling tacit knowledge. For example, the application of UML results in the compilation of a vocabulary specific to software development, described as a "kernel vocabulary" [30]. Such vocabularies during functional requirements modeling can become unfamiliar to different types of stakeholder who use different terms and definitions in a project. Some terms and definitions stakeholders use are known explicitly, but other ones are known tacitly. Hence, stakeholders are likely to find it difficult to understand how

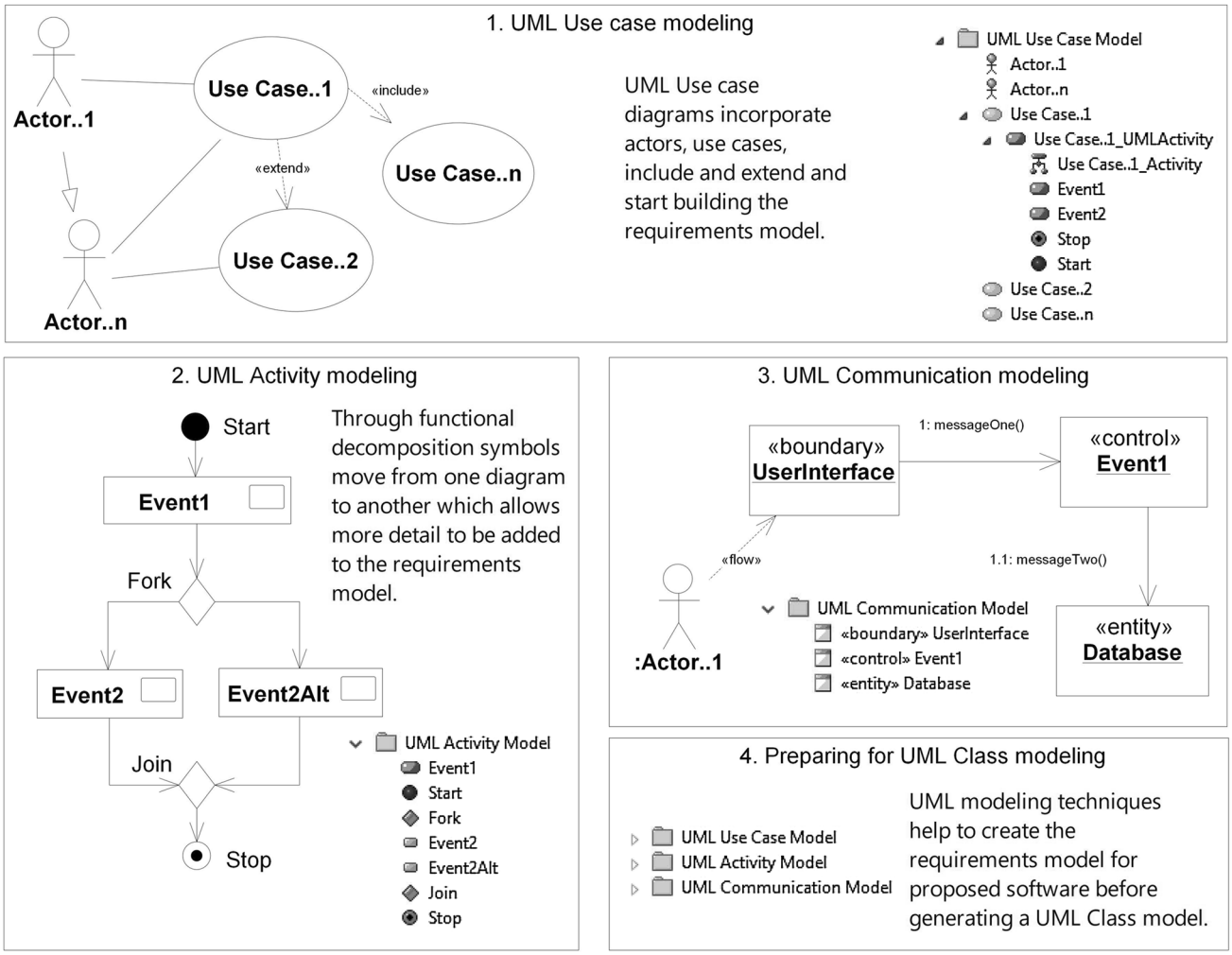

Fig. 1 Requirements modeling with the UML

their views and expectations for new software are represented if they do not understand how their requirements are being represented by UML. Hence, where tacit knowledge is owned by stakeholders and is not as expressible as explicit knowledge, requirements elicitation techniques are not guaranteed to capture tacit knowledge [5, 21, 45, 50]. To explain this dichotomy further, the duality of social structure and agency in structuration theory illuminates how tacit knowledge forms in stakeholder minds, and when engaged with social systems, how it might be discovered more expediently during functional requirements modeling.

2.1 The duality of structure, tacit knowledge and social systems

The sharing of tacit knowledge between individuals, previously discussed, suggested that it requires specific contexts, extensive personal contact, regular interaction, and trust, in social systems [26]. Structuration theory helps to explain this situation. The duality of structure and agency is the central configuration in structuration theory, as duality relates to the relationship that people as agents have with structure, as agency cannot exist independently [25]. The interplay of the two sustains social systems and how people acquire tacit knowledge. With reference to Fig. 2, structure is determined by signification, domination, and legitimation supported by three modalities, interpretive scheme, facility,

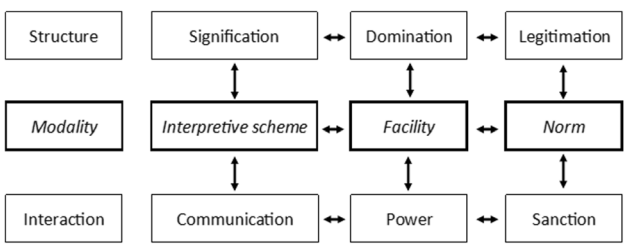

Fig. 2 Structuration theory [25]

and norm, each characterized by language and communication. In Fig. 2, for agents to interact in social systems, structure incorporates the rules of language used to facilitate communication, but also allows agents to create new words, acronyms, and so on with associated meaning. The known system of interaction maintains the rules of language and communication so that hearers understand what speakers say. Language and communication delineate social systems, thus the association between structures and agents generate signs that are then interpreted in communication [25].

2.2 Signification, semiology and semiotics

Referring to Fig. 2, for interpretive schemes to work structuration theory includes a semiology and semiotic view of signification [25]. The view, however, is not developed in detail. In semiology and semiotic theories, different sign parts unify in a relationship, as either dyadic or triadic [40, 46]. These theories have a subjective element to understanding signs present in communication, as signs can have alternative meanings within different social systems [40, 46]. For example, when the parts work in combination, interpreting signs in communication, called signification, involves semiosis [2, 20, 35, 47, 49]. Semiosis combines how signification occurs involving the actual sign, its object, and someone's understanding. However, to simplify semiosis it can be organized into three dimensions [35]. Syntactic as the relationship between signs, semantic as the relationship between signs and the objects they represent, and pragmatic as the relationship between the sign and the interpreter [35].

Semiology, the science of signs, advocates that a sign is a dyadic conceptual object which consists of a signifier, the sign's physical attributes such as a sound, a printed word, or an image, and the signified, as the meaning suggested by a sign distinct from its physical form in someone's mind [46]. The purpose of semiology helps to understand the relationship between the signifier and signified of the language used in different social systems. Alternatively, a triadic view of signs exists called semiotic [40–42]. In semiotics a sign is said to be triadic as there is a relation between the sign, its object and someone's mind, the interpreter. Signs in this context help to construct a relationship between the mind and experience, and signification occurs when signs cause habitual changes in the interpreter. The triadic form of semiotics also relates to understanding. First, qualisign (a quality), sinsign (something that exists) and legisign (a general law) categorize signs according to their type, the sign itself. Second, the trichotomy of icon (similarity), index (points to) and symbol (symbolic) identifies how signs relate to their object. Third, rheme (possibility), dicisign (fact), and argument (reason) relate to the degree of comprehension, the actual interpretation of signs.

Regarding semiology, a sign in a signifier and signified association are indivisible for meanings to emerge, and semiotic theory is similar by including a sign and object in place of signifier and signified [40–42, 46]. However, semiotics includes the concept of an interpretant which includes the three types of sign, rheme, dicisign, and argument to classify how signs influence people, and how people can change the meaning of signs in repeatable processes of semiosis. Hence, the interpretant identifies how people interpret signs within the social systems they belong to, explicitly and tacitly, in a duality of structure and agency. Based upon this premise, semiotic theory (referred to as Peircean semiotics for the remainder of this paper) is favorable to show how tacit knowledge, held by individuals in social systems, may be obtained and modeled by requirements engineers.

3 Research methodology

The rationale for using the DSRM links to its focus upon the creation of how things should be to achieve goals [24, 28, 39, 48]. Consequently, the purpose of design science is "to change existing situations into preferred ones" to address specific problems through artifact creation [39]. The artifact shown in this paper, a functional requirements modeling technique called "Normative statements" fits the two important characteristics of design science, novelty and relevance [24, 28, 39]. For novelty, Normative statements solved the problem associated with obtaining tacit knowledge in the requirements phase of the software development project (called SupportManagerDB for the remainder of this paper). Regarding relevance, Normative statements add to the discussion associated with tacit knowledge during functional requirements modeling.

The DSRM additionally provides a research process and a template for presenting and conducting design science research, applicable to the challenge associated with tacit knowledge that emerged during the SupportManagerDB project. Table 1 shows how the DSRM [15, 24, 27] was applied for the research in this paper. Design science research methodologies must be theoretically, internally and empirically grounded [15]. This research is grounded so that other researchers can evaluate this paper with confidence. For theoretical grounding, the knowledge base used in this paper incorporates widely understood functional requirements techniques based upon UML, and structuration and semiotic theories. Internal grounding means to have logical consistency, a clearly described and reusable process [15]. The stages of the DSRM provides the apparatus for logical consistency in Table 1. To show the empirical grounding of DSRM, the application of the methodology was successful in practice, it changed a difficult situation into an improved one [39].

Table 1 DSRM phases, activities, knowledge base and grounding

DSRM phases (internal grounding)	Phase and associated activity description	Linked knowledge base (theoretical and empirical grounding)
1. Problem identification and motivation (Sect. 4)	*What is the problem? Justify the value of a solution* Requirements modeling needs to be changed to accommodate tacit knowledge to improve deployed software for end-users	Real-world problem extracted from the software development project SupportManagerDB project to *empirically ground* the research methodology
2. Define the objectives of a solution (Sect. 4)	*How should the problem be solved? What criteria are pertinent to success?* To adjust UML to allow the capture and use of tacit knowledge. Tacit knowledge must be used to improve the specification of requirements for new software	Knowledge of software engineering and requirements modeling based upon UML to *theoretically ground* the DSRM owned by the requirements engineer/researcher. Structuration and semiotic theories to understand the capture and use of tacit knowledge to further the *theoretical grounding* of the DSRM
3. Design and development (Sect. 5)	*Create an artifact that solves the problem using constructs, models, methods or instantiations in which the research contribution is embedded* Devise a framework that informs how to change requirements modeling and instantiate those changes using UML so that the wider software development community may make use of the functional modeling technique(s) devised	Structuration and semiotic theories, and requirements modeling with UML to *theoretically ground* design and development
4. Demonstration (Sect. 6)	*Demonstrate the use of the artifact by solving one or more instances of the problem* Illustrate the composition of, and demonstrate how, the functional requirements modeling technique(s) obtained and used tacit knowledge	Demonstrate how to apply the new functional requirements modeling technique within the real-world SupportManagerDB project to *empirically ground* the research
5. Evaluation (Sect. 7)	*How well does the artifact work? Did it meet the stated objectives (in 2) with observed results?* Comparative analysis with requirements modeling techniques in UML and the requirements phase in the real-world SupportManagerDB project. Also, indicate how the evaluation of the artifact might iteratively revise Phase 2 and 3 of the DSRM	Understanding of the functional requirements modeling technique, its strengths, weaknesses and applicability to contemporary functional requirements modeling theory and practice and observe its appropriateness for requirements modeling to show the *empirical grounding* of the DSRM
6. Communication	*Communicate the problem, usefulness, novelty, and effectiveness of the solution*	

The first column in Table 1 lists the research phases and their sequence, and the corresponding row for each phase identifies the tasks completed. For rigor, the last column links the knowledge base with the different tasks in each phase. The knowledge base is made up of knowledge tools such as foundational theories, frameworks, instruments, constructs, models, methods, and instantiations [39]. Integrating a knowledge base improves the application of the DSRM as the most suitable knowledge tools are selected according to the objectives of the proposed solution [24, 28, 39]. Building the knowledge base was first governed by the identification of "dialog reference points" [14, 34], second, the expected application of UML set by the SupportManagerDB project, and third structuration and semiotic theories as they have been applied in information systems research [32, 47, 49, 52]. For example, research into requirements modeling maximizes the potential to consider the role of structuration theory using its signification and communication pillar [32] which embeds semiotics. The pillars of structuration theory were also used to explore cross-cultural software production [52].

4 Problem identification, motivation and objectives of a solution

The structuration theory view of social systems, made up of structure and agency organized as signification, domination and legitimation pillars, refers to the group of people that required a new software system, SupportManagerDB, for managing tenants and accommodation. TenantManagement, a pseudonym, denotes the social system for the remainder of this paper. The accommodation management team comprised eight Support Managers (one as a Senior Support Manager), a Head of Residential Services, a requirements engineer (also the researcher), and to help scope initial project requirements, two members from an Information Technology (IT) support department. Support Managers and the Senior Support Manager specified most of the functional requirements and are referred to as end-users for the rest of this paper. The two members from the Information Technology (IT) support department met only at milestone meetings to check progress. They had knowledge of UML and programming, and with the requirements engineer, ensured the consistent use of UML.

The IT support department requested that the requirements engineer follow a user-centered UML [51] Use case driven software development process as they wanted end-users involved during the requirements analysis phase of the project. All end-users attended meetings linked to functional requirements elicitation, such as project scoping and providing functional requirements as verbal and textual descriptions during semi-structured meetings. The IT support department helped to verify the requirements model as end-users were not expected to understand the application of UML [51] beyond UML Activity diagrams. The process allowed the review and modification of the requirements model and other artifacts such as the design model and software at timely intervals. For design and implementation, all team members met to approve functioning software only. Key end-users (the Senior Support Manager and two Support Managers) had between 5 and 10 years of work experience at TenantManagement and had built-up extensive knowledge related to business functions. The remaining Support Managers had comparative experience when working at other social systems. Each end-user owned a portfolio of accommodation according to location and number of tenants. End-users attended training courses associated with their role adding to their expertise. They had varied levels of IT skills but as a minimum showed competence when using computer-based office products.

The requirements engineer worked with the accommodation management team, carried out all software development activities, organized with the IT support department review meetings and recorded feedback. To help understand functional requirements, previously completed paper-based forms were made available. The project timescale included 1 year for software development, and an additional year to review the new software system and make small modifications if required. The project incorporated a further 2 years consisting of four reviews to monitor the functioning of the software system. The project lasted for three-and-half-years and the team structure remained for the whole duration. During that time five of the eight Support Managers changed employment and TenantManagement recruited others to maintain the team structure. However, core members of the team, the Head of Residential Services, Senior Support Manager and two Support Managers provided stability during the DSRM. In addition, while the project lasted for three-and-half-years, there were "spikes" in activity linked to the supporting software development process, as phases, shown in Table 2.

To use UML [51], the supporting software development process followed iterative development and included the phases project scoping, requirements analysis, systems design, and implementation. In the first year of the project, formal meetings occurred to check progress and comprised three scheduled major reviews, in the first month *project scoping*, in the sixth a *requirements and design review*, and in the last month, a *post implementation review*. In between the major reviews, informal meetings took place with end-users, mostly in one-to-one settings, for functional requirements elicitation. To help with functional requirements modeling a UML-based Computer-Aided Software Engineering (CASE) tool was used.

Table 2 Composition of the requirements model

Phase	Team activities	Artifact
1. Initial scope of requirements	Individual meetings with all Support Managers Team review	Project initiation document
2. Use case modeling included: narrative for each use case (11 use cases) Activity model for complex use cases (for 8 use cases—8 Activity models) Communication model for each use case (for 11 use cases) according to UML stereotypes Pilot class model (merging of 11 communication models)	Individual meetings with all support managers Team review (including IT support team to very UML diagrams)	Requirements specification
3. Class modeling	Team review	Design specification—class model
4. Detailed class modeling: finalized class model, relational database model, and C# skeleton code generation	Team review	Programming specification
5. First phase deployment of SupportManagerDB project	Team review (all end-users)	Full working software

The CASE tool facilitated the start of functional requirements modeling with a UML Use case diagram [51] and this specified how the proposed software from actor (end-user) perspectives should work. Typically, the actors depicted the different jobs that Support Managers were responsible for when interacting with the proposed software system, and several use cases according to the needs of actors detailed initial software functionality. Actors stated roles worked through by end-users and any expected data transfer modeled interaction. In the UML-based CASE tool, numbered steps showing the full functionality of each use case, completed as chains of actions by actors, then moved to the UML Activity [51] component of the requirements model for further elaboration, and then onto UML Communication modeling to show the coordination of boundary, control and entity stereotype classes [51] and the ownership of functions between them. Based upon the UML Communication diagram, the CASE Tool facilitated the creation of a pilot UML Class model.

The first key review meeting, *project scoping*, assessed preliminary requirements using a project initiation document supported by a UML Use case model with additional narrative. The key meeting 6 months into the project, *requirements and design review*, considered the entire requirements model for the project incorporating UML Use case, Activity, and Communication diagrams and a pilot UML Class diagram. During the first 6 months, the requirements engineer organized with all Support Managers other less formal meetings (two each) to begin work on requirements. At the end of each informal meeting, Support Managers provided feedback informing any changes needed.

After achieving agreement for the UML Use case model, complications surfaced. When discussing the requirements for the proposed software during one-to-one and group meetings, applying UML forced the creation of a kernel vocabulary [30]. The UML Use case model consisted of actors, use cases, and different associations (includes and extends), formed its composition. Team members did not experience communication and understanding difficulties in meetings when the requirements engineer used the basic elements of UML Use case modeling only (which excluded includes and extends). However, as UML Use case modeling started a functional decomposition process by naming use cases according to a verb-object convention, the process of clarifying and agreeing use cases also allowed specific terms to emerge which demonstrated how the social system was organized in line with structuration theory. For example, when using a term such as "incident", it invoked a range of responses from the end-users. For example, Support Manager A stated, "during each *incident* I also have to consider a range of *incidents* from assault to threatening behavior". Support Manager B said, "when I deal with an on-call *incident*, report forms must have follow-up actions. I add my write-up of follow-up actions to the forms in the spare boxes provided". The Senior Support Manager also indicated "all support managers must discuss additional *incidents*. My experience in many cases show that most often when a tenant is at fault over an *incident*, there are others to be discussed and reported. I also amend the form with additional *incidents*".

Difficulties appeared, however, when progressing into UML Activity and Communication modeling and completing the kernel vocabulary as communication was difficult which resulted in missing tacit knowledge. The Senior Support Manager said, "Those diagrams look interesting. I don't understand them that well, but I trust what you are doing. I am looking forward to getting the new database to make my job easier". Using the modeling techniques that incorporate formalized but different procedures and symbolic notation, team members were unable to express their understanding with the same clarity when the requirements engineer used UML Use case diagrams. Consequently, time quickly advanced toward the six-month key meeting, *requirements and design review*, resulting in an inaccurate

requirements specification. The key meeting 6 months into the project, *requirements and design review*, considered the entire requirements specification which incorporated the UML Use case, Activity, and Communication models, and the pilot UML Class model. The accommodation management team judged the requirements specification to be inaccurate. However, a preliminary software system built from the functional requirements model helped to meet the initial one-year deadline. This plan eased time pressure allowing the DSRM, for 3 months, to be followed at the beginning of the second year of the project. At the last key review meeting, *post implementation review*, the accommodation management team indicated that the SupportManagerDB project met very few requirements. Table 2 presents the requirements model (particularly phases 1 and 2) at this stage of the SupportManagerDB project. It also shows each phase followed and corresponding requirements and design tasks, team activities with each phase, and the artifacts created.

When using the UML Use case diagramming techniques, end-users did not experience communication and understanding difficulties in meetings, as the basic structure of UML Use case modeling was explained and supported by the requirements engineer, end-users engaged with the modeling process. However, when introducing complexity such as "includes" and "extends" to begin functional decomposition, end-users became concerned that their views were not being documented in the requirements model. For example, Support Manager D said, "I can't follow the picture showing what I do after you added more detail. Maybe I should wait until I see the database to see if it works as I hope it will". However, as previously pointed out, the UML Use case driven approach started a functional decomposition process by naming use cases according to a verb-object convention, ready for further modeling with UML Activity and Communication diagrams. The process of clarifying and agreeing the structure of use cases did allow end-users to identify and agree to specific terms such as "case reporting", "incident", "case time allocation", "accommodation change request", and others and for the requirements engineer/researcher to record them.

The difficulty of capturing and modeling tacit knowledge emerged as a challenge during meetings with end-users. Often, contradictory requirements were provided by end-users according to their understanding of localized needs as each Support Manager had responsibility for a specific area and several tenants. The more experienced Support Managers had developed different work processes which achieved expected outcomes in terms of work deliverables but were not cognizant of the changes they made. Hence, at the beginning of year two of the project, the researcher piloted the idea of working with the terms identified by asking team members to explain them further and following the DSRM. Working with strict time pressures, travelling to different locations and again in mostly one-to-one meetings, consensual agreement did, however, quickly form across end-users when identifying different terms. The researcher took the view that such specific terms might give clues to unlocking functional requirements based upon tacit knowledge more accurately than UML modeling techniques alone. Set by the first two phases of the DSRM, the following list defines the objectives of a solution agreed with end-users:

1. Create a functional requirements modeling technique that begins with specific terms and models tacit knowledge while maintaining user-centeredness within the SupportManagerDB project;
2. As a condition placed upon the SupportManagerDB project, ensure that the new functional requirements modeling technique works with UML Use case modeling and aligns seamlessly with design, specifically with UML [51] Class modeling;
3. To maintain the original schedule, the functional requirements modeling technique must enable requirements to be modeled speedily as end-users agreed to check requirements but insisted on the deployment of a revised software system at the end year two of the SupportManagerDB project to match the original timescale.

5 Design and development

To represent tacit knowledge as it appeared in the SupportManagerDB project, the specific terms that emerged, "case reporting", "incident", "case time allocation", "accommodation change request", and others, are signs [2, 20, 35, 47, 49]. Such specific terms take the form of "dialog reference points" and can include acronyms, labels, diagrams and so forth known only to agents within social systems [14, 34]. For example, end-users consistently used dialog reference points such as "case reporting", "incident", "case time allocation", "accommodation change request", and others, during the original requirements analysis phase of the SupportManagerDB project. Table 3 shows a collection of dialog reference points discovered during meetings and detailed further when following the DSRM.

The dialog reference points identified can be explained by the signification and communication pillar in structuration theory. To understand the role of dialog reference points [14, 25, 34] in the SupportManagerDB project, highlighted that they identified the scope of the problem domain in the social system. To explain this further, the following example based upon "Fire training" (taken from Table 3) is used. In the problem domain the process of signification occurs when a dialog reference point is identified—"Fire training". The interpretative scheme is made up of causes and effects. Allocative and authoritative resources are used to

Table 3 Dialog reference points and tacit knowledge

Dialog reference point	Tacit Knowledge (agreed to and shared by end-users)	
	Cause	Effect
Case reporting	Contact time statistics for Incident on call reporting must include episodes, minutes, and hours Account for types of incidents with totals Report types of contact as application, drop-in, visit, email, letter, telephone with a total for each type	Submit a report for each case with timely figures to verify resource allocations linked to each support manager
Case time allocation	Record and account for our time used in minutes for each case	Calculated and record time used in a report and forwarded to head of residential services
Incident	Consider a range of incidents from "assault" to "threatening behavior" On-call incident report forms must have follow-up actions Discuss additional incidents Save incident records	Ensure accuracy and recording of incident data in a report and escalate to head of residential services and other stakeholders
Accommodation change request	All tenants must see a support manager before the transfer to other accommodation Use a priority scale 1, 2 and 3 for requests. 1 being the highest. The priority scale determines a queueing order Approve accommodation requests before a change of circumstances can begin	Approve tenants only for a change of accommodation once an accommodation change request is complete
Change of circumstances	Give reason for leaving Record new address Verify and record other items such as keys returned, damages bill applied and amount, date of leaving and the new address to complete a change of circumstances	A tenant moves accommodation
Calculate time in lieu	Link accumulation of worked minutes and hours to Incident cases	Recorded report from "Case time allocation" must match calculated time in lieu
Fire training	Record malicious activation of a fire alarm as an incident A tenant must attend a welfare meeting A tenant must go to fire training	Fire training completed and head of residential services notified
Discipline	Allegation details of misconduct complete each discipline record before saving Agree and record outcomes between the tenant and support manager Discipline meetings result in a range of warnings and actions from "informal verbal warning" to "notice to quit"	Discipline case escalated to head of residential services

identify effects associated with each dialog reference point. For example, Support Manager G said, "fire training must be completed", and the Senior Support Manager stated that a "report must be sent to the Head of Residential Services". The Head of Residential Services also specified in a team meeting that reports related to the malicious activation of fire alarms must be compiled. When identifying effects, power was exerted through relationships, for example between Support Managers and from the Head of Residential Services. To achieve the desired effects, causes were identified, "malicious activation of a fire alarm is recorded as an incident, a tenant must attend a welfare meeting, and a tenant must go to fire training". In identifying causes, if Support Managers did not complete expected actions it invoked "sanctions" as the desired effects, work outputs, were not evident. The process of identifying dialog reference points and associated causes and effects resulted in amendments made to Fig. 2 of the original form of structuration theory and is shown in Fig. 3.

The structure in Fig. 3 represents how end-users began detailing dialog reference points based upon their tacit knowledge of the problem domain. The top layer represents the problem domain, and the bottom layer end-users. In structuration theory the term structure and not problem domain is used, and interaction and not end-users. However, the term problem domain follows structuration theory as rules and resources organized as properties of social systems, and end-user aligns to interaction. The top and bottom layers of each pillar in Fig. 3 connect through the same

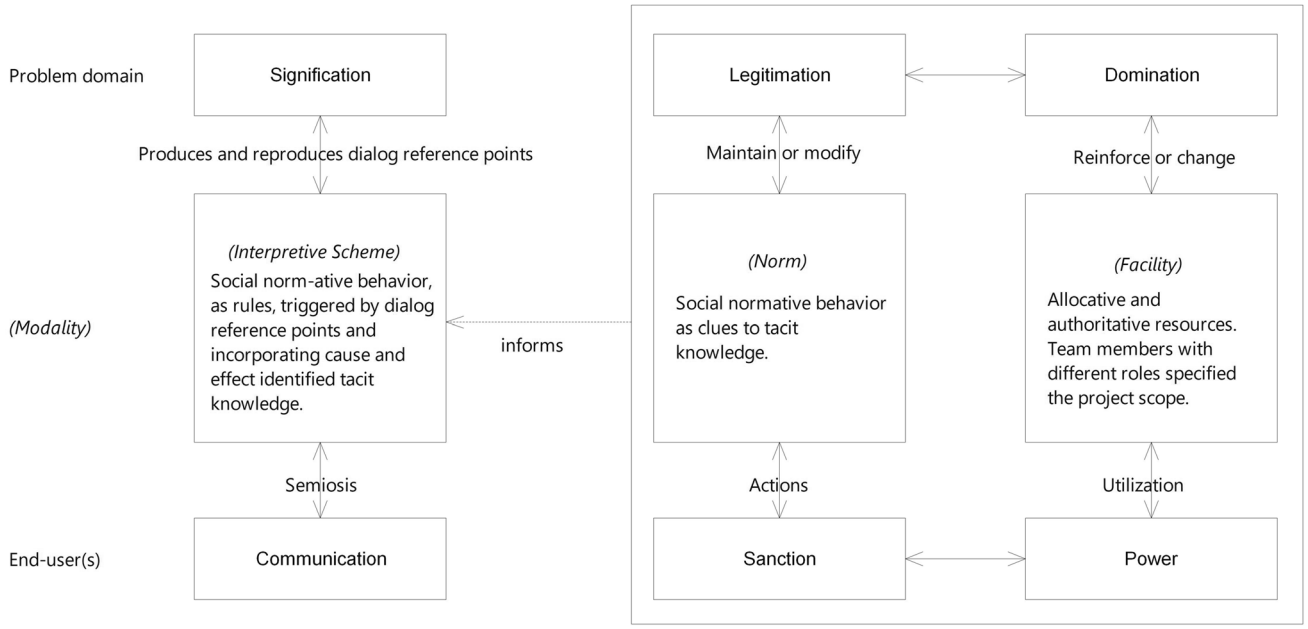

Fig. 3 DSRM knowledge base derived from structuration theory [25]

three modalities, interpretive scheme, norm, and facility [25]. Working across the top layer in Fig. 3, the pillar *signification and communication* was adjusted to show how end-users encoded and decoded tacit knowledge, starting with dialog reference points, and detailing each one using cause and effect statements. Also, following the cause and effect statements in Table 3, the *legitimation and sanction* and *domination and power* pillars represent the identification of causes and effects. For example, regarding the control of resources, *domination and power* depicted how the cause and effect statements were influenced by power relationships between end-users and other team members with a higher level of authority, such as the Head of Residential Services.

The combination of causes and effects linked to each dialog reference point is a process of Peircean semiosis. Referring to Fig. 3, the modality interpretive scheme assigns importance to signification. When end-users communicated functional requirements in meetings when repeating the requirements analysis stage during the DSRM, they did so by choosing to state effects and causes linked to dialog reference points. The researcher discovered this approach simplified communication and improved understanding when compared to UML [51] beyond basic UML Use case modeling.

When semiosis occurred during this phase of the DSRM, the researcher observed end-users drawing upon tacit knowledge to identify causes and effects. For example, when end-users discussed the dialog reference point "incident" in a team meeting, the agreed desired effect that emerged included the necessity to submit a report to the Head of Residential Services. The Senior Support manager said, "TenantManagement now requires all case time allocations to be recorded, reports produced which I have to send to [Head of Residential Services] with Support Managers holiday allowances according to time in lieu". Then Support Manager C said, "I haven't been completing the forms required by [Head of Residential Services]. Does that mean my time allocations are incorrect and my time in lieu hasn't been calculated correctly?" In the process of semiosis, end-users decided that they had to consider a range of incidents from "assault" to "threatening behavior", record any follow-up actions, discuss any additional incidents, and ensure that all information is saved to a database and shared with other end-users (in the Support Manager role) occurred before forwarding reports to the Head of Residential Services.

The interpretation of dialog reference points previously described are interpretant signs [40–42]. These are signs that evolve in someone's mind when the circumstances of a situation permit interpretation to work toward completion. In the process of semiosis, it is an interpretative result that each person will arrive at if signs are sufficiently reflected upon. In terms of functional requirements modeling for the SupportManagerDB project to capture and model tacit knowledge, this pointed to the creation of a formalism that could be used by the requirements engineer with end-users to understand, communicate, and qualify tacit knowledge. When aligning dialog reference points to signification in structuration theory, Peircean semiotic theory pointed to rhemes (formal quality such as the word "report" which represents the possibility of a "report"), dicisigns (formal indexicality as a sentence formed of rhemes to assert existence,

"You have to create a report for the Head of Residential Services"), and arguments (formal mediation as a sequence of dicisigns, "You have to create a report for the Head of Residential Services. Therefore, you should save it to the database"), relate to the degree of tacit knowledge comprehension. Interpretant signs took the leading perspective to develop the functional requirements modeling technique to represent tacit knowledge.

Tacit knowledge when linked to foundational categories to infer meaning in Peircean semiotics, also emerged as different grades of clarity based upon interpretant signs. The first grade involves familiarity with a concept in day-to-day encounters (the representamen), and the second the ability to offer general definitions of a concept (the object). In the third, end-users knew, and agreed to, what effects to expect from holding a concept to be true (the interpretant) [2]. In terms of a change to functional requirements modeling, this structured the formalism to arrive at interpretant signs. For example, regarding dialog reference points and generating cause and effect statements, the first grade of clarity related to an unreflective awareness of them in everyday experience. When personal experience and consensus between end-users existed for each one, this offered a second grade of clarity and provided some generalized concepts associated with dialog reference points. In these two grades of clarity, although end-users understood and communicated functional requirements for the new software, deeper comprehension was available as the third grade, to intellectualize dialog reference points as the interpretant. The interpretant emerged by knowing what effects to expect from holding a dialog reference point to be true and being able to ensure correctness of the effects identified in terms necessary and sufficiency conditions in cause statements [11].

Working further with structuration theory, the pillar *legitimation and sanction* combined with the pillar *domination and power* are the extra dimensions of structuration theory that helped to uncover tacit knowledge through power relationships between Support Managers and authoritative relationships, with and between, the Senior Support Manager and the Head of Residential Services. Both affected the quality of cause and effect statements and their communication. These two pillars follow the idea of social norms [9]. They represent how end-users understood the problem domain through an array of influences which marked further consensus to cause and effect statements. Hence, dialog reference points and linked cause and effect statements are representative of personal end-user experience and views through autonomy, formed tacitly and agreed to as interpretant signs.

Using the dialogue reference points documented in Table 3, they follow the semiosis process associated with interpretant signs. Throughout this phase of the DSRM, when end-users communicated using dialog reference points, this began shaping a formalism based upon the *interpretive scheme* modality. Thus, to intellectualize dialogue reference points in a formalism and use these for functional requirements modeling as a third grade of clarity, modeling the cause and effect statements in the SupportManagerDB project guided the development of the meta-model, presented in Fig. 4, to fit the DSRM objectives of a solution. The meta-model determined how to generate the requirements modeling technique that captures and models tacit knowledge.

6 Demonstration

The modalities in structuration theory and added to the knowledge base framework in Fig. 3, interpretative scheme, norm and facility, helped to configure changes to requirements modeling based upon Fig. 4. The Interpretative Scheme in Fig. 3 incorporated the *signification and communication* pillar by identifying the encoding and decoding of tacit knowledge driven by dialog reference points. Also, according to the cause and effect statements shown in Table 3, the *legitimation and sanction* pillar shown as Norm in Fig. 4, included normative perceptions rooted as cause and effect statements. Regarding the control of resources, *domination and power* depicted power relationships between team members and how these manipulated cause and effect statements, endorsed by all team members and shown as Facility in Fig. 4.

The user-centeredness of UML Use case modeling continued to identify all team members as roles linked to the proposed software, required by the *legitimation and sanction* pillar, which resulted in the creation of the functional requirements modeling technique called "Normative statements", from Fig. 4, in this research. As the SupportManagerDB project had a project initiation document coupled with a range of use cases, with each use case identifying linked actors, the term 'role' replaced 'actor' to show the pillar *domination and power*. Information recorded with each role included the influence it had over other team members. Each Normative statement inclusive of causes that supported an effect became a candidate function within a candidate class to seamlessly link with design, and each effect had one or more causes combining necessary and sufficient conditions [11]. The structure in Fig. 4 provided the apparatus to intellectualize tacit knowledge as Peircean interpretant signs and devise a functional requirements modeling technique that met the demands of stage 2 of the DSRM.

Regarding the revision to functional requirements modeling within the context of structuration and semiotic theories to model tacit knowledge, quick consensus formed between end-users when working with Normative statements. Agreement occurred in one-to-one meetings and approved in final requirements and design review. The Normative statements

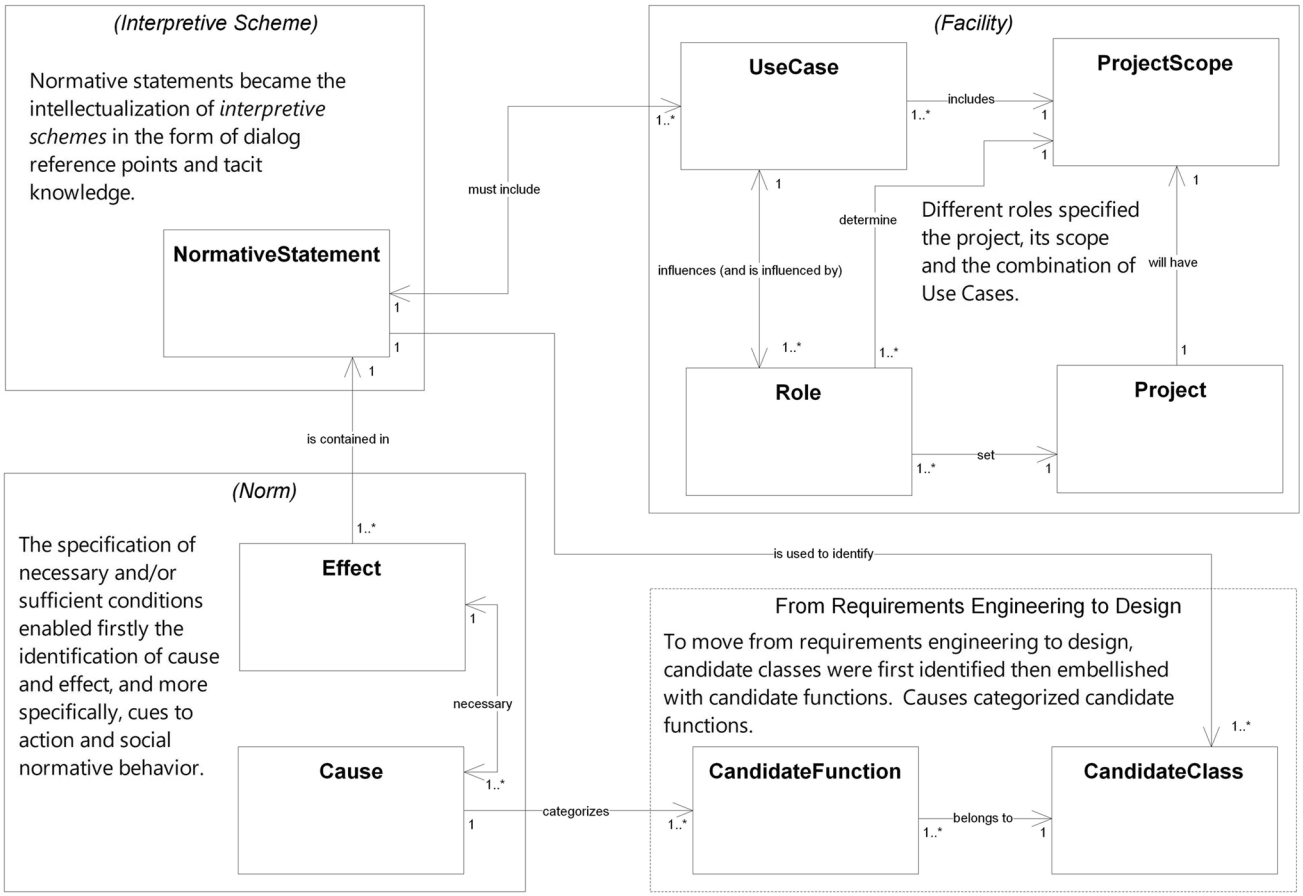

Fig. 4 Normative statements meta-model

represented the functional requirements incorporating tacit knowledge by end-users better. Normative statements allowed an autonomy of expression with short natural language blocks of text permitting all team members in different roles to identify supporting necessary conditions to effects. Figure 5 presents an example Normative statement.

Candidate functions in Normative statements were mapped over to the stereotype classes, entity, controller and boundary to generate a pilot Class diagram as it is shown in Fig. 6. For detailed design, the class diagrams for each Normative statement were merged into one forming a completed UML Class model. The UML-based CASE Tool helped to manage this process by eliminating similar classes. The stereotypes used in the UML Class model used (boundary, control and entity [51]) structured the type of software asked for in the SupportManagerDB project.

Regarding the importance of iteration as part of the DSRM [39], the dialog reference points starting with "Case reporting" were modeled first followed by "Case time allocation" and then working through the list in Table 3. Refinements included how candidate functions linked to candidate classes. The stereotypes boundary, control and entity indicated in UML [51] provided a mechanism to allocate functions based upon causes in Normative statements. When reaching the dialog reference point "Incident", using boundary, control and entity stereotypes routinized the adding of functions to classes. The objectives set out in Phase 2 of the DSRM remained unaltered.

7 Evaluation

To address Objective 1 set in phase 2 of the DSRM, requirements modeling began with a basic UML Use case diagram to confirm dialog reference points with end-users, shown in Fig. 7. Normally, when naming use cases, the verb-object naming convention is used [51], but unnecessary for the SupportManagerDB project as the sole purpose of the Use case model was to show dialog reference points, actors, and interaction. The Senior Support Manager confirmed the UML Use case model, "I found this quite easy to follow. I could also see visually the scope of the project which makes it look quite simple".

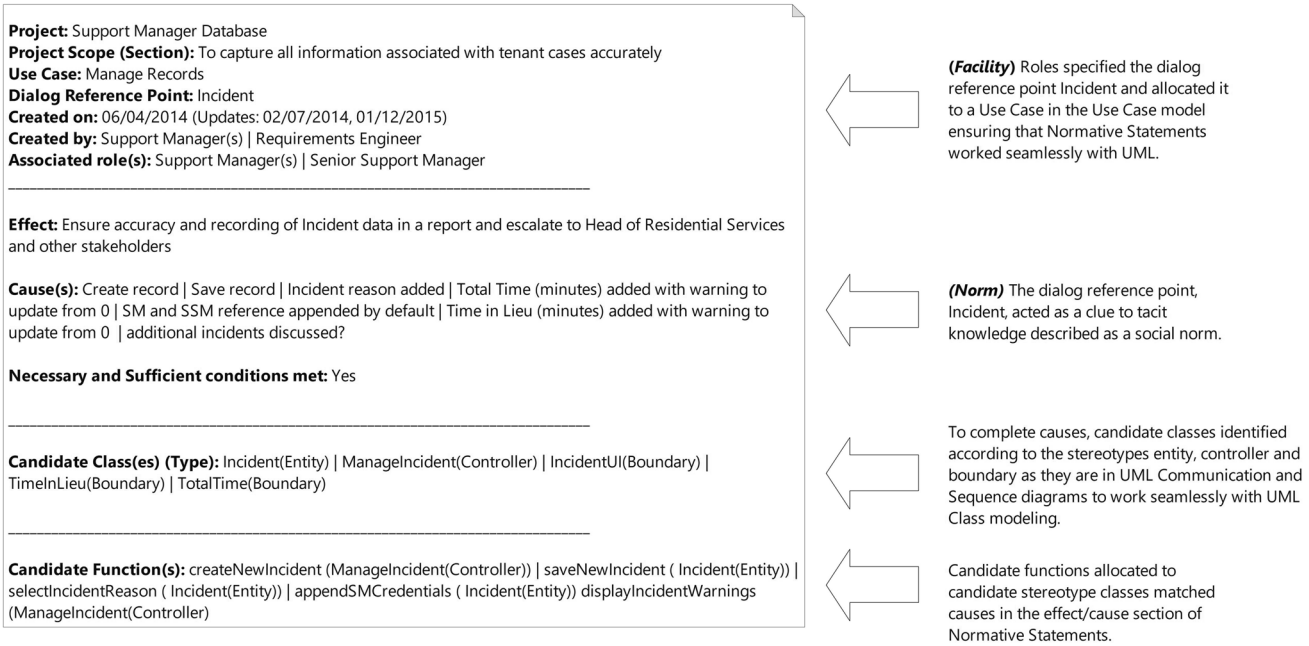

Fig. 5 Example normative statement as an interpretative scheme

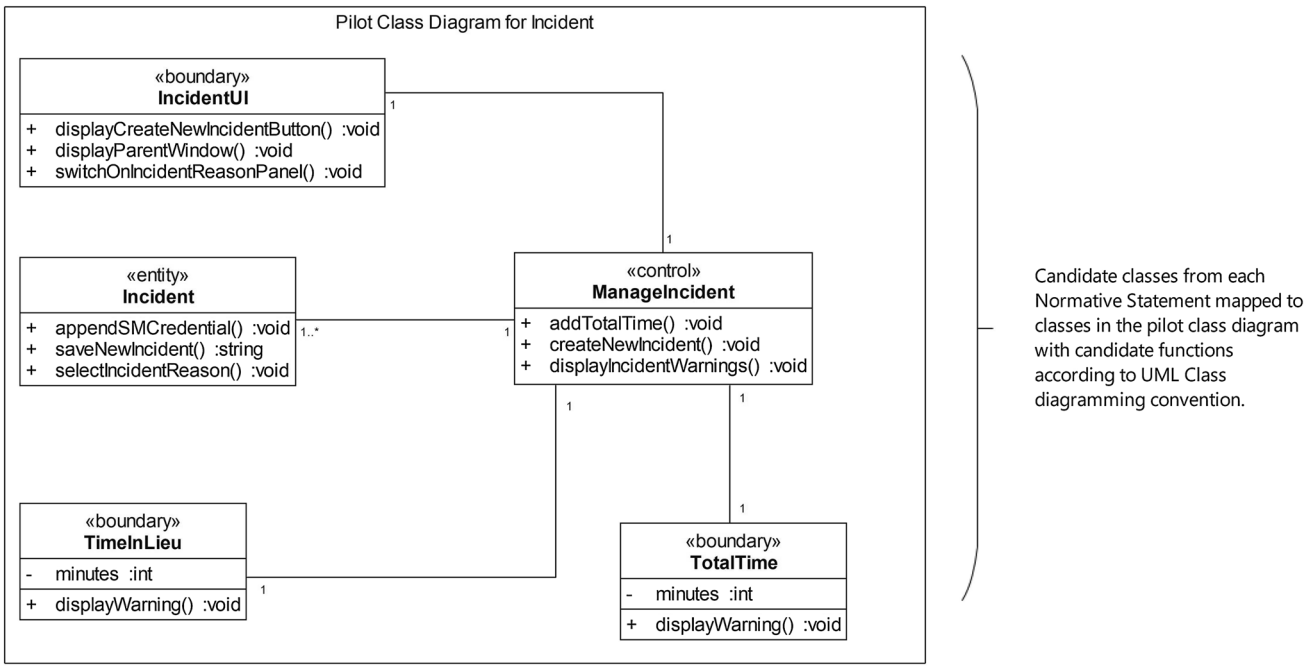

Fig. 6 An example normative statement and linked pilot class model

Once all end-users agreed to the UML Use case diagram, the next stage of the requirements analysis phase considered Normative statements. For each dialog reference point shown in the Use case diagram, a Normative statement was compiled using a structured document based upon Fig. 5. The UML CASE Tool managed the document for each Normative statement, it linked each document to corresponding dialog reference points in the UML Use case model. Each Normative statement was discussed in semi-structured one-to-one meetings with end-users. The requirements engineer first visited the Head of Residential Services and Senior Support Manager to corroborate the scope of the

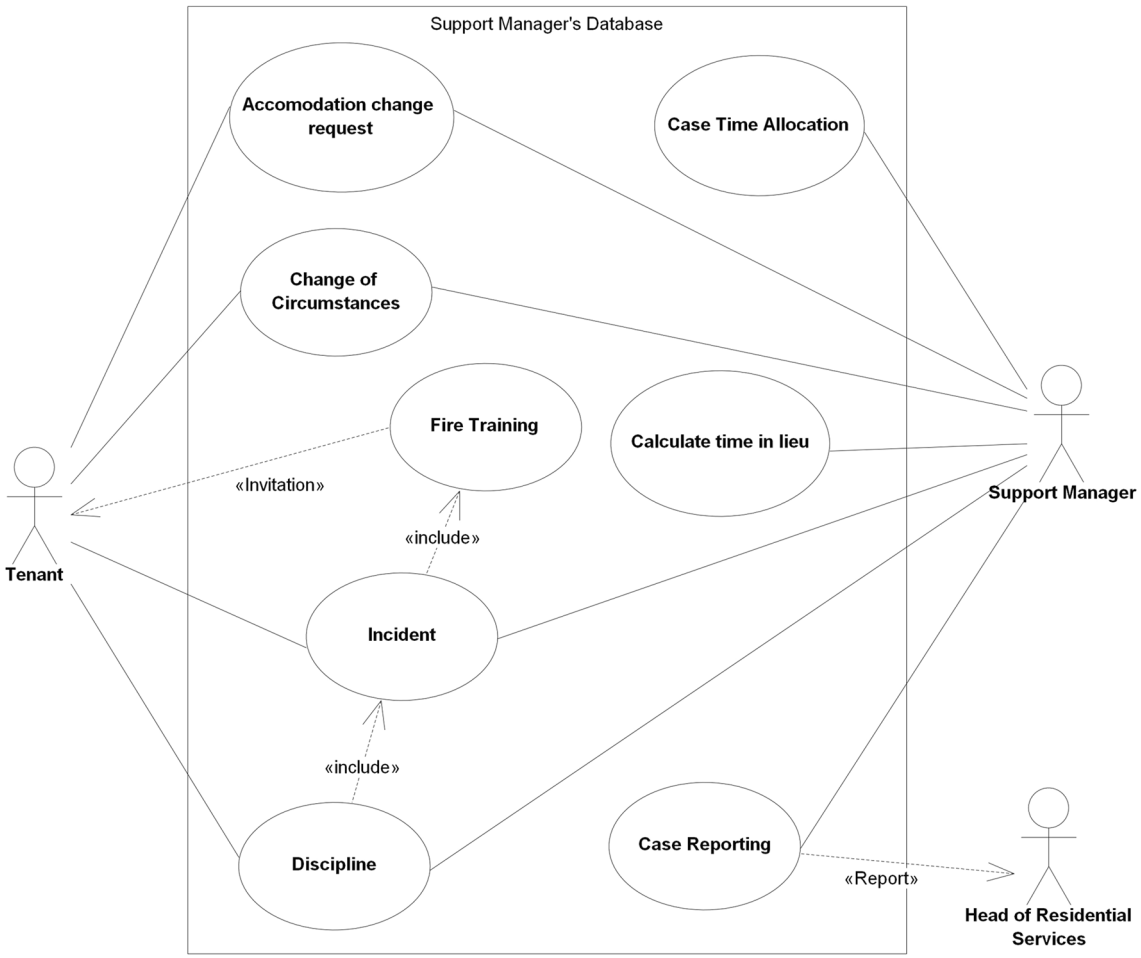

Fig. 7 Basic UML diagram to show dialog reference points

project then organized, with each of the remaining Support Managers, one meeting each. During all meetings, end-users agreed to effects first and then individually stated what they believed to be causes. The requirements engineer assembled the list of causes and effect(s) for each dialog reference point and confirmed these with all end-users before a team meeting, the final requirements and design review. Table 4 shows a comparison between the original requirements analysis phase of the SupportManagerDB project, and the one after following the DSRM. Table 4 provides evidence that the

Table 4 Comparison between original and revised requirements modeling phase

Phase (original)	Revised (after the DSRM)
1. Initial scope of requirements	Initial scope of requirements
2. Use case modeling included: narrative for each use case (11 use cases) Activity model for complex use cases (for 8 use cases—8 activity models) Communication model for each use case (for 11 use cases) according to UML stereotypes Pilot class model (merging of 11 communication models)	Basic use case model which confirmed dialog reference points Normative statement model. (8 Normative statements) Pilot class model achieved by merging class diagrams from each normative statement
3. Class modeling	Detailed class modeling
4. Detailed class modeling: finalized class model, relational database model, and C# skeleton code generation	Relational database model C# skeleton code generation and code completion
5. First phase deployment of project	Full working software

DSRM and its research process with links to its knowledge base helped to reduce the effort associated with requirements modeling while capturing and using tacit knowledge in the requirements model for the SupportManagerDB project.

For Objective 2, Normative statements worked with UML Use case modeling and linked seamlessly with design, specifically with UML [51] Class modeling. Figure 8 shows that Normative statements bypassed the need for modeling requirements further using UML Activity and Communication diagrams. Figure 8 also shows how Normative statements bridged requirements modeling from the UML Use case model to the Class model ready for detailed design.

Regarding Objective 3, the functional requirements modeling technique had to enable requirements to be modeled speedily as end-users agreed to check requirements but also insisted on the deployment of a revised software system at the end of year two in the SupportManagerDB project. Normative statements used for functional requirements modeling within the SupportManagerDB project, although they have an element of framing, the formalism applied permitted functional requirements modeling to quickly advance. They made obtaining and using tacit knowledge more effective and sped up the functional requirements modeling process. A new software system deployed in the second year of the SupportManagerDB project met the conditions set out in phase 2 of the DSRM. Support Manager D remarked:

> I didn't like the way the software looked or worked when you first showed us what you had programmed. I know I agreed to the diagrams you had created for us but I didn't understand them. I stopped using your first database as it didn't work for me and went back to paper versions. I admit that I also felt nervous about using the software you created as I don't feel confident. I understand the new things you did [Normative statements] and feel more confident with the software. I learned more about how [Support Managers] work and that the way they work is very similar to me. I have

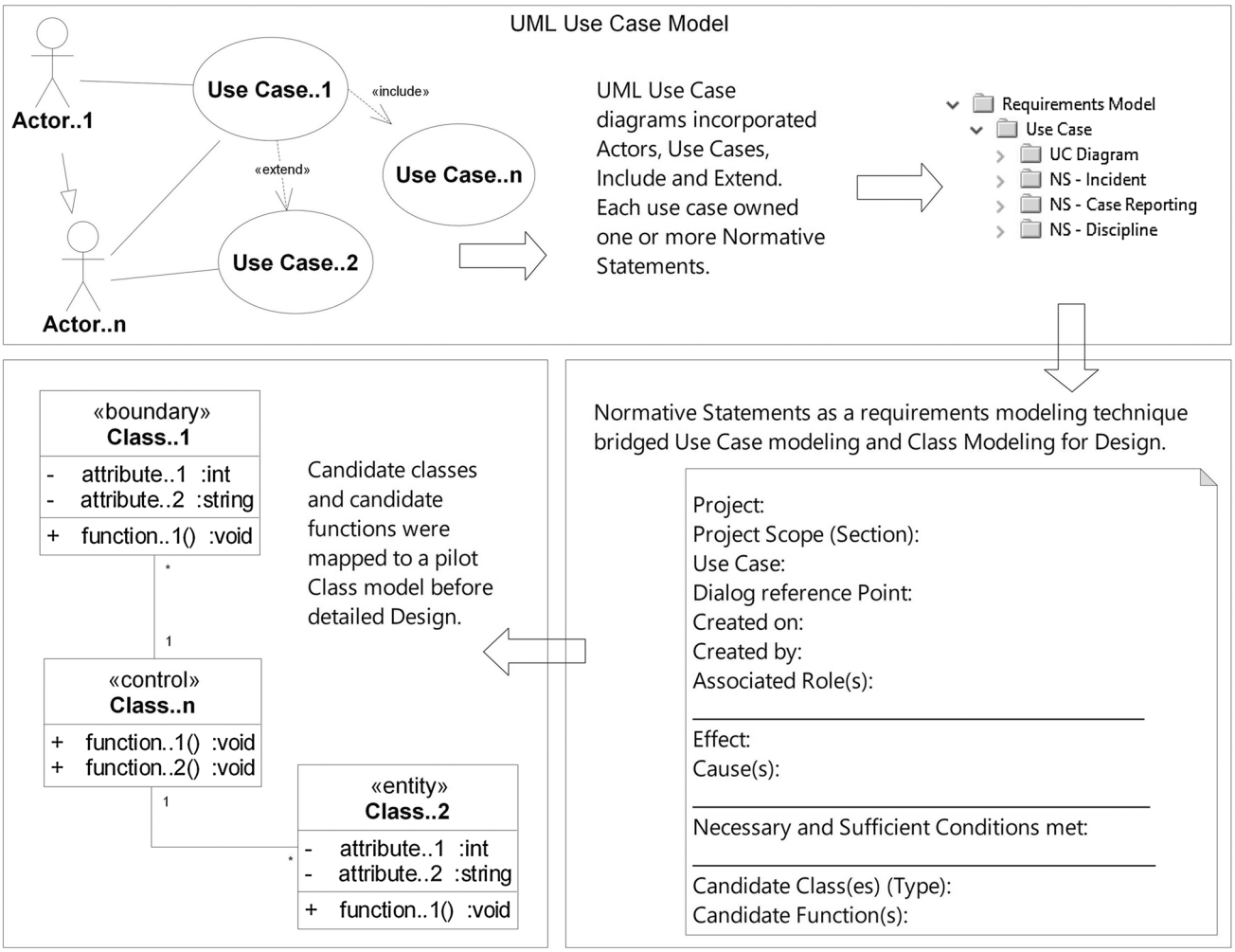

Fig. 8 Combining normative statements with UML

also had to adjust some of things I do for the better but I have now used the software and it supports what I am doing.

Following the DSRM showed that dialog reference points help to unlock tacit knowledge while maintaining the user-centeredness of the requirements analysis phase in the SupportManagerDB project. Hence, the design science research project shows that effective understanding in the functional requirements modeling process was possible when using Normative statements to reveal and model tacit knowledge. Normative statements crystallized tacit knowledge aligned to the three pillars in structuration theory when end-users discussed them in one-to-one and team meetings. From a semiotic viewpoint, signification helped to comprehend the use of Normative statements exchanged in communication and meanings inferred from them.

This study has limitations however. First, based upon the triangulation of data to act as the underlying method to establish validity and to achieve consistency could not include interviews related to end-user views of the requirements analysis stage. The researcher intended to collect these views, but at the end of the SupportManagerDB project, team members dispersed to other roles within TenantManagement and the researcher had difficulty organizing interviews. Hence, the empirical methods employed included archived paperwork associated with the SupportManagerDB project collected as functional requirements and design models, coding specifications, one-to-one and team meeting agendas, notes taken and emails, and formal records of meetings. Second, this study also focused upon the issues with modeling functional requirements based upon a team of twelve people involved in a three-and-half-year project with room to allow the DSRM to run in parallel. Although the project lasted for three-and-half-years, only the Head of Residential Services, the Senior Support Manager and two Support Managers remained throughout. A further threat to reliability connects to the combined role of requirements engineer and researcher. To mitigate the requirements engineer role, the two IT support department members that understood UML and programming provided checks related to the application of UML during the SupportManagerDB project. A further risk to reliability relates to the knowledge base presented in Fig. 3 as the researcher took sole responsibility for its development which can introduce bias and limitations. First, to lessen this threat in relation to the application of structuration theory, the knowledge base is limited to the basic tenets of the structure—agency relationship. A significant amount of work related to this relationship is used in information systems research, hence the knowledge base was designed not to deviate from the known application of structuration theory [32]. Regarding Peircean semiotics, the interim account includes several agreed and affirmed sign classifications [2, 13, 20, 27, 40–42, 49]. The researcher ensured consistency with those agreed classifications only.

Design science research methodologies must be theoretically, internally and empirically grounded [27]. The purpose of this type of grounding provides researchers with the case that applying the DSRM increases researcher confidence. For theoretical grounding, the knowledge base used in this paper incorporated widely understood functional requirements techniques based upon UML, and structuration and semiotic theories, also used extensively [32]. Internal grounding means to have logical consistency [18], a clearly described and reusable process. The stages the DSRM applied for this research provided the apparatus for logical consistency. To show the empirical grounding of DSRM, the application of the methodology was successful in practice, it changed a difficult situation into an improved one and resulted in the desired outcomes being achieved [39].

The knowledge base in the DSRM, inclusive of structuration and semiotic theories matched the situation extant in the case study, thus generalizations are only viable when linked to similar types of software development projects as it is documented in phase 1 of the DSRM for this paper. The study also has wider implications to contemporary functional requirements modeling associated with the objectives of the solution in phase 2 of the DSRM. For example, agile software development aims to create deployment success by using cross-functional project teams that apply user stories [1]. User stories are short statements that follow a < role > , a < requirement or feature > , and a < goal/value > structure [1]. The proposed Normative statements incorporate a different structure to identify tacit knowledge and matching the format, < role > , a < requirement or feature > , and a < goal/value > structure would not capture and model tacit knowledge. Also, Normative statements show which requirements link specifically to design classes, possibly connecting software code more effectively to functional requirements than user stories. However, this is an area for further research allied to the iterative nature of the DSRM.

The implications of this design science research relate to modeling tacit knowledge. This paper emphasizes that uncovering tacit knowledge is a key component when specifying new software and answers the RQ, how can tacit knowledge be obtained and managed in order to contribute to functional requirements modeling? The RQ is resolved by using structuration and semiotic theories as the knowledge base in the DSRM to revise functional requirements modeling. These theories advance the discussion related to understanding and managing tacit knowledge on software development projects [5, 21, 45, 50]. For instance, the modality *interpretive scheme* helped to identify structures of signification as dialog reference points that influenced the understanding of tacit knowledge in processes of semiosis. Uniting with the modalities *norm* and *facility* to appreciate

structures of signification, they helped to generate Normative statements which included a component based upon social norms. *Norm* and *facility* helped to identify tacit knowledge legitimized in power relationships between end-users. Regarding Peircean semiotics, the third grade of clarity and interpretant signs, Normative statements surfaced as an alternative functional requirements modeling technique for tacit knowledge.

Regarding iteration as part of the DSRM [39], and for further research, Normative statements need further investigation within other software development projects. For example, Peircean semiotics provided the grades of clarity when identifying tacit knowledge and intellectualized Normative statements as interpretant signs. However, three accounts of Peircean semiotics exist, namely early, interim and final [2, 40–42]. The early account provides underpinning to the interim account, and the final account although considered incomplete, adds other features to the interim account. Using the interim account, Peircean semiotics identifies the concept of a triadic sign integrating the three elements, representamen (the sign), object and interpretant. Understood by shared properties, a representamen signifies an object by conveying something about an object. Interpretant signs represent the concept of mediation, whereby the representamen and object form a relation that divides interpretant signs. Based upon this foundation, the Peircean view of semiotics in the interim and final accounts may generate a different set of signs beyond the textual structure used in Normative statements, that with further research, other stages of software development inclusive of new modeling techniques may align to.

8 Conclusion

The implications of this research relate specifically to the knowledge base described and presented in Fig. 2, the application of DSRM, and the creation of Normative statements. The knowledge base emphasizes that the utilization of tacit knowledge improves functional requirements modeling. The application of the DSRM acted as a stimulus to the knowledge base, and the knowledge base supported the DSRM with appropriate theory. It shows that structuration and semiotic theories can be closely coupled with functional requirements modeling. Principally however, this study contributes to challenges when it is difficult to capture and model tacit knowledge. Without tacit knowledge, functional requirements quality is lower and has implications for the design and coding of software. The research in this paper also discovered that when using tools such as the UML, they create "competitive vocabularies", also inhibiting the management of tacit knowledge. Normative statements are also an alternative functional requirements modeling technique to the ones available in UML such as Activity, Communication, and Sequence diagrams. They shortened the requirements analysis phase of the SupportManagerDB project, and might do so in similar software development contexts.

References

1. Agile (2018) http://www.agilemodeling.com/artifacts/userStory.htm. Accessed 25 Oct 2018
2. Atkin A (2018) Peirce's theory of signs. http://plato.stanford.edu/archives/sum2013/entries/peirce-semiotics/. Accessed Oct 2018
3. Avison DE, Fitzgerald G (2006) Information systems development: methodologies, techniques and tools, 4th edn. McGraw-Hill Maidenhead, Maidenhead
4. Avison DE, Wood-Harper AT (1990) Multiview: an exploration in information systems development. Alfred Waller Limited, Oxfordshire
5. Bagheri S, Kusters RJ, Trienekens JJ (2017) Eliciting end users requirements of a supportive system for tacit knowledge management processes in value networks: a Delphi study. In: Engineering, technology and innovation (ICE/ITMC). IEEE, pp 1317–1326
6. Baskerville RL (1999) Investigating information systems with design science research. Commun AIS 2(3es):4
7. Baskerville R, Myers MD (2004) Special issue on design science research in information systems: making IS research relevant to practice: foreword. MIS Q 28:329–335
8. Bano M, Zowghi D (2015) A systematic review on the relationship between user involvement and system success. Inf Softw Technol 58:148–169
9. Bicchieri C (2006) The grammar of society. Cambridge University Press, Cambridge
10. Bjarnason E, Sharp H (2017) The role of distances in requirements communication: a case study. Requir Eng 22(1):1–26
11. Brennan A (2017) Necessary and sufficient conditions, the stanford encyclopedia of philosophy (summer 2017 edition). In: Edward N Zalta (ed). https://plato.stanford.edu/archives/sum2017/entries/necessary-sufficient. Accessed 10 Oct 2018
12. Brinkkemper S (1996) Method engineering: engineering of information systems development methods and tools. Inf Softw Technol 38(4):275–280
13. Chandler D (2017) Semiotics: the basics. Routledge, Abingdon
14. Charlton B, Andras P (2003) The modernization imperative, vol 8. Imprint Academic, Exeter
15. Chugh R (2015) Do Australian universities encourage tacit knowledge transfer? In: Proceedings of the 7th international joint conference on knowledge discovery, knowledge engineering and

knowledge management (IC3K 2015), vol 3. KMIS, pp 128–135. ISBN: 978-989-758-158-8
16. Cockburn A (2002) Agile software development, vol 177. Addison-Wesley, Boston
17. Collins H (2010) Tacit and explicit knowledge. University of Chicago Press, Chicago
18. Cronholm S, Göbel H (2015) Empirical grounding of design science research methodology. In: International conference on design science research in information systems. Springer, Cham, pp 471–478
19. DSDM (2018) The DSDM agile project framework (2014 onwards). https://www.agilebusiness.org/content/choosing-dsdm-your-agile-approach-0. Accessed 10 Oct 2018
20. Eco U (1976) A theory of semiotics, vol 217. Indiana University Press, Bloomington
21. Ferrari A, Spoletini P, Gnesi S (2016) Ambiguity and tacit knowledge in requirements elicitation interviews. Requir Eng 21(3):333–355
22. Fernández DM, Wagner S (2015) Naming the pain in requirements modeling: a design for a global family of surveys and first results from Germany. Inf Softw Technol 57:616–643
23. Gane CP, Sarson T (1979) Structured systems analysis: tools and techniques. Prentice Hall Professional Technical Reference, Upper Saddle River
24. Geerts GL (2011) A design science research methodology and its application to accounting information systems research. Int J Account Inf Syst 12(2):142–151
25. Giddens A (1984) The constitution of society: outline of the theory of structuration. University of California Press, Berkeley and Los Angeles
26. Goffin K, Koners U (2011) Tacit knowledge, lessons learnt, and new product development. J Prod Innov Manag 28(2):300–318
27. Goldkuhl G (2004) Design theories in information systems-a need for multi-grounding. J Inf Technol Theory Appl (JITTA) 6(2):7
28. Hevner AR, March ST, Park J (2004) Design research in information systems research. MIS Q 28(1):75–105
29. Hofmann HF, Lehner F (2001) Requirements engineering as a success factor in software projects. IEEE Softw 18(4):58–66
30. Jacobson I, Booch G, Rumbaugh J (1999) The unified software development process, vol 1. Addison-Wesley, Reading
31. Jia J, Capretz LF (2017) Direct and mediating influences of user-developer perception gaps in requirements understanding on user participation. Requir Eng 23:1–14
32. Jones MR, Karsten H (2008) Giddens's structuration theory and information systems research. MIS Q 32(1):127–157
33. Loucopoulos P, Karakostas V (1995) System requirements modeling. McGraw-Hill Inc., New York
34. Luhmann N (2012) Introduction to systems theory. Polity Press, Cambridge
35. Morris CW (1938) Foundations of the theory of signs. In: Neurath O, Carnap R, Morris CFW (eds) International encyclopedia of unified science. Chicago University Press, Chicago, pp 1–59
36. Mumford E (1995) Effective systems design and requirements analysis: the ETHICS approach. Macmillan, New York City
37. Nonaka I (1994) A dynamic theory of organizational knowledge creation. Organ Sci 5(1):14–37
38. Nuseibeh B, Easterbrook S (2000) Requirements modeling: a roadmap. In: Proceedings of the conference on the future of software engineering. ACM, pp 35–46
39. Peffers K, Tuunanen T, Rothenberger MA, Chatterjee S (2007) A design science research methodology for information systems research. J Manag Inf Syst 24(3):45–77
40. Peirce CS (1932) The categories in detail. In: Hartshorne C, Weiss P (eds) Collected papers of Charles Sanders Peirce. Volumes I and II: Principles of philosophy and elements of logic, Book III, Chap 2. Harvard University Press, Cambridge, MA
41. Peirce CS (1933) The logic of relations. In: Hartshorne C, Weiss P (eds) Collected papers of Charles Sanders Peirce. Volumes III and IV: Exact logic (published papers) and the simplest mathematics, Chap XVI. Harvard University Press, Cambridge, MA
42. Peirce CS (1998) The essential writings: selected philosophical writings (1893–1913), 1st edn. Indiana University Press, Bloomington
43. Pohl K (2010) Requirements modeling: fundamentals, principles, and techniques. Springer, Berlin
44. Polanyi M (1958) The tacit dimension. University of Chicago Press, Chicago
45. Serna E, Bachiller O, Serna A (2017) Knowledge meaning and management in requirements engineering. Int J Inf Manag 37(3):155–161
46. Saussure F de (1983) Course in general linguistics (trans. Roy Harris). Duckworth, London
47. Short T (2007) Peirce's theory of signs, 1st edn. Cambridge University Press, Cambridge
48. Simon HA (1996) The sciences of the artificial. MIT Press, Cambridge
49. Sowa J (2000) Knowledge representation: logical, philosophical, and computational foundations. Brooks/Cole Thomson Learning, Pacific Grove
50. Sutcliffe A, Sawyer P, Stringer G, Couth S, Brown LJ, Gledson A, Bull C, Rayson P, Keane J, Zeng XJ, Leroi I (2018) Known and unknown requirements in healthcare. Requir Eng 25:1–20
51. UML (2018) Unified modeling language resource page. https://www.omg.org/spec/UML/2.5.1/. Accessed 10 Oct 2018
52. Walsham G (2002) Cross-cultural software production and use: a structurational analysis. MIS Q 26(4):359–380
53. Yagüe A, Garbajosa J, Díaz J, González E (2016) An exploratory study in communication in agile global software development. Comput Stand Interfaces 48:184–197