International Journal of Software Engineering
and Knowledge Engineering
Vol. 27, No. 1 (2017) 1750006 (27 pages)

DOI: 10.1142/S0218194017500061

World Scientific
www.worldscientific.com

A Design Theory for Cognitive Workflow Systems

Jarkko Hyysalo*,†, Markku Oivo*,‡ and Pasi Kuvaja*,§

*Faculty of Information Technology and Electrical Engineering, M3S
University of Oulu, P. O. Box 8000
FI-90014 Oulu, Finland
†jarkko.hyysalo@oulu.fi
‡markku.oivo@oulu.fi
§pasi.kuvaja@oulu.fi

Received 24 October 2014
Revised 22 November 2014
Accepted 10 February 2016
Published

This paper addresses the design problem of providing cognitive support for workflow systems in software development. Software development is demanding knowledge work that requires creativity and adaptability to changing requirements and situations. This type of work involves cognitive actions that require substantial support in several forms in order to address needs such as collaboration, communication, knowledge management, awareness and transparency, and the coordination and structuring of the development processes. The literature and our empirical results show that there is a lack of cognitive support in current workflow models. Hence, we identify the need for a design theory for cognitive workflow systems (CWS). In this paper, such a theory is presented. The proposed design theory for CWS is validated through an action research intervention. This design theory has important implications from both research and practical perspectives. The results will help developers in their daily work, enhance the efficiency of the development processes, and facilitate decision-making activities.

Keywords: Cognition support; cognitive work; design theory; knowledge work; software development; workflow; work support.

1. Introduction

Software development is increasingly challenging and intellectually demanding creative knowledge work [1]. It is a human- and knowledge-intensive activity in which managing knowledge is paramount [2]. Such information-intensive knowledge work is primarily a cognitive activity [3] based on the worker's internal mental processes rather than on physical labor. Cognitive perspectives are fundamental factors for successful collaboration [4]. Thus, knowledge work and cognitive aspects should be

†Corresponding author.

supported properly in order to obtain good results. Software systems are usually developed as a multidisciplinary effort [5], typically in collaboration with several types of stakeholders such as engineers, industrial designers, and marketing personnel [6, 7].

Workflows are used to support these collaborative processes. Here, we define a workflow as a sequence of working steps or logically-related tasks, including the use of resources to achieve a common goal — transforming a set of inputs into outputs that provide value for stakeholders. However, it is not easy to plan the work in detail beforehand [8], as software processes are often complicated in nature; they involve a large number of tasks, performers, and coordination constraints. Moreover, cognitive work and cooperation are not properly supported in current workflow models [9, 10].

A practical solution is needed for supporting cognitive work in collaborative development. Hence, we need to design a workflow that addresses the above-mentioned issues. As this is a design issue, we claim that the design theory approach proposed in [11] is an effective approach to address it. Bringing forth a design theory for a cognitive workflow system (CWS) is the main contribution of this work.

Building a workflow that supports cognitive work — intellectually demanding and creative knowledge work — requires both the development of a formal methodology that can help model the knowledge flows and an extension of current workflow technologies to handle information that is characterized by dynamic changes, requirements for innovative problem solving, and cognitive processing. The framework and formalisms, we propose in this paper serve as the foundation for such a methodology. Design theory has proven to be successful for the purposes of describing improved design processes [12]. We propose that the kernel and/or justificatory theories and extensive case studies resulting in the design theory provide a solid basis for prescribing the principles and requirements for designing and constructing an effective artifact.

The remainder of this paper is organized as follows. Section 2 discusses related studies, workflow modeling, and design theory and states the motivation for this research. Section 3 presents the research process and provides the theoretical framework for this research in the form of kernel theories. Further, it describes the results of empirical work. Section 4 evaluates how the implementation of the six design principles addresses the identified issues by analyzing the research findings, also including a comparison to existing approaches. Section 5 discusses the results, implications, and future research directions and concludes the paper.

2. Theoretical Background

This section presents the theoretical concepts related to workflow modeling in the context of knowledge work, discusses related studies, and states the motivation for this research. In addition, the design theory approach is briefly described.

2.1. *Cognitive work support in workflows*

Software development involves several challenges related to individual and team cognition, such as complex decision-making, innovative problem solving, handling of vast amounts of information, building of a shared understanding, and information and knowledge sharing [13]. The greater the flow of knowledge, knowledge sharing, and learning between different organizational parts and external agents, the greater are the opportunities for knowledge generation [14–16]. However, one of the problems in knowledge work is the number of disconnected tools, that is, tools that are disconnected not only from other tools but also from the work processes; moreover, knowledge is often disconnected from the context [17]. Thus, a workflow that supports knowledge sharing and enables the use of cognitive skills is required. The workflow must also be designed such that it addresses different views, including those of business analysts and software developers [18], and connects the knowledge to work, processes, and the context of work.

In this study, cognitive activities and processes include, for example, the following aspects of knowledge work: making observations; reasoning; processing information; learning, understanding, and remembering information content, using information systems and conceptualizing.

Support for cognitive work is crucial when the work concerns abstract matters and knowledge, and the following challenges are identified: knowledge is not easily transferred unless it is made explicit, knowledge elements are context-specific, and cooperation is needed because of humans' cognitive limitations — one does not know everything [19]. Cognitive skills are also required to respond to changes, that is, creativity and human problem-solving skills are required to deal with changes and unexpected events [20].

It has been proposed that the purpose of collaborative software development environments is to facilitate and nurture developers' creative knowledge processes [21]. For this purpose, for example, companies implement workflows to help manage processes, transfer work and data from one worker to another, and establish a logical order for task implementation. However, it has been acknowledged that traditional workflow approaches are too static and do not address the changes and unexpected events that inevitably occur in creative knowledge work; moreover, workflow models lack cognitive support [9, 22–25]. Nor are cognitive work and cooperation properly supported in current workflow models [9, 10].

Hence, there is a need for workflow design principles to provide guidance on how to model and implement a CWS that supports software development processes in which uncertainty prevails and how knowledge flows can be utilized to create a solid basis for decision-making activities. We suggest that most cognitive challenges can be tackled with properly designed cognitive workflows that address collaboration and communication practices, implicit and explicit knowledge management, awareness and transparency, and processes supporting coordination and negotiation.

Abstracting and synthesizing from related works, we identified a set of characteristics of software development, which we classified into six categories: cognitive work support, collaborative work, communication, knowledge management, awareness and transparency, and coordination. Table 1 presents a brief summary of this categorization. Then, we mapped each characteristic to a design principle for CWS and identified its requirements relevant to a workflow system.

Other characteristics and requirements could also be identified; however, this work addressed the research problem from these six viewpoints, all of which complement each other and contribute to defining cognitive support for collaborative work. Many other concepts and viewpoints can also be seen as relating to CWS, such as distributed work, global software development, decision-support systems, virtual collaboration tools, learning, and computer-supported cooperative work. However, this work did not address these areas individually but, rather, focused on viewing workflow systems from a cognitive point of view.

Table 1. Summary of characteristics and their system support requirements.

Software development characteristics	Requirements for workflow systems	Design principles for CWS
Complex tasks, knowledge-intensive processes, abstract knowledge	Supporting understanding, shared understanding, and reducing cognitive load	Cognitive work support
Different kinds of tasks, different expertize areas, distributed work, team work	Supporting collaboration, distributed interactions, and realization of business processes	Collaborative work
Information exchange, understanding of each other	Creating and maintaining shared understanding	Communication
Knowledge-intensive processes, team knowledge, knowledge management	Acquiring, sharing, and using knowledge	Knowledge management
Understanding of others' activities, context, information visualization, situation awareness, workspace awareness	Enabling knowledge, supporting awareness and transparency	Awareness and transparency
Simultaneous, sequential, distributed	Creating structure, synchronization, integrating contributions, decision points	Coordination

2.2. *Design theory approach*

Design theory has proven to be successful for the purposes of describing improved design processes [12], and there are examples of successful attempts to use the design theory approach; see [17, 26–29]. This section briefly describes the concept. For a more detailed description, see, for example, [11, 12, 30–32], where the use of design theory is discussed extensively.

Design theory describes components and their relationships for subsystems that are then used to construct the system [11]. In other words, the "pieces" can be designed relatively independent of one another; however, they eventually form a complete system in which the subsystems are not completely independent. Thus, all the pieces and the entire system must be satisfactorily constructed — this is the goal of design theory. Another important aspect of design theory is that it deals with both the product and the process of design; these two cannot be completely isolated, because the process eventually yields the product [11].

Design theory consists of product and process aspects, and for both aspects, there are kernel theories that describe the fundamental behavior underlying the concepts, for example, the laws of nature. Meta-requirements are derived from the kernel theories, and they describe a class of goals to which the theory is applicable [11]. Based on the meta-requirements, meta-design principles are drawn up to describe the class of artifacts that are hypothesized to meet the meta-requirements. Finally, there are testable design hypotheses for verifying whether the meta-design principles meet the meta-requirements. However, it should be noted that context-specific modifications are required to accommodate domain-specific aspects.

Walls *et al.* [11] formally specified and coined the term, "design theory," and others have also used and modified the approach. Thus, there are several useable ways of presenting design theories. The original approach by Walls *et al.* [11] has been reviewed in literature, and there are considerations that it might initially have been too cumbersome to use (cf. [29, 33]).

It has been argued that testable design hypotheses are unnecessary and not essential to the theory, as design theory explains generalized solution components with the related generalized requirements or design principles [31]. Gregor and Jones [30] also agree, and they suggest that even without testable hypotheses and empirical indicators, design theory is sufficient to describe the idea of an artifact that could be constructed. There are also other examples where the testable hypotheses are not seen as necessary components of design theory; see, for example, [29, 32].

Building the design theory in this article follows the approach presented in Hanseth and Lyytinen [34]. Hanseth and Lyytinen also base their approach on Walls *et al.* [11], but they have made small changes; for example, they have omitted the use of testable hypotheses.

Table 2. Components of a design theory [32].

Design goals	Describe the goals to which the theory applies.
A set of system features	A set of artifacts hypothesized to meet the requirements.
Kernel theories	Theories or justificatory knowledge from natural and social sciences governing the design requirements or the processes for arriving at them.
Design principles	A codification of procedures, which, when applied, increase the likelihood of achieving a set of system features. These procedures are derived from kernel theories.

In all, there are different approaches for design theory, as discussed in [32]; the authors have summarized the interrelated elements of design theory as shown in Table 2.

In addressing these elements, a set of guidelines is defined for designers to design an effective artifact, justified by accepted theories or justificatory knowledge [28].

3. Industrial Case and Development of a Design Theory for CWS

This section introduces the study that prompted the development of a new design theory for CWS. The research methods and the evolution of design principles are presented.

3.1. *Research process*

Various cognitive issues have been identified in the literature, as discussed in Sec. 2. A study was conducted to understand the practical problems currently faced by software development companies. In addition to identifying industry-relevant issues, this study provided an opportunity to verify the meta-requirements for CWS and to develop a research prototype for validating the new design theory for CWS.

An action research intervention within a large global company was set up to define and validate the design theory for CWS, and the empirical work was carried out via the action research intervention. Action research is an approach for putting theories into practice in order to help companies solve concrete problems while expanding scientific knowledge at the same time [34]. Action research is an iterative process whereby theory-based diagnosis is followed by practical intervention through action planning, action taking, and evaluation. Learning from action and evaluation should result in change, and the action research cycle is repeated until satisfactory results are obtained. Action research was applied to put the developed model into practice, to improve it, and to evaluate the results. Evaluations were made by implementing a workflow prototype on the basis of the proposed design principles and applying it in its intended settings, thus providing empirical evidence on its use.

The study was conducted at a large global company that provides systems and services, one of the largest information and communication technology companies in the world. Most of the company's products are considered software-intensive systems, which are developed as a parallel and interconnected set of processes run by various stakeholders.

Our research began with a pre-study, where existing literature was first analyzed to identify the need for CWS. Relevant kernel theories were selected, and the initial goals for design theory for CWS were then developed. This work provided the initial requirements and design principles. The case company was studied to understand the state of its practice; company material was studied and weekly workshops were arranged to understand practical needs and issues. Interview questions were

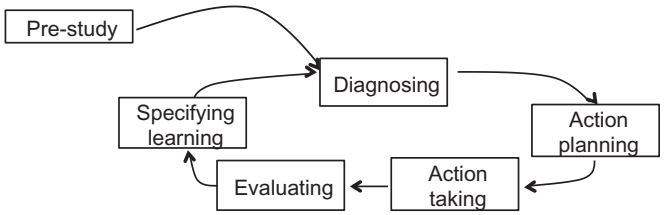

Fig. 1. Action research cycle with pre-study (adapted from [35]).

formulated based on the understanding achieved. The pre-study phase was iterative in nature, to ensure a solid basis for subsequent actions.

A total of 35 qualitative interviews were conducted over a period of 18 months at several sites in several countries. Interviewees represented various positions, from "floor level" to mid-management. One of the interviewees was a service capability manager, 14 were development team leaders with at least 15 years of work experience, and one was the head of the development team leaders. The rest were experienced developers with 10–20 years of work experience. Altogether, 20 different organizational units were represented; they covered several aspects of the products' life cycle. Different questionnaires were used, as the interviews were targeted at each phase with relevant questions concerning the interviewees' work.

After the pre-study, the action research cycle was initiated. The action research cycle, shown in Fig. 1, was repeated three times, until the results were satisfactory.

In the action research intervention, a prototype workflow system was designed and developed in order to evaluate its use in practice. During the action research, the design principles and the developed system were iteratively refined until satisfactory results were obtained. Eight developers from the case company participated in the prototype evaluations. Their roles in the company were as development team leaders (4), and the rest were specialists (4). All of them had considerable experience in practical work (10–20 years of work experience). Most of them had already participated in previous interviews.

In the action research intervention, the implementation of the prototype and its integration into the case company's environment was carried out in order to evaluate iteratively the prototype and the design theory for CWS and its assumptions. During the action research intervention, the requirements, design principles, and prototype were refined.

3.2. *Toward design principles for CWS*

Next, we describe our design theory for CWS as a set of six design principles. Each design principle is presented in terms of how it stems from kernel theories. Several kernel theories that relate to the design goals and desired system features could be

identified. The desired system features are based on the literature, kernel theories, and practical industrial needs, as discussed in Sec. 2.1. Practical needs ensure that the problems and solutions are relevant to the industry. We also show how the initial needs evolved into final, revised requirements and design principles, which concern the topics of development environment, cooperation and distributed work, communication, information and knowledge, transparency and awareness, and structure and support for development processes.

First, each design principle is briefly presented along with related kernel theories. Then, the required system features are defined. Next, industrial experiences of current workflows and methods are presented along with their shortcomings. Finally, the evaluation of the workflow prototype designed according to our design principles to validate the design theory for CWS is discussed, based on analyses of industrial interviews and observations.

Principle 1. Design methods must develop a workflow that provides cognitive support

This design principle is an essential goal that distinguishes the design theory for CWS from other design methods. It broadly defines the overall goal of the new design theory for CWS. In a sense, it is a definitional principle that all modern workflow systems should address. However, cognitive work and cooperation are not properly supported in current workflow models [9, 22–25]. Hence, the need for design Principle 1 that addresses the requirements for supporting understanding, shared understanding, and reduction of cognitive load.

Kernel theories: Design Principle 1 stems from cognitive workflows, which are discussed in [9, 10]; it focuses mainly on the importance of providing cognitive support and knowledge flows for collaborative and distributed work. In addition, a theory of distributed cognition is proposed [36]. A cognitive framework for reengineering knowledge-intensive processes is also set forth, and it is proposed that developers' cognitive work is the result of interactions among individuals, artifacts, and resources in their environment [19]. Support for cognitive work is required when working with abstract knowledge, in order to meet various challenges. The following challenges are identified: knowledge is not easily transferred unless it is made explicit, knowledge elements are context-specific, and cooperation is required because of the cognitive limitations of humans, one cannot know everything [19].

System features: Cognitive workflows must create a common, flexible development environment that reduces developers' cognitive load and supports cognitive activities and processes. Software development is a type of demanding knowledge work that can be characterized as cognitive activity. It has also been argued [4] that cognitive perspectives are fundamental for successful collaboration. As this type of knowledge work is based on the internal mental processes of developers, it requires cognitive support.

Experiences from case company: Initially, we began with an idea to create a common, flexible development environment that provides data and guidance for

development tasks. However, the pre-study and interviews promptly showed that there was a need for a more thorough study, and the focus was then adjusted toward cognitive support. Interviews revealed that the needs and level of cognitive support varied among developers with different skills and experiences. Our research also promptly revealed that there were several issues that needed to be tackled. In general, the development environments were inadequate for complex knowledge work. An empirical study revealed several issues with tools, data, and other work items, which imposed an additional cognitive load on developers. These challenges were identified with regard to the following topics: communication and collaboration practices, knowledge management and coordination, and transparent tools and processes. A question was raised as to how solutions to these challenges may be integrated into a workflow that structures and supports the development process. These issues are further elaborated in discussions on design Principles 2–6. In summary, the cognitive activities and cognitive processes that must be addressed include making observations; reasoning; processing information; learning, understanding, and remembering information content; using information systems; and conceptualizing. The action research intervention proved that it is useful to have a common development environment and flexible cognitive workflow, so that collaboratively processed data become more accurate and more complete. It is proposed that principles should be provided to properly build a development environment with cognitive support.

Principle 2. Development of a workflow system should address the needs of collaborative work

This design principle proposes that the workflow system should have collaboration supporting facilities. By definition, workflows should help software development companies to transform a set of inputs into outputs that provide value for stakeholders, whereas collaborative workflows "connect" several persons and transfer data, documents, work, etc., among them. However, problems emerge as human collaborative activities cannot be fully predicted or anticipated [37]. We identified the need to address the design principle of collaborative software development, which stems from such characteristics as different kinds of tasks, different areas of expertize required, the necessity for distributed work, and teamwork. Design Principle 2 addresses the requirements for supporting collaboration, distributed interactions, and realization of business processes.

Kernel theories: Collaborative work has been discussed in [38], the authors of which have studied the dimensions of the problems of global software development; they suggest that the solutions that would help globally-distributed development teams would also help developers that work more closely with one another. Moreover, they suggest that the development sites should be able to operate as independently as possible, with the support of flexible and effective technological solutions. Several aspects of collaboration could be supported with information technology, and the selected kernel theories provide a starting point for defining the

system features and design principles of a CWS. When the focus is on collaborative workflows, the workflows should manage and coordinate the distributed interactions and facilitate the realization of the business processes [37].

System features: Cognitive workflows must foster cooperation and distributed work. In particular, workflow systems must be flexible in collaborative settings and cater to the definitions of change and dynamic process. In addition, they must support the independent work of development teams.

Processes, activities, and tasks must be mapped to high-level expectations (purposes, goals, and objectives) so that each activity and task belongs to a context, and tasks and work items are connected to others via a network of dependencies, thus creating the context for one's work.

Experiences from case company: The interviews and action research intervention indicated that even though the case company has defined and documented its processes, several developers had no proper overall view of the processes or development phases. Alarmingly, in some cases, developers did not know whether there was an official process at all. Further, not all organizational units were known, and in some cases, interviewees viewed cooperation possibilities as completely lacking. Cooperation — or the lack of it — among different developers, teams, and development phases indicated that apart from different actors having, in some cases, different perspectives, there were varying understandings of a product's life cycle. In particular, the developers described cooperation among the phases of a development chain, with their focus and interests being restricted to their own work phase and the phases immediately preceding and following theirs, instead of considering a product's entire life cycle. As a result, the developers' own work was not put into the context of the entire development chain, and the dependencies were not followed all the way through. Thus, the real needs of processes and phases were not considered adequately, nor were the relationships between processes and phases. Furthermore, the purposes of different processes and phases were not recognized.

There was a clear need to foster cooperation and communication. Serious problems arose as a result of inadequate cooperation and communication, including an insufficient understanding of what the others' activities required. To address the inadequacies in cooperation and communication, communication channels were requested to be integrated into the workflow system to support social and organizational cognition. The prototype also enabled developers to access documents and development items by using a simple "invite to collaborate" functionality. Simultaneous work was enabled by allowing others to access the entire document instead of only those data fields that were currently being manipulated. In addition, the prototype also allowed users to see what others were doing, and this information was constantly updated and made visible.

The workflow system that was implemented according to our design principle provided information about the goals and objectives of the company's processes. For example, different phases were handled by describing the dependencies and relationships between different processes and development phases as well as the data

flows between them. The activities in the development process were justified, and their purposes were clearly defined. This provided the work context and enabled developers to achieve full comprehension of the entire work system with all its purposes, goals, and objectives.

The action research intervention proved that enhanced communication and cooperation resulted in developers gaining a better understanding of one another's needs and work. With the prototype, the visibility of others' work was enhanced, as was the understanding of the different phases or development processes and their relationships and purposes, thereby enabling the developers to formulate a holistic understanding of the entire context. This enabled the developers to understand the impact of their work and its results on others' work.

Principle 3. Development of a workflow should address communication This design principle focuses on the communication aspects of workflow. In collaborative development, especially when the work is geographically distributed, it is essential to have communication tools. There is a need for different types of applications, including instant messaging and video conferencing, and interfaces for other types of tools should also be integrated into the system to provide instant access to work items. The most convenient approach is to have those applications integrated into the workflow so that all the data is directly available for all tools. The justification for this design principle comes from the following two characteristics of software development: extensive information exchange and each party's understanding of the other. Design Principle 3 addresses the requirements for creating and maintaining shared understanding.

Kernel theories: This principle is based on several theories. Communication is considered in theories of media richness [39], media synchronicity [40], and social presence [41]. These theories discuss the importance of understanding information as well as how information depends on the media format. The more uncertain and ambiguous the task is, the richer the media should be, including a degree of awareness. Team members communicate in order to have a shared understanding, and information is exchanged either synchronously or asynchronously; technology is often needed, for a variety of reasons. One of the problems reported in [42] is that developers have to rely on various tools and formats that do not necessarily follow any communication standards or that may not provide all the necessary cognitive support, and this can lead to misunderstandings. When combined with a complex infrastructure, this may adversely affect both the frequency and quality of communication and, ultimately, productivity [42].

System features: Creation and maintenance of shared understanding requires constant communication. Cognitive workflows must provide communication infrastructures that foster communication; they must also provide rich information, shared understanding, and awareness information.

Experiences from case company: Our study revealed a lack of communication between different teams and organizational units. This led to serious problems such

as a lack of feedback and an insufficient understanding of others' activities. On the other hand, informal means of communication were used extensively, which posed a hindrance to documentation.

Experiences with the prototype highlighted the need to integrate easy-to-use communication infrastructures into the workflow. Interviewees commented that different kinds of tools were necessary, ranging from instant messaging applications to video conferencing. The inclusion of communication infrastructures supported the collaborative nature of software development, eased the dissemination of knowledge, improved cooperative work, and provided several other benefits. In addition, rigorous documentation was encouraged.

Principle 4. Development of a workflow should address knowledge management

Data, information, and knowledge are all covered in this principle, as these concepts are closely interrelated. It is necessary to have a means of gathering, managing, and disseminating knowledge. For example, the knowledge obtained from various stakeholders may be used to form guidelines and principles, and this knowledge can be used by adopting knowledge bases while making the different development perspectives more visible. The number of various types of knowledge-management applications and solutions indicates the importance of this design principle. In addition, it is necessary to address the need for knowledge management in CWS. This design principle stems from the following characteristics of software development: knowledge-intensive work and processes, team knowledge, and knowledge management. Design Principle 4 addresses the requirements for enabling acquisition, sharing, and using knowledge.

Kernel theories: Several potential candidates for kernel theories are available. Here, we highlight the following: team knowledge and its management and coordination in geographically-distributed software development [43], the knowledge foundations of effective collaboration [4], and support for expert-level communication in developer-centered collaborative software development environments [21].

Nakokoji *et al.* [21] suggest that "software development is about information, generating information, and making information artifacts." To perform knowledge-intensive tasks and solve problems, developers must have an understanding of both the current state and the goal state, and naturally, they have to find a way to reach the goal. This understanding provides the basis for problem solving and task implementation. In addition, both team cognition and knowledge will support coordination and improve developer interactions, as developers can understand and anticipate what others do [43]. However, it is not a trivial task to have and understand all this information.

System features: Knowledge management aspects have to be covered, such as acquisition, sharing, and utilization, including the use of experience and skills.

Experiences from case company: Our studies revealed inadequate knowledge acquisition, sharing, and utilization in the case company. For example, information was either insufficient or it was not stored in a way that was accessible and useful for

others. Practical problems also included finding relevant data, which was often scattered over several sources without proper structure, classification, or means of identification. In some cases, the information was missing, while in others, there was too much information. Developers wanted to have instant access to all necessary information, tools, databases, and repositories directly from the workflow, with proper search and filter functionalities to allow the development to be based on up-to-date facts.

A workflow system prototype was built with the intention of making the developers produce accurate data in the correct format. The system helped the developers to think about what was really asked for in the tasks, and it guided them to produce the most relevant data. If there was a need for guidance and instructions, developers could request help, as the system included context-sensitive guidance. Developers could easily check the guidance, for example, for processes, activities, and tasks. The system enabled users to quickly understand how the process worked, how information flowed, and the purposes of each process element, activity, and task.

In their work, the developers created, stored, and shared information while implementing their tasks, and each developer accumulated information, knowledge, and experiences. These were recorded to ensure maintenance of the data even if the member were to leave the team. Different aspects of development were represented by groups of stakeholders. Behind each aspect, a platform of knowledge and technology was created to be adapted to the development process. These platforms were knowledge bases that covered both products and processes. Among other benefits, these data provided solution strategies to support the developers' work. They could be used to train new team members and to support team members who were unfamiliar with a particular task. As a result, developers seemed to understand what was needed to create the necessary content or work items.

Principle 5. Development of a workflow should address awareness and transparency

Awareness is critical for collaboration [42, 44, 45]. Awareness can be defined as an understanding of others' activities, which also provides the context for one's own activities [46, 47]. It is also suggested that awareness is the key to transparency [48]. Awareness and transparency enable the developers to be aware of the status of development activities and work items; further, they facilitate a common, shared understanding of the development goals and objectives. For example, transparency helps in achieving effective and open communication, while transparent tools and processes ensure the success of product development. The following characteristics of software development are also requirements of CWS: the need for understanding others' activities, information visualization, context awareness, situation awareness, and workspace awareness. Design Principle 5 addresses the requirements for enabling knowledge and supporting awareness and transparency.

Kernel theories: Selected kernel theories and justificatory works include discussing awareness and coordination in shared workspaces [46, 49] and developing and

maintaining awareness in collaborative software development [44]. Information visualization and situational awareness are needed in workflows [50].

System features: Transparency and awareness are key knowledge enablers. Technology is needed for creating and supporting transparency and awareness, and such technologies should be designed to support the development work and processes. Thus, we propose that cognitive workflows must provide transparency and awareness for data, tools, resources, and processes.

Experiences from case company: Various awareness issues with tools, documents, and work items were revealed in the case company. For example, tools, interfaces, and data usage caused an additional cognitive load owing to inadequate transparency or awareness information. Developers or managers were not always aware of the work conducted during the preceding or following phases, as it was not visible anywhere in practice; they were also not aware if something had been changed or work had progressed. This caused tracking issues, especially for managers, as there was no way to track the progress of the work toward subsequent process phases.

Proposals included providing awareness of others' actions and ensuring the transparency of the work to reduce overlapping activities and to control redundancy and the repetition of work. Further, the processes and phases were required to be visible so that the developers were aware of the context and dependencies between tasks and work items. Awareness was also needed to identify and understand the current conditions. Therefore, information was provided about the workspace, work items, and others' activities to help create a shared understanding and to tackle awareness issues. Workspace awareness, including, for example, information about developers' availability, instant messaging systems, and contact information, supported the instant connection between people and facilitated communication and collaboration. Task management and coordination ranging from backlogs to work allocation created the awareness of development tasks and objectives and provided information about the state of the work items and tasks.

Process visibility was addressed in the prototype by displaying an overview and the status of documents and other work items. The prototype also indicated the process and phase to which each of the different documents and work items was related. In addition, the prototype also enabled users to see what others were doing. The aim was to provide an understanding of all elements and their state in order to enhance one's awareness of a situation and, ultimately, to produce more accurate data and results. In summary, the action research intervention with the prototype proved that the overall transparency of the work was improved.

Principle 6. Design methods should result in a workflow that coordinates and structures the product development process

This design principle focuses on the ways to coordinate and structure the development activities and processes, which facilitate human interactions in collaborative development. The coordination, structuring, and synchronization of work are necessary in collaborative development, as different teams carry out their tasks

simultaneously or in sequence. In any case, the efforts must be integrated at some point, as there are often several dependencies and relations between the tasks and work items. Synchronization points can be used for this.

Coordination of activities is also supported by solid guidelines on what to do, that is, clearly stated requirements of the work to be done and acceptance criteria that explicitly show what is needed for each task. These guidelines also provide justification and purpose for the work. Synchronization points are natural sites for defining the information content to be produced in development tasks, and the results are checked against the guidelines and acceptance criteria at these points. Thus, we can identify the need to address the CWS requirements that are covered by design Principle 6, including creating structure, synchronizing, integrating contributions, and decision points.

Kernel theories: Hazelrigg [51] discusses decision-based engineering, where engineering design is basically recognized as a decision-making process. As decision-making requires clearly stated decision criteria, we can recognize the importance of synchronization points when they also provide the development direction (goals and objectives) and decision criteria (acceptance criteria). Cooper [52] has proposed decision points in his Stage-Gate® process. Gates, or decision points, can be used as "natural" synchronization points for development activities. Perrin and Godart [37] present an overview and the concept of synchronization points as points at which to share, manage, coordinate, and communicate in collaborative development. In summary, a decision-based development approach is proposed, where decision gates are also used as synchronization points. These points also coordinate and structure the development process.

System features: Workflows must structure and support the development process and integrate the contributions of each participant seamlessly into a coherent whole.

Experiences from case company: To obtain feedback on design Principle 6 from the case company, a workflow was built to provide a defined input for all activities, along with information and knowledge about tasks, resources, objectives, decision criteria, and the context. In the workflow, the development activities were handled as loosely coupled sets of tasks that aimed at producing a coherent piece of work focusing on defined topics. When defined content for an activity or a task was completed, the results of the activities and tasks were presented for decision makers as proposals that could be evaluated against the given decision criteria defined by relevant stakeholders. The completion order of tasks and activities was not strictly enforced, and developers could choose how to reach their goals within specifically defined and acceptable performance boundaries and existing constraints.

Before the workflow prototype was implemented, cooperation between different development phases was lacking, and collaboration was mostly observed between successive phases; there was no clear picture as to why other phases needed the input and work results. With the prototype, the justification and purpose for work was more visible to the developers.

The work results were also synchronized and/or integrated at the decision points. Decision points clearly defined what was necessary in order to produce the required content or work items. In addition, the results were examined at each decision point, and decisions were made as to whether the work could progress to another stage.

4. Evaluation and Specifying Learning

Design theory provides a prescriptive theory that informs how to do something [11, 53]. This type of theory proposes guidelines that are meant for constructing an artifact. It provides a prescriptive theory, with, for example, methods, techniques, and principles of form and function for constructing an artifact [53].

In the present study, the result of design theory is a list of design principles that, if followed in the design process, both provide a system that addresses design goals and produce a system that has the requested features implemented in it — a CWS.

Design theories rely on the knowledge of kernel theories for their design principles [53]. In this work, the knowledge was collected first from the literature in the form of kernel theories, and then the resulting design theory was evaluated in the industry in an action research study using a system prototype. During the action research, the system was implemented and evaluated in an iterative manner. In each cycle, new learning emerged, and this was used in the following cycle.

The design theory is proven through the validation and evaluation of design principles by constructing a system and using it in its intended settings in a case company. The empirical work shows clearly that the design principles and the proposed system features are recognized as necessary for successful CWS.

The evaluation of each design principle with learning from the action research intervention is discussed next. The individual design principles are all related to each other, even if they have different focuses. They are complementary and contribute partially to define the overall research problem.

The first design principle necessitates having cognitive support integrated into the workflow. Software engineering is not only about how to build a software system; in addition to that, and perhaps even more importantly, software engineering is about understanding what to build. This requires, for example, the use of existing knowledge and the creation of new knowledge is supported.

The prototype was built for practical evaluation. In the prototype, all development activities, inputs, and outputs were specified along with information and knowledge about tasks, resources, objectives, decision criteria, and the context. A graphical presentation and links to data, development and collaboration tools, and guidance were provided.

When the tasks were defined and described carefully with justification for all activities, developers easily could understand what was expected from them, and what was the intention in each task. The empirical evaluation of the prototype proved that by linking the developers and their tasks to development processes, information, knowledge, and tools, and by helping them understand what they

needed to do as well as why and how they needed to do it, the developers' cognitive performance was improved. Developers were able to create shared understanding and to work better toward the development goals. The developers also understood better the development activities and tasks as they were put into a context. The graphical presentation helped in navigating a workflow, tracking the work, and visualizing the dependencies of work items. The necessary cognitive support was found adequate to meet the first design principle.

The second design principle addresses the collaborative needs of workflow systems. It is not enough only to have a system that connects persons and transfers work items to the next handler; there must also be several other collaboration supporting features in the workflow. The workflow system must be flexible and allow independent work as well as teamwork; it must manage and coordinate the collaborative work.

In practice, developers must understand how their work is positioned in the overall picture, in order to understand the whole context of their work — each individual's work is part of the larger whole. For this, the graphical presentation of the process, activities, and tasks was developed, clearly showing the dependencies and relationships of tasks, activities and work items. The persons who worked with an item could also be identified in order to see the work item's history, enable the tracking of changes, etc. Moreover, it was clearly shown in the prototype where the outcomes of one's tasks were to be delivered, so that the whole path of the work items could be traced, back and forth. A set of collaboration tools was also integrated into the workflow. All of this together satisfied the second design principle.

Furthermore, we learned from the empirical evaluation that having a clearly articulated common goal is necessary; otherwise, different developers, teams, and development phases have different perspectives and varying understandings of a product and product development. In our prototype, the common goal was then further divided into smaller pieces of work, activities, and tasks that instantiated the higher level goals and purposes. Individuals and teams could then implement activities and tasks, and their contributions were integrated to form the desired product. The inclusion of collaboration tools such as change management tools and history, tracking, and project management tools were also found to be a very useful addition to support the collaboration. All of these created a good basis for successful collaboration and proved the importance of the second design principle.

The third design principle addresses the communication in workflows and is quite straightforward. Communication tools must be integrated into the workflow. The collaboration and the distributed development that are often met in software engineering means that a lot of communication is required, for example, to set shared goals or to exchange information. Communication channels between the experts support the collaborative nature of development, which eases the dissemination of knowledge and improves cooperative work.

The "process chart" linked each activity and actor with contact information, enabling developers to communicate with the correct persons. In addition, each task,

activity, and work item was assigned an owner, enabling others to see who was responsible for each. With contact information readily available, owners could be contacted via email, telephone, or a note left in the workflow system. This latter option enabled developers to contact owners by clicking on contact information, which presented the option of leaving a note or sending an email. In empirical evaluations, this use of technology to support communication was found to be helpful and necessary to collaboration. Further ideas also emerged for future development. In particular, instant messaging tools (like Skype) were requested to be included in the prototype; however, those requests were not yet met in the prototype. The research clearly indicated the validity of this design principle, and even without instant messaging tools, the prototype already proved to be successful in fulfilling the third design principle.

The most important thing learned from the action research intervention was the importance of fulfilling developers' information needs and defining the information flows to support communication and information distribution. In our prototype, communication supported coordination and synchronization by providing the whos, whats, hows, whys, and wheres. Supporting communication also provided cognitive support.

The fourth design principle is about knowledge management in workflows. Software development is intellectually demanding knowledge work, involving hard decisions, innovativeness, knowledge creation, and many other mental activities, all dealing with knowledge and information. Knowledge work and managing knowledge is also cognitive effort and thus requires cognitive support.

In practice, certain managers representing different stakeholders were nominated. Their task was to gather the experiences and knowledge and then to formulate guidelines and principles, for example. These guidelines were then followed in the product process. Such managers are where the knowledge of various stakeholders, representing different development viewpoints, is collected. The main role of managers with respect to each viewpoint is to use that knowledge and to form guidelines and principles. There were also managers who were "champions," advancing their viewpoints, documenting practices and know-how, and making them known. Their experiences with and knowledge of both the product and the process were documented and made available through links from the workflow system. The data included background information about the work and the work environment, examples of solutions to development problems, and links to relevant standards and legislation. Along with the communication tools, this data from champion managers improved knowledge management appreciably.

The action research showed that when knowledge and information were gathered, stored, and disseminated, then the developers were able to utilize that information to implement development tasks and solve problems they encountered in development. This way, the workflow provided shared knowledge about processes, products, tools, and team members, including the use of experience and skills. This knowledge is developed over time, and the utilization of shared knowledge is furthered during

development tasks. Evaluations showed that this kind of shared knowledge helped developers to understand different perspectives and created a common ground, thus demonstrating the importance of the fourth design principle. Finally, the utilization of the prototype provided common guidelines and instructions and a harmonized way to aim for the common goal, instead of leaving each group or department to implement its own plans.

The fifth design principle concerns awareness and transparency. Technologies are used for creating and supporting transparency and awareness, and it is important that those technologies are designed to support the development work and processes. On the other hand, processes and practices must also support awareness and transparency, as tools alone cannot provide that.

Transparency is the visibility of things, and awareness can be defined as an understanding of others' activities. These both help in communicating and help to understand what is going on. They both provide shared understanding and help to put things into the correct context. The transparency and awareness of the prototype were improved upon, for example, by providing information about others' activities and showing ongoing work, who was working with what, the status of the development process, and showing where individuals' actions affected. In addition, the work item histories and change log enabled activities to be tracked throughout the development cycle. This support for transparency and awareness was available from the prototype's graphical user interface.

As software development is inherently cooperative, it requires several developers who need to coordinate their efforts. Our empirical evaluation showed that an important part of software development is the creation and maintenance of a shared understanding that concerns the state of project tasks and artifacts and the activities and expertize of developers. The results show the importance of understanding each other and of transparency and awareness. Awareness was agreed to be critical to collaboration and to be a necessity for a better understanding of the information. Thus, this design principle was agreed to be necessary. The prototype also addressed the awareness and transparency needs of developers discussed in Sec. 3.2.

The sixth design principle requires the coordination and structuring of the development process. The workflow must provide proper information about the goals and objectives of organizations' processes. How and in what order the work is done does not have be strictly defined; instead, it is preferable that the developers have the ability to choose how to reach their goals within specifically defined acceptable performance boundaries and existing constraints. Coordination and structuring the processes by defining these boundaries and constraints and making them visible are the main goals of this principle.

In the prototype, work activities and items were defined based on the decision points. For each activity that was necessary to enter any decision point, clearly defined acceptance criteria covering completeness and quality were created. The decision criteria and decision-making were integrated into the development process, and the development goals were defined within decision points, with clearly specified

decision criteria for various abstraction levels of work, including processes, activities, and tasks. These criteria guided the decision-making activities and defined the information content that needed to be created during development tasks between decision points. In addition, the data flows were described. By these means, the prototype offered information about the developers' goals and described the dependencies between activities and work items. All of this also clarified why all the processes and activities were necessary in the development process, even if the developers previously did not understand the need for all the development activities. The coordination and integration of development results also occurred at the decision points. In decision points, the results of activities and tasks were presented for decision makers as proposals that were evaluated against given decision criteria defined by relevant stakeholders. This particularly helped managers, who could obtain accurate and up-to-date information about the state of the work and then could "steer" the work and provide guidance and other corrective actions whenever needed.

From the action research intervention, we learned that this design principle was seen as important, and having this kind of support in the workflow proved to be useful. In the prototype, various development phases were handled by describing the dependencies and relationships among various processes and the data flows among them. The developers easily understood the intended characteristics of the end product and why they had been requested. The activities in the development process and the expected work results were justified, and the criteria for achieving the purposes, values, and priorities were defined.

Evaluation of current workflow systems revealed that many traditional workflow systems could already be counted out, based on the literature research; current workflow models do not support cognition and cognitive work [9, 10, 22–25]. There are also other good approaches, such as unified modeling language (UML) or the "4 + 1 view model," which enable either analyzing and designing software products and decision-making or describing the interests of various stakeholders. Similarly, there are numerous computer-aided software engineering (CASE) tools to help designers and developers. However, they do not specifically address the cognitive aspect; thus, they do not meet the first design principle, which is definitional in the sense that all CWS must address it.

However, there are works that have discussed cognition in the context of workflows [cf. 9, 10, 50], and it is reasonable to see how our workflow model compares to those efforts. Table 3 (following page) evaluates workflow systems with respect to how they implement the proposed design principles. In the table, "+" means that the design principle is met, and "−" means that the design principle is not met.

Most of the current workflow models do not meet the first design principle. The workflow models by Zhuge [9], Wang and Wang [10] and Jalote-Parmar *et al.* [50] were identified from the literature as also addressing the first design principle, and thus they were included in the more thorough examination. The evaluation of other workflow systems than our own is based on the literature, as the implementation of

Table 3. Evaluation of workflow models in relation to proposed design principles.

Workflow models Design principles	Zhuge [9]	Wang and Wang [10]	Jalote-Parmar *et al.* [50]	Our work
1. Design methods must develop a workflow that provides cognitive support	+	+	+	+
2. Development of a workflow system should address the needs of collaborative work	+	−	−	+
3. Development of a workflow should address communication	+	−	−	+
4. Development of a workflow should address knowledge management	+	+	+	+
5. Development of a workflow should address awareness and transparency	−	+	+	+
6. Design methods should result in a workflow that coordinates and structures the product development process	−	+	−	+

those workflow systems in any form into the real-life processes of the case company would not have been possible. Our own system, however, was evaluated in practice in a real development environment.

The examination of existing workflow models showed that Zhuge's [9] proposition addresses collaboration to an extent. It recognizes the different abstraction levels of work, communications, and connections between persons and the coordination and management of collaborative work. Flexibility is also very well included in his work, as is knowledge management. Together these solve many of the issues related to design Principles 1–4. However, Zhuge's model lacks the awareness and transparency aspects and does not address the coordination and structuring of the development process. Nor does it propose the inclusion of other kinds of collaboration supporting tools.

Wang and Wang [10] also address several requirements related to the design principles. They include knowledge management and awareness support as integral mechanisms in their model. They also address process structuring and coordination quite well. However, their support for collaboration is limited, focusing mostly on context creation and slightly touching the management and coordination of resources. Thus, their approach does not offer much support for developers in their daily work, as it is more focused on the real-time routing and strategic control of process management.

The solution by Jalote–Parmar *et al.* [50] is a workflow-centered framework for an expert decision-making system; they integrate the knowledge of cognitive processes into system development. Their focus is on providing situational awareness for decision-making through information visualization. There are strengths and weaknesses in their proposal; for example, it takes into account awareness, information visualization, knowledge management, and the context of work. However, it does not address collaboration and communication, probably due to the nature of the domain. The main issue is that their work is strongly in the clinical domain, and it cannot be straightforwardly transferred to the software domain. Thus, the application of their

results to software development workflows would require further industrial evaluations in the software domain. Still, their proposal for designing a workflow seems to be solid, and it should also be taken into serious consideration in software engineering where applicable.

A CWS designed by following the design principles addresses all of these principles, as discussed earlier in this section. The experiences in the case company proved that the designed systems addressed all the design principles.

To specify the learning, this work complements the traditional workflow models [cf. 22–24, 54] by providing cognitive support. Workflows are one way to manage processes, transfer the work and data from one to another, and help establish a logical order for task implementation. There are several works that discuss the workflows and their modeling and technical solutions; however, not many authors have applied the cognitive viewpoint to workflows.

As the study shows, this cognitive support is not adequately provided within current workflow models. We argue that proper support for cognitive work in workflows will help developers use available knowledge to come up with creative solutions to non-routine situations and to improve the efficiency and the results of the product creation process. Cognitive support also helps to address changes and unexpected events; enough information is provided so that developers can make solid decisions based on facts, and they are supported by earlier experiences and other guides that they might apply while working.

The empirical evaluations showed how cognitive support integrated into the workflow was useful, and it improved developers' abilities to accomplish their work. With the prototype, the developers were also able to respond to changes and solve practical development problems more efficiently. The quality of the work results was improved as well.

5. Discussion and Conclusions

The aim of the proposed design theory for CWS is to provide a set of requirements and design principles for creating an effective workflow system that supports cognitive work in software development. Table 4 summarizes the elements of our design theory, which was validated in a real-world environment with an action research intervention. Each of the features, kernel theories, and design principles has been extensively discussed in the preceding section.

The proposals complement each other, and each contributes partially to defining CWS. Together, they tackle the cognitive challenges discussed in this paper. The action research results verified the usefulness of a common, flexible workflow that connects the development phases, so jointly processed data is available to all relevant stakeholders, and that data can be completed and made more accurate as the development progresses.

The results of this research will support developers' cognitive work, which is a result of interactions among individuals, artifacts, and resources in their

Table 4. Design theory for CWS.

Design goals	CWS design methods must develop a workflow that provides cognitive support, i.e. design methods must provide the means to address the needs of complex knowledge work that requires cognitive support.
1. A set of system features	1.1. Cognitive workflows must create a common, flexible development environment that reduces developers' cognitive load and supports cognitive activities and processes. 1.2. Cognitive workflows must foster cooperation and distributed work. 1.3. Cognitive workflows must provide communication infrastructures that foster communication as well as rich information, shared understanding, and awareness information. 1.4. Information and knowledge must be distributed among the development teams. 1.5. Cognitive workflows must provide transparency and awareness for data, tools, resources, and processes. 1.6. Workflows must structure and support the development process and integrate the contributions of each participant seamlessly into a coherent whole.
2. Kernel theories	2.1 Theory of distributed cognition, workflow-based cognitive flow management for distributed team cooperation, and cognitive approaches to business process management. 2.2 Approaches for alleviating distance and for cooperative work and cognition. 2.3 Theories of media richness, media synchronicity, and social presence. 2.4 Knowledge foundations of effective collaboration, team knowledge, knowledge management, and coordination in geographically distributed software development, and support for expert-level communication in developer-centered collaborative software development environments. 2.5 Awareness and coordination in shared workspaces. 2.6 Planning the design process and quality of execution of key activities.
3. Design principles	3.1 Design methods must develop a workflow that provides cognitive support. 3.2 Development of a workflow system should address the needs of collaborative work. 3.3 Development of a workflow should address communication. 3.4 Development of a workflow should address knowledge management. 3.5 Development of a workflow should address awareness and transparency. 3.6 Design methods should result in a workflow that coordinates and structures the product development process.

environment, as suggested in [36]. The aim of the proposed design principles is to reduce the cognitive burden — the burden of keeping unnecessary factors in mind, such as how to use tools, how to link information between tools, and how to search for relevant data — as well as to enable and foster communication, transfer knowledge, provide awareness, provide context for work and knowledge, and support collaborative work.

5.1. *Implications and future research areas*

The results presented in this article constitute a concept that is, to the best of our knowledge, not discussed in existing literature. This work provides valuable insights

for academic research and lays the foundation for further scholarly inquiry, including a validation of the findings in other companies and domains besides information and communication technology, and as no testable hypotheses are presented, there is an opportunity for further development of this design theory. Furthermore, this research serves as the basis for developing the actual implementation of cognitive workflows. One of the possible future research topics would also be the quantitative analysis to support the validation of design theory more.

For practitioners, the design theory for CWS provides new ideas to improve the efficiency of work and the accuracy of results. In summary, cognitive workflows designed by following the proposed design principles would provide a better understanding of the context of work, especially the real needs of all processes, phases, and functions; this in turn would provide better development results, primarily because the produced data and other results will fulfill their purpose more effectively and ensure less wastage.

5.2. *Conclusions*

Current workflow models do not address the needs of collaborative software development. In particular, cognitive support is lacking in current workflows. Existing methods for designing CWS do not meet the six design principles discovered through design theory and presented in this paper. As a result, although current workflow systems address different cognitive aspects, they lack the comprehensiveness needed for such systems.

Our design theory for CWS leads to six design principles that help to resolve the shortcomings of current workflow systems and their design methods. The proposed design theory was validated in practical settings with an action research intervention. The action research demonstrated the relevance, feasibility, and usefulness of the proposed design theory in a real-world environment. Thus, the proposed design principles provide a promising solution to current issues in cognitive workflows.

Acknowledgments

This research is supported by the European ITEA2 program with national funding from Tekes (the Finnish Funding Agency for Technology and Innovation). The authors would like to thank AMALTHEA, ONION, and DHR project partners for their assistance and cooperation.

References

1. F. O. Bjørnson and T. Dingsøyr, Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used, *Inf. Softw. Technol.* **50**(11) (2008) 1055–1068.
2. D. Panagiotou and G. Mentzas, Leveraging software reuse with knowledge management in software development, *Int. J. Softw. Eng. Knowl. Eng.* **21**(5) (2011) 693–723.

3. J. J. Sung, Representation-oriented software development: A cognitive approach to software engineering, in *Proc. 17th Annual Psychology of Programming Interest Group Workshop* (*PPIG'05*), 2005, pp. 173–187.
4. D. Noble, Knowledge foundations of effective collaboration, in *Proc. of 9th Int. Command and Control Research and Technology Symp.*, 2004.
5. P. Helo, Managing agility and productivity in the electronics industry, *Ind. Manag. Data Syst.* **104**(7) (2004) 567–577.
6. R. G. Cooper, S. J. Edgett and E. J. Kleinschmidt, Benchmarking best NPD practices-III, *Res. Technol. Manage.* **47**(6) (2004) 43–55.
7. A. Gupta, K. S. Pawara and P. Smart, New product development in the pharmaceutical and telecommunication industries: A comparative study, *Int. J. Prod. Econ.* **106**(1) (2007) 41–60.
8. S. Buckingham Shum, Negotiating the construction of organizational memories, *J. Univ. Comput. Sci.* **3**(8) (1998) 899–928.
9. H. Zhuge, Workflow- and agent-based cognitive flow management for distributed team cooperation, *Inf. Manage.* **40** (2003) 419–429.
10. M. Wang and H. Wang, From process logic to business logic: A cognitive approach to business process management, *Inf. Manage.* **43**(2) (2006) 179–193.
11. J. G. Walls, G. R. Widmeyer and O. A. El Sawy, Building an information system design theory for vigilant EIS, *Inf. Syst. Res.* **3**(1) (1992) 36–59.
12. J. G. Walls, G. R. Widmeyer and O. A. El Sawy, Assessing information system design theory in perspective: How useful was our 1992 initial rendition? *J. Inf. Technol. Theory Appl. (JITTA)* **6**(2) (2004) 43–58.
13. J. Hyysalo, J. Lehto, S. Aaramaa and M. Kelanti, Supporting cognitive work in software development workflows, in *Proc. of Profes 2013, 14th Int. Conf. Product-Focused Software Process Improvement*, 2013, pp. 20–34.
14. J. Birkinshaw, G. Hamel and M. J. Mol, Management innovation, *Acad. Manage. Rev.* **33**(4) (2008) 825–845.
15. W. Tsai, Social structure of "coopetition" within a multiunit organization: Coordination, competition, and intraorganizational knowledge sharing, *Organ. Sci.* **13**(2) (2002) 179–190.
16. A. C. Inkpen, Creating knowledge through collaboration, *Calif. Manage. Rev.* **39**(1) (1996) 123–140.
17. M. L. Markus, A. Majchrzak and L. Gasser, A design theory for systems that support emergent knowledge processes, *MIS Q.* **26**(3) (2002) 179–212.
18. D. Draheim, V. Geist and C. Natschlaeger, Integrated framework for seamless modeling of business and technical aspects in process-oriented enterprise applications, *Int. J. Softw. Eng. Knowl. Eng.* **22**(5) (2012) 645–674.
19. H. van Leijen and W. R. J. Baets, A cognitive framework for reengineering knowledge-intensive processes, in *36th Annual Hawaii Int. Conf. System Sciences*, 2003, pp. 97–106.
20. J. J. G. van Merriënboer, *Training Complex Cognitive Skills* (Educational Technology Publications, Englewood Cliffs, NJ, 1997).
21. K. Nakakoji, Y. Ye and Y. Yamamoto, Supporting expertise communication in developer-centered collaborative software development environments, in *Collaborative Software Engineering* (Springer, Berlin Heidelberg, 2010), pp. 219–236.
22. M. Klein and C. Dellarocas, A knowledge-based approach to handling exceptions in workflow systems, *Comput. Support. Coop. Work* **9** (2000) 399–412.
23. N. R. Jennings, P. Faratin, M. J. Johnson, T. J. Norman, P. O'Brien and M. E. Wiegand, Agent-based business process management, *Int. J. Coop. Inf. Syst.* **5**(2–3) (1996) 105–130.

24. W. M. P. van der Aalst and T. Basten, Inheritance of workflows: An approach to tackling problems related to change, *Theor. Comput. Sci.* **270**(1–2) (2002) 125–203.
25. M. Minor, R. Bergmann, S. Gorg and K. Walter, Reasoning on business processes to support change reuse, in *Proc. of the 13th Conf. Commerce and Enterprise Computing*, IEEE, 2011, pp. 18–25.
26. G. M. Kasper, A theory of decision support system design for user calibration, *Inf. Syst. Res.* **7**(2) (1996) 215–232.
27. M. Siponen, R. Baskerville and J. Heikka, A design theory for secure information systems design methods, *J. Assoc. Inf. Syst.* **7**(1) (2006) 725–770.
28. O. Hanseth and K. Lyytinen, Design theory for dynamic complexity in information infrastructures: The case of building Internet, *J. Inf. Technol.* **25**(1) (2010) 1–19.
29. E. W. Stein and V. Zwass, Actualizing organizational memory with information systems, *Inf. Syst. Res.* **6**(2) (1995) 85–117.
30. S. Gregor and D. Jones, The anatomy of a design theory, *J. Assoc. Inf. Syst.* **8**(5) (2007) 312–335.
31. R. Baskerville and J. Pries-Heje, Explanatory design theory, *Bus. Inf. Syst. Eng.* **2**(5) (2010) 271–282.
32. O. Hanseth and K. Lyytinen, Theorizing about the design of information infrastructures: Design kernel theories and principles, *Sprouts: Working Papers on Information Systems* **4**(12) (2004) 4–12.
33. J. G. Walls, G. R. Widermeyer and O. A. El Sawy, Assessing information system design theory in perspective: How useful was our 1992 initial rendition? *J. Inf. Technol. Theory Appl.* **6**(2) (2004) 43–58.
34. R. Baskerville and A. T. Wood-Harper, A critical perspective on action research as a method for information systems research, *J. Inf. Technol.* **11**(3) (1996) 235–246.
35. G. Susman and R. Evered, An assessment of the scientific merits of action research, *Adm. Sci. Q.* **23**(4) (1978) 582–603.
36. J. Hollan, E. Hutchins and D. Kirsch, Distributed cognition: Toward a new foundation for human-computer interaction research, *ACM Trans. Comput.-Human Interact.* **7**(2) (2000) 174–196.
37. O. Perrin and C. Godart, A model to support collaborative work in virtual enterprises, *Data Knowl. Eng.* **50**(1) (2004) 63–86.
38. J. D. Herbsleb and D. Moitra, Global software development, *IEEE Softw.* **18**(2) (2001) 16–20.
39. R. L. DaftR. H. Lengel, Organizational information requirements, media richness, and structural design, *Manage. Sci.* **32**(5) (1986) 554–571.
40. A. R. Dennis and J. S. Valacich, Rethinking media richness: Towards a theory of media synchronicity, in *Proc. of the 32nd Annual Hawaii Int. Conf. System Sciences*, IEEE 1999, pp. 1–10.
41. J. Short, E. Williams and B. Christie, Communication modes and task performance, *Readings in Groupware and Computer Supported Cooperative Work* (Morgan-Kaufmann Publishers, 1976), pp. 169–176.
42. M. Jiménez, M. Piattini and A. Vizcaíno, Challenges and improvements in distributed software development: A systematic review, *Adv. Softw. Eng.* **2009**(3) (2009) 1–14.
43. J. A. Espinosa, S. A. Slaughter, R. E. Kraut and J. D. Herbsleb, Team knowledge and coordination in geographically distributed software development, *J. Manage. Inf. Syst.* **24**(1) (2007) 135–169.
44. I. Omoronyia, J. Ferguson, M. Roper and M. Wood, A review of awareness in distributed collaborative software engineering, *Softw.-Pract. Exp.* **40**(12) (2010) 1107–1133.

45. A. Sarma, A survey of collaborative tools in software development, UCI ISR Technical Report, UCI-ISR-05-3 (Institute for Software Research, 2005), https://isr.uci.edu/tech_reports/UCI-ISR-05-3.pdf.
46. P. Dourish and V. Bellotti, Awareness and coordination in shared workspaces, in *Proc. ACM Conf. Computer Supported Cooperative Work*, 1992, pp. 107–114.
47. T. Robertson, Cooperative work and lived cognition: A taxonomy of embodied interaction, in *Fifth European Conf. Computer-Supported Cooperative Work* (*ECSCW '97*), 1997, pp. 205–220.
48. M. Beaudouin-Lafon and A. Karsenty, Transparency and awareness in a real-time groupware system, in *Proc. ACM Symp. on User Interface Software and Technology,* (*ACM 05*), 1992, pp. 171–181.
49. C. Gutwin, S. Greenberg and M. Roseman, Workspace awareness in real-time distributed groupware: Framework, widgets, and evaluation, in *Proc. Int. Conf. on Human–Computer Interaction: People and Computers*, 1996, pp. 281–298.
50. A. Jalote-Parmar, P. Badke-Schaub, W. Ali and E. Samset, Cognitive processes as integrative component for developing expert decision-making systems: A workflow centered framework, *J. Biomed. Inform.* **43**(1) (2010) 60–74.
51. G. A. Hazelrigg, A framework for decision-based engineering design, *J. Mech. Des.* **120**(4) (1998) 653–658.
52. R. G. Cooper, Stage-gate systems: A new tool for managing new products, *Bus. Horiz.* **33**(3) (1990) 44–54.
53. S. Gregor, The nature of theory in information systems, *MIS Q.* **30**(3) (2006) 611–642.
54. M. Adams, A. ter Hofstede, D. Edmond and W. van der Aalst, Worklets: A service-oriented implementation of dynamic flexibility in workflows, in *Proc. CoopIS'06*, 2006, pp. 291–308.