Author(s): Käkölä, Timo; Koivulahti-Ojala, Mervi; Liimatainen, Jani

Title: An Information Systems Design Theory for Integrated Requirements and Release Management Systems

Year: 2009

Version:

Please cite the original version:

Käkölä, T.; Koivulahti-Ojala, M.; Liimatainen, J., "An Information Systems Design Theory for Integrated Requirements and Release Management Systems," System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on , vol., no., pp.1-10, 5-8 Jan. 2009. doi: 10.1109/HICSS.2009.66

An Information Systems Design Theory for Integrated Requirements and Release Management Systems

Timo Käkölä & Mervi Koivulahti-Ojala
University of Jyväskylä
40014 Jyväskylä
Finland
{timokk, meelheko}@jyu.fi

Jani Liimatainen
Accenture
00101 Helsinki
Finland
jani.liimatainen@accenture.com

Abstract: High-tech companies need to collect and analyze requirements and allocate them to appropriate product releases in market-driven product development. Development activities are typically scattered across multiple sites and involve multiple partners in different countries, complicating requirements and release management. Flexible, scalable, and secure groupware-based support for the activities provides substantial payoffs. Yet, the extant literature provides little theoretical guidance for designing and using requirements and release management systems in multi-site, multi-partner environments. This article develops the meta-requirements and a meta-design of an Information Systems Design Theory for the class of Requirements and Release Management Systems based on a case study in a global company and a literature review. The theory is scalable to meet the needs of global companies but simple enough so small and medium sized companies can also leverage it to implement requirements and release management solutions.

Keywords: Global software development, Information systems design theory, Knowledge management, Release management, Requirements management, Software process improvement

1. Introduction

To succeed in the global markets of software-intensive products, high-tech companies need to shorten the cycle time of new product development (NPD) while improving product quality and service delivery and maintaining or reducing the total resources required [7;40].

This concern can be dealt with (1) internally through strategies such as global software development, where development resources are distributed globally to reap cost benefits and address specific needs of geographically-defined markets [10;44], and software product line engineering and management, that is, the strategic acquisition, creation, and reuse of software assets [32; 36; 43] or (2) externally by acquiring commercial off-the-shelf components and outsourcing software development, maintenance, and related services to best-in-class international outsourcing service providers (IOSPs) [11, 31]. A viable third strategy is to enact both strategies in parallel.

All the strategies require companies to effectively collect, analyze, and utilize requirements in their market-driven product development [1;22;23;26;45;48]. This is particularly true during the earliest phases of NPD in which different stakeholders need to integrate their knowledge into product concepts that direct the internal personnel and the IOSPs during the downstream phases of NPD [2;8;21]. A well-defined product concept is necessary to establish a viable product line architecture that can be shared across the products within the product line to enable strategic reuse. Well-defined requirements, architectural interfaces, and product structures are prerequisites for assigning appropriately scoped projects to IOSPs for implementation [11; 31].

The achievement of such integration is complicated by several factors [13]. Large amount of requirements ranging from abstract wishes to detailed technical solution proposals are created continuously. Development activities are scattered across many sites and partners in different countries, limiting possibilities for setting up face-to-face meetings [26]. Organizational changes, differences in organizational cultures, and divergent perceptions about the prospective product's mission may make it difficult to reach an agreement about the product definition [12].

A commonly enacted software product line governance model and a strategic product line roadmapping process need to be instituted to ensure the organization is ready for multi-site development [36;52]. All sites should use as compatible processes, methodologies, tools, and terminology as possible to enact the governance model [24]. The roadmaps detail the planned evolutions of the product lines and products by explicating common and variable features and allocating the features to scheduled product releases and responsible development organizations [43].

A critical component of the governance model is that all requirements are (1) captured in a repository to ensure they are neither missed nor overlooked and (2) subjected to effective filtering in order to prevent information overload [34;48;49]. The remaining requirements are then refined, specified, estimated in terms of cost and resource implications, prioritized, and allocated to product releases and development units. Flexible, scalable, and secure communication, coordination, and collaboration support

978-0-7695-3450-3/09 $25.00 © 2009 IEEE

for these activities poses a significant challenge with substantial payoffs.

Design theories, unlike other theories, support the achievement of goals [4;25;37;50;51]. Walls, Widmeyer, and El Sawy [50, p. 37] argue that the information systems (IS) "field has now matured to the point where there is a need for theory development based on paradigms endogenous to the area itself" and call for information system design theories (ISDT) to fulfill that need. An ISDT is "a prescriptive theory based on theoretical underpinnings which says how a design process can be carried out in a way which is both effective and feasible" (ibid, p. 37).

Salo and Käkölä [48] found that groupware-based requirements management systems (RMS) need to be designed and used to redesign and enact the earliest phases of product development effectively in multi-site, cross-functional organizations. They developed an ISDT for RMS in order (1) to facilitate endogenous theory development in the context of RMS research, (2) to help RMS designers build successful RM systems, and (3) to guide organizations in evaluating and deploying RMS.

Salo and Käkölä [48] found that the benefits afforded by RMS were hampered if the RMS instances prescribed by the ISDT were not integrated with systems used in the downstream phases in order to provide transparent end-to-end support throughout the product development lifecycle. For example, customer representatives responsible for entering requirements could not use RMS to follow-up if and when the requirements would be implemented, lowering their interests in entering the requirements. The scope of the ISDT should thus be broadened to design systems that support the lifecycle more comprehensively.

This research focuses on integrating requirements management with release management. Release management is concerned with the identification, packaging, and delivery of product's elements [27]. It mirrors requirements management in the other end of the NPD lifecycle ensuring that internal and external releases meet the (specified and managed subset of) requirements identified in the front end of NPD and agreed upon during release planning and product line roadmapping. Based on our extensive industrial experience and review of academic literature, we hypothesize that the theoretical validity and practical relevance of the ISDT for RMS can be enhanced most effectively by extending the ISDT to provide integrated support for requirements and release management.

Release planning must be conducted carefully and systematically and the release plans must be communicated clearly and in time upstream to the stakeholders responsible for the requirements and the product strategy and downstream to the internal and external service providers [34]. Otherwise, it is difficult for the providers to schedule and synchronize their production activities to meet the requirements specified in the release plans. For example, requirements are unlikely to be measurable and the functional sizes of software releases cannot be estimated [17;28] if the requirements are not linked to releases implementing them. The extant literature provides little guidance for designing and deploying integrated Requirements and Release Management systems.

This article develops the product aspects of the ISDT for the class of Requirements and Release Management Systems (RRMS). It addresses the following research question: What are the necessary and sufficient properties of RRMS in a multi-site, multi-partner environment? The main contributions of the article are the meta-requirements of the ISDT and a meta-design that partially meets the meta-requirements. They are crystallized and validated based on (1) a case study in a global organization that deploys a RRMS instance (hereafter, RRMS) for effective requirements and release management and (2) a literature review in the areas of requirements management, release management, and process integration. The ISDT has been created based on the experiences in the global organization to ensure the meta-design is scalable. However, we have made every effort to simplify the meta-design so even small and medium sized organizations can leverage it to implement RRMS solutions.

2. Research Method

A literature review was performed to develop preliminary meta-requirements and meta-design elements before the case study started. The review was essential to reduce bias induced by the single case study and ensure generalizability of the meta-requirements and the meta-design to maximum possible extent [54]. Potential meta-design elements that according to the review were peculiar to the organization were eliminated.

RRMS was a proprietary Lotus Domino based application developed in the organization. It had been productized, that is, one RRMS design was used. Business units had been closely involved in designing the system from the beginning and become committed to using and further developing it. They considered RRMS highly malleable to changing business needs partly because the organization controlled RRMS and business units did not need to negotiate with external vendors when changes were needed. RRMS had been institutionalized across the organization by the time the study was started. It was used by all development projects and updated by more than 10 000 people. RRMS enabled the projects quickly to locate existing reusable assets, substantially increasing productivity. Details concerning the organization and RRMS are beyond the scope of the paper.

While there were many reasons why RRMS was successful in the organization, one reason is crucial from the viewpoint of creating the ISDT for RRMS: RRMS enabled the real-time transparency and management of NPD efforts organization-wide but it was well scoped, did not impose more control and order than was necessary, and enabled people to use other information systems they we-

re familiar with. For example, RRMS did not replace existing project and portfolio management and product line roadmapping systems, which require advanced algorithmic models (e.g., what if-analyses, cost and effort estimation, optimization of inter-dependent releases) and visualization techniques. Software development processes and systems were not significantly affected either because software development was beyond the scope of RRMS. Projects increasingly leveraged agile development practices. RRMS thus could not impose unnecessarily stringent control mechanisms on them. Only the inputs to and the deliverables of management and software development processes and systems were dealt with by RRMS. For example, if requirements in RRMS needed specific product or organizational models to make them understandable, the models in the modeling environments were linked directly to RRMS or, sometimes, imported to RRMS. The analysis of RRMS has helped us to scope the partial ISDT for RRMS appropriately.

Two authors, a doctoral student and a master's student in information systems research, worked full time in the organization during a 6 month study period. RRMS had become increasingly complex over the years when its designers had tried to meet the ongoing influx of new requirements. While its functionality had been documented well, the organization was keen to further develop it. A current state analysis of RRMS was thus deemed necessary to better understand the limitations and possibilities. Major RRMS design changes were not realized during the study. Authors had access to all relevant information and could interact with all people who had been involved with the RRMS design. They observed the use of RRMS, analyzed documentation, and conducted six semi-structured interviews with middle-level managers who had been involved with both process improvement and the design and use of RRMS. After interviews were completed, interviewees were provided with interview transcripts and summaries. Interviewees reviewed the meta-requirements and proposed new ones that were added to the original set if all interviewees considered the proposed meta-requirements critical for RRMS. The proposed meta-requirements and meta-design were used in the organization for further development of the RRMS.

Interestingly, the people authors interviewed or otherwise interacted with could not specify academic papers influential during the RRMS design. They were experts with long organizational tenures and relied on theories-in-use [3] developed primarily through social interactions (c.f., [42]), experiences from earlier projects, and agile development practices instead of academic papers or design theories. Authors thus became increasingly intrigued with how to build such a simple but scalable and effective ISDT for RRMS based on the case study that designers and managers in other organizations would be willing to use such a theory in addition to trial and error mechanisms and long-reinforced theories-in-use.

3. Meta-Requirements of the ISDT for RRMS

This section presents the meta-requirements for the ISDT for RRMS. They are introduced by revising a framework of Salo and Käkölä [48]. The framework considered meta-requirements in relation to three categories of services that RMS have to offer: (1) communication, (2) control and (3) change. Communication refers to the ability of RMS to disseminate requirements information within organization, including information about the rationale for RM and its relationships to external environment. Support for control ensures that requirements are dealt with in accordance with approved principles and procedures. Support for change is needed because products, technologies and customers change and RMS must remain amenable to adjustments at all levels of RM activity.

Table 1. A framework for categorizing the meta-requirements of the ISDT for RRMS.

Communication	Control	Change	Platform development	Process Integration
▪ Prioritization and valuation of requirements and the allocation of requirements into releases ▪ Traceability ▪ Single capture of information	▪ Content ownership and accountability ▪ Management and coordination ▪ Creating and sharing of metrics information ▪ Access rights and information security	▪ Version management of requirement documents ▪ Release planning ▪ Change management and impact analysis ▪ Defining and maintaining the requirements baseline	▪ Creation and reuse of reusable assets ▪ Knowledge creating capacity	▪ Process transparency ▪ Providing information at the right level of detail ▪ Providing high quality information

The three categories are valid for RRMS but two new ones are needed: (4) Platform-based product development and (5) Process integration. Platform is the physical implementation of the baseline entity that contains the common business requirements for all the derivative products the platform has been designed to support (c.f., [6;38;39; 40;43]). All customer specific product development occurs on top of the platform. Table 1 classifies all RRMS meta-requirements.

3.1 Meta-Requirements in Support of Communication

Prioritization and valuation of requirements and the allocation of requirements into releases

Requirements must be allocated into releases using requirement prioritization and valuation methods that enable the most crucial requirements to be implemented and released first [9; 47]. The methods are typically based on trade-off analysis between the (economic) values and implementation costs and resource constraints associated with the requirements [9,p.140;19;52]. They need to take

into consideration that all stakeholders do not have the same relative importance and that each stakeholder may valuate each requirement very differently [20;47].

According to interviews, customer involvement in valuation, prioritization and selection adds value to these processes: "*Requirements can be prioritized to releases by communicating with customers and agreeing on what functionalities are wanted and on what timetable.*"

Traceability

The purpose of requirement management is to achieve complete traceability from customers via the organizational departments to delivery [22,p.69]. Traceability improves risk assessment, project scheduling and change control [23,p.12]. All interviewees agreed that traceability from requirements elicitation to product delivery is critical for RRMS success. Interviewees also suggested other needs for traceability (e.g., linking components, errors, and use cases). But it is expensive to collect and manage traceability information [30,p.129]. The following traceability meta-requirements should always be implemented:

- Ability to trace forward from requirement sources to requirements and from requirements to subsequent features and corresponding design elements and designs [15]
- Ability to forward trace from design elements to subsequent designs and backward from corresponding features to requirements [15]
- Ability to trace from requirements directly to design entities and backward [30]
- Ability to trace requirements dependencies [18;30]

Single capture of information

RRMS instances should be the single capture points for requirements, ensuring that there is no redundant and inconsistent requirement information in the organization and that all requirements are handled appropriately in a single effective and transparent process reducing the risks of missing or forgetting requirements. RRMS should thus be (1) easy to use for occasional users in order to ensure that they enter the information and (2) interfaced to partners' systems to ensure that partners can create, update, and review information as necessary.

3.2 Meta-Requirements in Support of Control

Content ownership

Requirements and release management activities should have appropriate owners. The main obstacles to successful requirements management are the lack of norms for project and requirements management [22,p.70]. Content ownership can be enhanced by assigning roles with clearly defined responsibilities to persons. For example, a set of requirements could be allocated to a requirement manager while particular releases could be assigned to a release manager. The role definitions and assignments improve accountability, enable evaluations of people with respect to their role expectations (e.g., release managers may be evaluated based on the quality of releases they are responsible for), and can ensure that all agreed upon deliverables are delivered in time and meet the defined quality criteria. Role based management also facilitates organizational (e.g., people may continue their work in their old roles in a new organization) and individual level changes (e.g., a new person takes responsibility of a role) [33; 34]. This meta-requirement can thus be used in personnel evaluation, process development, and quality control.

Management and coordination

Flows of requirements are coordinated and requirements are allocated to releases through managerial activities and decisions. For example, decisions need to be taken concerning the acceptance of particular requirements to the development process and the maturity levels of the requirements. To allocate requirements to specific releases and implementation teams and track their progress, requirements are assigned different statuses. Interviewees commented this meta-requirement: "*It should be possible to see from the tool who is responsible for certain parts of the process, who makes decisions concerning those parts, what the timetables are, and what kind of documentation should be available.*"

Creating and sharing of metrics information

Measurement is an essential part of process management [14;29;33]). Defined and measurable objectives are needed to evaluate the current status and develop the process. Metrics information enables comparison of process effectiveness across projects and products over time (e.g., project portfolio and product line management).

Access rights and information security

Access rights and information security policies are important in high-technology companies. Products are typically designed and implemented in complex networks of companies where competitors are also involved. RRMS must help ensure that partners do not access each other's sensitive information. Multiple partners can build competing products on top of a shared platform and thus need to share information about the platform but they should know nothing about each other's products and objectives.

3.3 Meta-Requirements in Support of Change

Version management of requirement and release documents

Versions of individual requirements and requirements specifications need to be controlled [53, p.268]. Change management and document version management processes must be in place to create and maintain requirement baselines. Requirement document version management is related to the general workflow management. However, one interviewee emphasized that it is most useful in requirement development phase, not in later phases when the documents are relatively stable. Change management and version management processes should be aligned and agreed upon: "*If the change is large, a new requirement can*

be created or the old one can be changed via the change management process. In practice, we could use version management for small revisions. But if the changes are too large, change management needs to be used."

Release re-planning

When software development is market-driven [45], release planning and requirement prioritization are parts of strategic product planning [9]. Release re-planning is needed when, for example, the product strategy is changed. Release re-planning may be related to planning the release cycles and the scope, role and timing of each individual release project or to release project management (i.e., the length, scope, and the number of iteration cycles of a project) when individual release projects develop the product versions [46].

Change management and impact analysis

A clearly defined change management process is needed to estimate impacts of possible changes and to control the changes made to the requirements and releases [34]. Change management and impact analysis enable organizations to be aware of the influences of requirement changes on the resources and market situation.

Defining and maintaining the requirements baseline

Baselining refers to the freezing of current agreed upon requirements. When the baseline decision has been made, subsequent requirements are treated as change requests and compared to the baseline. If the requests are accepted, they will enter product development through the change management process [34].

3.4 Meta-Requirements for Platform Development

Creation and reuse of reusable assets

Platforms are strategic organizational assets designed to be reusable and afford common features and predefined variation mechanisms through which mass-customized products can be created quickly and cost effectively [43]. Platforms consist of assets such as requirements, design elements, components, and user interfaces that are often also reusable. RRMS should provide a comprehensive information model to describe the assets adequately and advanced search functionalities to help projects easily find assets suitable for their needs.

Knowledge creating capacity

Knowledge creating capacity [16] of RRMS refers to a functionality which helps users to *leverage information and the platform assets in novel and possibly unforeseen ways*. The following quote clarifies the matter:

"*Requirement management involves the identification and management of baselines and products [for strategic reuse]. For example, we could see from the tool that this baseline is good for our purposes and we just have to change it from there and there in order to have a right configuration for our needs.*"

Ideally, users should not have to manually check the suitability of baselines or products for strategic reuse. RRMS should suggest alternative baselines and automatically rank them by taking the parameters and constraints imposed by the context of reuse into account. This is possible when a mature platform has been created.

3.5 Meta-Requirements in Support of Process Integration

Process transparency

RRMS should help make integrated requirements and release management processes transparent, that is, the stakeholders involved should be able to understand the results of their RRMS-mediated actions and deal with unexpected errors or coordination breakdowns quickly before expensive disruptions in routines and flaws in deliverables occur [33;34]. Transparency can be facilitated by standardizing the terms and forms of information transfer. One interviewee stated: "*Process transparency is especially important in decision making situations. Another important situation is when someone cannot implement, for example, a requirement in a given timetable.*"

High quality information

Organizations have to be able to base their requirements and release decisions on high quality (i.e., accurate, specific, relevant, reliable, timely, and accessible) information [41]. Transparent RRMS-mediated processes and committed and skillful people are crucial to ensure high quality. RRMS should also help users to identify which pieces of information are the most critical in each phase of the process, for example, by sending reminders and highlighting the required fields in the different phases.

Providing information at the right level of detail

Appropriate granularity of information facilitates decision making and eliminates extraneous activities required to decompose or summarize information [5]. The right level of detail depends on the situation. For example, highly mature requirements and release management processes can leverage much more detailed (quantitative) information than immature processes. RRMS must incorporate and represent requirement- and release-related information in granularity levels that are useful for different process contexts and roles [48].

4. A Meta-Design of the ISDT for RRMS

This section first outlines a generic meta-design for RRMS based on the analyses of interview transcripts, RRMS, and the literature review. It covers a subset of the meta-requirements specified in the previous section. The section concludes by explicating the linkages between the meta-requirements and the meta-design to validate the meta-design and to justify its scope.

4.1 Information Model for Integrated Requirement Management and Release Management Process

To design an effective requirement management and release management process, the information model for

the process must be defined. We have synthesized a simple but scalable model based on the literature review and a detailed examination of RRMS. The experts of the organization have reviewed and accepted the model. It consists of four entities used in the integrated process and links between and within the entities to enable traceability, hierarchical composition (i.e., each entity can consist of any number of the same type of entities), and appropriate granularity of information (Figure 1): Customer Requirement, Requirement, Feature and Release. Requirement and Release are the base entities. The other two are derivatives of Requirement.

Customer Requirement is used for requirements from the external environment. Internal and external requirements are separated to meet the meta-requirements related to platform-based product development and information security. For example, customer requirements are business critical and can provide competitive advantage by enabling organizations to focus efforts on implementing the differentiating and high-value adding requirements. Access rights for them and for requirements have to be defined and enacted differently.

Requirement is used for internal requirements developed by internal product creation processes within an organization or a network of companies collaborating to create a joint platform for effective product development. Requirement can thus be used for platform requirements related to the platform offering. In the platform context, customer requirements are used as the basis for developing derivative products on top of the platform. Separation of Customer Requirement and Requirement also facilitates change management. Requirements can be changed only through negotiations with their originators. Negotiations with external requirement suppliers are more challenging than with suppliers of internal requirements.

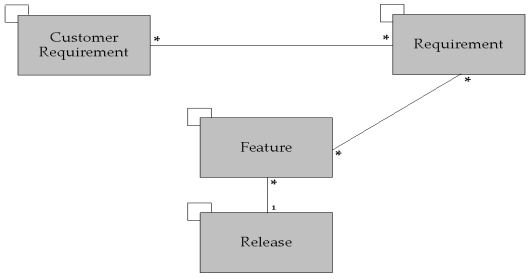

Figure 1. Information model of the meta-design of the ISDT for RRMS

Features denote intended behaviors or properties of software-intensive systems. They are usually created and managed as hierarchical solutions to the problems Requirements identify [55]. The solutions may reflect, for example, business processes, organizational structures, or product architectures. Feature is the largest entity in the information model containing the technical specification, workflow planning, and implementation. By using Feature as a basis of implementation and technical specification, requirements become more manageable and there are clear traceability links to the origins of Feature instances and to implementation phases, that is, specifications, responsible teams, and realized pieces of software code.

Release is used to group the implemented features into manageable entities that are released. Releases can also be organized hierarchically into, for example, platform and product releases.

The relationships between entities are explained next. Customer requirements are allocated to one or more requirements, which, in turn, are allocated to features. Highest level requirements are typically large scale system-level definitions of business problems. Dividing them into sub-problems enables more accurate project cost, schedule, and effort estimation and better workflow management. For example, one feature can be implemented by one team. Features describe implementable partial solutions to the business problems. Releases are comprehensive, valuable solutions consisting of releases and features

4.2 Generic Structures of Entities

This section introduces generic structures and attributes of the entities presented in the information model. According to the ISDT, RRMS instances should include at least these structures and attributes to be effective.

Requirement and Customer Requirement

Table 3 presents the generic structure of Requirement and Customer Requirement by revising and elaborating on the work of Salo & Käkölä [48]. Next, each class within the structure is presented.

Description describes the intent of and justification for a requirement and a customer Requirement. Version indicates the version number of the document. Name and ID are used for identification and traceability.

Origin describes where the requirement comes from and when. For customer requirements the sources are external organizations. For requirements, sources are customer requirements and internal organizational units.

Categorization describes the parts (i.e., platform, product, and responsible person) of the product and the development organization the requirement or customer requirement is related to. Product platform works as the base architecture for derivative products. Requirements and customer requirements will be linked to appropriate platforms and final products. Responsible persons further develop requirements and customer requirements.

Analysis is used to probe the implications of the requirement. Priority and customer value can be handled as one attribute, but organizations needing sophisticated valuation processes should divide them in two. Customer need is used to describe the detailed business case, which the requirement or customer requirement is trying to solve. If

there is a firm deadline by which the requirement needs to be implemented and released for use of customer(s) (e.g., in their products), the deadline must be made explicit. The total cost and effort need to be estimated [29] to determine whether the requirement is feasible from economic and schedule viewpoints. Risks associated with the (customer) requirement need to be assessed. Examples of requirement statuses include: New - Categorized – Analyzed – For Review – Approved / Rejected / Postponed [48].

Workflow describes what should be done next to this requirement or customer requirement and by whom. Customer requirements and requirements are allocated, respectively, to requirements and features and responsible persons are assigned.

History is used to provide information about all prior changes and editors of requirements documents [48]. It enables traceability and the development of organizational memory that is especially useful when routines break down unexpectedly and the reasons for the breakdowns must be found and eliminated to continue the effective execution of routines [33].

Table 3. Generic Structure of Requirement and Customer Requirement

Class	Question	Attributes
Description	What is the requirement about?	Name ID Description Rationale Version
Origin	Where does the requirement come from?	Author Source Date of creation
Categorization	What parts of the product and the development organization is the requirement related to?	Platform Product Responsible Person
Analysis	What are the implications of the requirement?	Status Priority Customer need Deadline for the customer need Customer value Risks *Required effort*
Workflow	What should be done to this requirement next? By whom?	Allocation to Requirements/Features Assignment to Requirement/Feature responsible
History	What has been done to the requirement? When?	Information about all prior edits, editors, and changes

Feature

Each class within the Feature-entity (Table 4) is presented in this section.

Description tells the intent of and justification for the Feature. *Origin* indicates the author, date of creation, and Requirements from which the Feature is allocated.

Categorization links Features to products and/or product platforms and identifies the person having the feature responsibility. Because Feature is an entity for managing detailed implementation, it has an attribute containing traceability links to technical specifications, documentations and code. Features tend to have complex dependencies [55]. For example, a feature may be incorporated into a product only if its parent feature is also included.

Analysis contains the other attributes that Requirement and Customer Requirement have, with the exception of Customer value-attribute used to decide whether requirements or customer requirements should be implemented or not. The required work effort needs to be estimated to help teams in their work allocation and scheduling.

Workflow consists of detailed task descriptions together with traceability links to provide the guidelines for implementation work and to enable organizational learning through, for example, post-mortem analysis (i.e., what was planned vs. realized). When starting the work, Features are assigned to responsible teams or persons.

History tells what has been done to Feature, when and by whom.

Table 4. Generic Structure of Feature

Class	Question	Attributes
Description	What is the feature about?	Name ID Description Rationale Version
Origin	Where does the feature come from?	Author Source Requirements Date of creation
Categorization	What parts of the product and the development organization is the feature related to?	Platform Product Responsible Person Traceability links (e.g. documentation, code)
Analysis	What are the implications of the feature?	Status Priority Customer need Risks Required work effort
Workflow	What should be done to this feature next? By whom?	Task description Assignment to teams/persons Assignment to Release Date when Feature is ready for Release
History	What has been done to the feature? When?	Information about all prior edits, editors, and changes

Release

Classes within the Release-entity (Table 5) that need elaboration are explained in this section.

Description describes what the release is about. For example, a release may fix some specific quality

problems without providing new functionality.

In *Origin,* Source Features attribute indicates which features are included in a release.

In *Categorization*, a release is related to specific product platforms and/or products and has a responsible person.

Analysis describes the statuses of a release such as. Planned – Ready to be Released (i.e., all features belonging to a release are ready) – Released.

Because releases constitute manageable and releasable entities, the only *Workflow* related attribute tells the estimated release date. This information together with History is adequate for organizational learning and performance monitoring (e.g., planned vs. actual release date).

Table 5. Generic Structure of Release

Class	Question	Attributes
Description	What is the release about?	Name ID Description Version
Origin	Where does the release come from?	Author Source Features Date of creation
Categorization	What parts of the product and the development organization is the release related to?	Platform Product Responsible Person
Analysis	What are the implications of the release?	Status Priority
Workflow	What should be done to this release next? By whom?	Release date
History	What has been done to the release? When?	Information about all prior edits, editors, and changes

4.3 Validating and Scoping the Meta-Design by Analyzing how it Meets the Meta-Requirements

Prioritization and valuation of requirements and the allocation of requirements into releases

Prioritization and valuation of requirements is enabled by the entities Requirement and Customer Requirement. Their attributes Priority and Customer Value are used to store and access the prioritization and valuation information in RRMS. The prioritization and valuation methods are not included in the meta-design for two reasons. First, the literature provides hardly any methods that are generalizable and scalable to meet the needs of complex industrial environments where multiple interdependent releases of interdependent products and platforms are planned simultaneously [35]. Second, the product programs of the case organization used different prioritization methods and tools because they differed in size, duration, and product maturity. The meta-design ensures that RRMS can provide the information for using the methods and store and share the results organization-wide.

Allocation of requirements into releases is enabled transitively through features, that is, requirements and customer requirements are allocated to features, which are linked to releases. Releases provide implemented functionality and are thus linked to features directly.

Traceability

The information model enables bi-directional traceability between entities through Origin- and Workflow-classes. In Customer Requirement and Requirement, Source-attribute is used for backward traceability and Allocation to Features-attribute enables forward traceability to features. Source Requirement- and Assignment to Release-attributes of Feature enable, respectively, backward and forward traceability from features. Traceability links-attribute enables the traceability from Features to implementation specific documentation and software code.

Change management and impact analysis

Change management is facilitated by the History-class in all entities. Change requests can be considered as normal (customer) requirements, analyzed, linked to the respective existing requirements in RRMS that are within the scope of change, and implemented and released following the integrated requirement and release management process. Impact analysis is enabled by Categorization- and Analysis-classes. Platform- and Product-attributes show the organizational entities affected by each requirement and release. Customer value- and Required effort-attributes are used to decide the feasibility of implementing a (customer) requirement.

Content ownership

Content ownership is determined through the Categorization-class. The attribute Responsible Person explicitly defines the content ownership.

Accountability of experts responsible for release planning tasks

All entities have Responsible Person-attributes facilitating the accountability of experts. Each release thus has to specify who is responsible for planning, which features are released in which release. The meta-design does not detail the metrics that could be used for measuring performance. However, it can be used as a baseline for sophisticated measurement systems.

Management and coordination

The meta-design supports management and coordination, for example, by explicating the schedules imposed on various entities, the products and organizational units the entities are related to, and the workflows the entities are subjected to.

Version management of requirement documents

Description- and History-classes enable the version management of requirements (and other entities) by, respectively, numbering versions and showing the actors involved with each version and the actions taken.

Release re-planning

The individuals responsible for particular features and releases decide about release re-planning (e.g., features belonging to a release cannot be released because they are unexpectedly delayed). Feature and release documents

and bidirectional traceability links between them (stored in Assignment to Release- and Source Features-attributes) facilitate the implementation of the meta-requirement.

When there are numerous interdependencies between releases, between features, and between features and releases, the appropriate data stored in RRMS can be transferred into a release re-planning and optimization system (c.f., [9;47]) for analysis. Prescribing the features of such systems is beyond the scope of the ISDT for RRMS because the systems are algorithmically complex, enable cost, effort and schedule estimation based on historical data [17], and operate on a higher level of analysis than RRMS where strategic and operational decisions (e.g., about the common features within and across the product lines) are taken based on information in RRMS and other systems. Another IS design theory needs to be built for such systems and interfaced with the ISDT for RRMS.

5. Conclusions and Future Research

This research focused on the RRMS-enabled multi-site and platform-based product development. It synthesized the meta-requirements and a partial meta-design of a simple but comprehensive and scalable ISDT for RRMS to help practitioners in both small and large organizations implement and evaluate RRMS solutions. The validity of the partial ISDT was enhanced by using methods such as the analysis of a RRMS instance in a case organization and by explicating the meta-requirements met by the meta-design. Due to space limitations, the design product effectiveness hypotheses of the ISDT (clarifying the expected organizational benefits from using a RRMS instance derived from the class of RRMS) have been beyond the scope of this article. The hypotheses are needed for the empirical validation of the theory in future research. RRMS is expected to reduce resources needed in product development, shorten time-to-market by ensuring right-information is available at the right time for right people, improve customer satisfaction by ensuring that requirements are transformed efficiently to product features, and improve quality by minimizing the number of errors during development and easing up error tracking.

Future research is necessary to devise extensions to the ISDT. Possible extensions include (1) strategic release management [46] that takes a more strategic and long-term view than release project management the ISDT deals with and (2) the inclusion of a Component entity in the information model of the meta-design to help projects better find and reuse implementation level assets that meet their needs. Such a solution further improves the generalizability of the ISDT because platform organizations often combine both software and hardware in their assets. Our preliminary industrial experiences show that Component is useful especially if it describes how and when Component instances have been tested.

The single case study methodology may not provide a sound basis for generalization [54]. Therefore, new case studies and action research projects are necessary to make the ISDT more credible for IS designers and researchers. Design science research leveraging the methods of action research [25] helps examine the applicability of the ISDT by finding out to what extent organizations that want to acquire or design and implement RRMS systems can utilize the ISDT for those purposes. The ISDT can then be revised as necessary.

6. References

1. Adelson, B. and Soloway, E. (1985). The Role of Domain Experience in Software Design. IEEE Transactions on Software Engineering, 11(11), 1351-1360.
2. Akao, Y. (1990). An Introduction to Quality Function Deployment. In Akao, Y. (ed.), Quality Function Deployment. Integrating Customer Requirements into Product Design, Productivity Press, 3-24.
3. Argyris, C. and Schon, D.A. (1995). Organizational Learning II: Theory, Method, and Practice. Prentice Hall.
4. Van Aken, J. E. (2004). Management Research Based on the Paradigm of the Design Sciences: The Quest for Field-Tested and Grounded Technological Rules. Journal of Management Studies, 41(2), 219-246.
5. Aubert B. A., Vandenbosch B. and Mignerat M. (2003). Towards the Measurement of Process Integration. Proceedings of the Annual Conference of the Administrative Sciences Association of Canada. Halifax, Nova Scotia
6. Bosch, J. (2002). Maturity and Evolution in Software Product Lines:Approaches, Artefacts and Organization. Proceedings of the Second Software Product Line Conference (SPLC2). Springer Lecture Notes in Computer Science, 257-271.
7. Brown, S. L. and Eisenhardt, K.M. (1995). Product Development: Past Research, Present Findings, and Future Directions. Academy of Management Review, 20(2), 343-378.
8. Burchill, G. and Fine, C.H. (1997). Time versus Market Orientation in Product Concept Development: Empirically-based Theory Generation. Management Science, 43(4), 465-478.
9. Carlshamre P. (2002). Release Planning in Market-Driven Software Product Development: Provoking an Understanding. Requirements Engineering, 7, 139-151.
10. Carmel, E. and Agarwal, R. (2001). Tactical Approaches for Alleviating Distance in Global Software Development. IEEE Software, March/April, 22-29.
11. Carmel, E. and Agarwal, R. (2002). The Maturation of Offshore Sourcing of Information Technology Work. MIS Quarterly Executive, 1(2), 65-78.
12. Ciborra, C. (1996). The Platform Organization: Recombining Strategies, Structures, and Surprises. Organization Science, 7(2), 103-118.
13. Curtis, B., Krasner, H. and Iscoe, N. (1988). A Field Study of the Software Design Process for Large Scale Systems. Communications of the ACM, 31(11), 1268-1287.
14. Daskalantonakis, M.K. (1992). A Practical View of Software Measurement and Implementation Experiences within Motorola. IEEE Transactions on Software Engineering, 18(11), 998-1010.
15. Davis, A. (1992). Software Requirements: Objects, Functions, and States. Prentice Hall, Eaglewood Cliffs, NJ.
16. El Sawy, O. (1998). Minding Your Own Business

Processes: The BPR Learning Book. New York: McGraw-Hill.
17. Forselius, P. and Käkölä, T. (2009). An Information Systems Design Product Theory for Software Project Estimation and Measurement Systems. Proceedings of HICSS-42. IEEE.
18. Forsgren, P. and Daugulis, A. (1998). Requirements Engineering In Control Center Procurement Projects: Practical Experiences from the Power Industry. Proceedings of the 3rd International Conference on Requirements Engineering. IEEE.
19. Gilb, T. (1998). Principles of Software Engineering Management. Addison-Wesley, Reading, MA.
20. Greer, D. and Ruhe, G. (2004). Software Release Planning: an Evolutionary and Iterative Approach. Information and Software Technology, 46, 243-253.
21. Griffin, A.J. and Hauser, J.R. (1992). Patterns of Communication among Marketing, Engineering and Manufacturing, - a Comparison between Two New Product Teams. Management Science, 38(3), 360-372.
22. Grynberg, A. and Goldin, L. (2003). Product Management in Telecom Industry – Using Requirements Management Process. Proceedings of IEEE International Conference on Software: Science, Technology and Engineering (SwSTE '03).
23. Halbleib, H. (2004). Requirements Management. Information Systems Management 21(1), 8-14.
24. Herbsleb, J.D., Moitra, D. (2001). Guest Editors' Introduction:Global Software Development. IEEE Software,18(2),16-20.
25. Hevner, A.R., March, S.T., Park, J. and Ram, S. (2004). Design Science in Information Systems Research. MIS Quarterly, 28(1), 75-105.
26. Hrones, J.A. Jr., Jedrey, B.C. Jr. and Zaaf, D. (1992). Defining global requirements with distributed QFD. Digital Technical Journal, 5(4), 36-46. http://www.hpl.hp.com/hpjournal/dtj/vol5num4/vol5num4art3.pdf.
27. ISO (2005). ISO/IEC TR 19759. Software Engineering -- Guide to the Software Engineering Body of Knowledge (SWEBOK). International Organization for Standardization, http://www.iso.org.
28. ISO (2006). ISO/IEC 14143 Information technology -- Software measurement -- Functional size measurement-Part 6. International Organization for Standardization, http://www.iso.org.
29. Jones, C. (2008). Applied Software Measurement, Third Edition. McGraw-Hill.
30. Kotonya, G. and Sommerville, I. (1998). Requirement Engineering: Processes and Techniques. John Wiley & Sons Ltd, England.
31. Käkölä, T. (2008). Best Practices for International eSourcing of Software Products and Services. Proceedings of HICSS-41. IEEE.
32. Käkölä, T., Dueñas, J. (Eds.) (2006). Software Product Lines: Research Issues in Engineering and Management. Springer.
33. Käkölä, T. and Koota, K. (1999). Dual Information Systems: Supporting Organizational Working and Learning by Making Organizational Memory Transparent. Journal of Organizational Computing and Electronic Commerce, 9(2&3), 205-232.
34. Käkölä, T. and Taalas, A. (2008). Validating the Information Systems Design Theory for Dual Information Systems. Proceedings of the 29th International Conference on Information Systems, Paris (in press).
35. Lehtola, L. and Kauppinen, M. (2006). Suitability of Requirements Prioritization Methods for Market-driven Software Product Development. Software Process Improvement and Practise, 11(1), 7-19.
36. Van der Linden, F., Schmid, K. and Rommes, E. (2007). Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer.
37. Markus, M. L., Majchrzak, A., and Gasser, L. (2002). A Design Theory for Systems That Support Emergent Knowledge Processes. MIS Quarterly, 26(3), 179-212.
38. McGrath, M.E. (2001). Product Strategy for High-Technology Companies: How to Achieve Growth, Competitive Advantage, and Increased Profits. Second Edition, New York, NY: McGraw-Hill.
39. Meyer, M.H. and Lopez, L. (1995). Technology Strategy in Software Products Company. The Journal of Product Innovation Management, 12(4), 294-306.
40. Meyer, M.H. and Selinger, R. (1998). Product Platforms in Software Development.Sloan Management Review,40(1),61-74.
41. O'Reilly, C.A. (1982). Variations in decision makers' use of information sources:The impact of quality and accessibility of information. Academy of Management Journal, 25(4), 756-771.
42. Perry, D.E., Staudenmayer, N.A. and Votta, L.G. (1994). People, Organizations, and Process Improvement. IEEE Software, 11(4), 36-45.
43. Pohl, K., Böckle, G. and Van der Linden, F. (2005). Software Product Line Engineering. Springer.
44. Ramasubbu, N., Krishnan, M. S., Kompalli, P. (2005). Leveraging Global Resources: A Process Maturity Framework for Managing Distributed Development.IEEE Software,22(3),80-86.
45. Regnell, B., Höst, M., Natt och Dag, J., Beremark, P. and Hjelm, T. (2001). An Industrial Case Study on Distributed Prioritization in Market-Driven Requirement Engineering for Packaged Software. Requirements Engineering, 6(1), 51-62.
46. Rautiainen, K., Lassenius, C., Vähäniitty, J, Pyhäjärvi, M. and Vanhanen, J. (2002). A Tentative Framework for Managing Software Product Development in Small Companies. Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS), 3409-3417.
47. Ruhe, G. and Saliu, M. O. (2005). The Art and Science of Software Release Planning. IEEE Software, 22(6), 47-53
48. Salo, A. and Käkölä, T. (2005). Groupware Support for Requirements Management in New Product Development. Journal of Organizational Computing and Electronic Commerce, 15(4), 253-284.
49. Van De Ven, A.H. (1986). Central Problems in the Management of Innovation. Management Science, 32, 590-607.
50. Walls, J. G., Widmeyer, G. R. and El Sawy, O. (1992). Building an Information System Design Theory for Vigilant EIS. Information Systems Research, 3(1), 36-59.
51. Walls, J.G., Widmayer, G.R. and El Sawy, O. (2004). Assessing Information System Design Theory in Perspective: How Useful was our 1992 Initial Rendition? Journal of Information Technology Theory and Application, 6(2), 44-58.
52. Wesselius, J. (2006). Strategic Scenario-Based Valuation of Product Line Roadmaps. In T. Käkölä & J.C. Dueñas (Eds.), Software Product Lines: Research Issues in Engineering and Management, Springer, 53-89.
53. Wiegers, K.E. (2003). Software Requirements: Practical Techniques for Gathering and Managing Requirements, 2nd edition. Microsoft Press, Washington.
54. Yin, R.K. (2003). Case Study Research: Design and Methods, third edition. Sage Publications.
55. Zhang, W., Mei, H. and Zhao, H. (2006). Feature-driven Requirement Dependency Analysis and High-level Software Design. Requirements Engineering, 11,:205–220.