Information and Software Technology 52 (2010) 720–732

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Incremental method evolution in global software product management: A retrospective case study [☆]

Inge van de Weerd [*], Sjaak Brinkkemper, Johan Versendaal

Institute of Information and Computer Sciences, University of Utrecht, The Netherlands

ARTICLE INFO

Article history:
Received 8 March 2010
Accepted 8 March 2010
Available online 17 March 2010

Keywords:
Software product management
Requirements management
Method engineering
Meta-modeling
Incremental method evolution
Distributed development

ABSTRACT

Company growth in a global setting causes challenges in the adaptation and maintenance of an organization's methods. In this paper, we will analyze incremental method evolution in software product management in a global environment. We validate a method increment approach, based on method engineering principles, by applying it to a retrospective case study conducted at a large ERP vendor. The results show that the method increment types cover all increments that were found in the case study. Also, we identified the following lessons learned for company growth in a global software product management context: method increment drivers, such as the change of business strategy, vary during evolution; a shared infrastructure is critical for rollout; small increments facilitate gradual process improvement; and global involvement is critical. We then claim that method increments enable software companies to accommodate evolutionary adaptations of development process in agreement with the overall company expansion.

1. Introduction

The software market has made a shift from developing customized software to primarily developing software as a standard product [36]. Many of these product software companies face difficulties in their product development processes [7,8]. An important solution to these problems is to implement a good software product management function. Ebert describes an empirical study with data from 178 industry projects that shows that time to market, schedule adherence and handover quality all improve with the strengthening of a coherent product management role [10]. The product management practice contains activities that are carried out on operational, tactical, and strategic levels. In earlier research, we identified the following four process areas within software product management: requirements management, release planning, product roadmapping, and portfolio management [31]. Activities in the first two areas are mainly on a operational level, whereas the latter two contain tactical and strategic activities.

Another trend that can be seen in many, especially globally operating, product software companies is company growth. It has been argued that the growth of a company is associated with the rising information intensiveness of production [5]. A rapidly expanding company needs a reorganization of its structural systems [14]. This also holds for the processes and methods that are used. Other authors emphasize that to make a strategy for fast growth work, the structure and processes should be changed in a way that lets them acquire or create specific knowledge about new technologies, customers and industries [12]. Also, strategic investments need to be made in a support infrastructure [28]. Especially in globally operating organizations, it is important to focus on these issues, since many risks are associated with the assumed benefits [6], such as more overhead in communication, coordination and control, and a reduced collaborative time window due to the temporal difference.

To control company growth, organizations use process improvement methods, such as SPICE [11] and CMM [22], to implement new and better practices. These practices can be implemented in different ways and can be evolutionary or revolutionary in nature. In this research, we focus on an evolutionary approach instead of a revolutionary approach, for several reasons: (a) it is a fundamental way to reduce risk on complex improvement projects [13]; and (b) we observe in practice that methods evolve [35]. This evolutionary approach has been subject of research in various scientific studies. Studies have been carried out to find the best approach to instigate a process improvement [24,29] and research has been done to the

[☆] This article was submitted in response to a special call for papers on Situational Requirements Engineering Processes in a Global Context. The paper was recommended for publication by Prof. Pär J. Ågerfalk, Uppsala University.

[*] Correspondence to: I. van de Weerd, Department of Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508TB Utrecht, The Netherlands. Tel.: +31 30 253 1756; fax: +31 30 251 2804.

E-mail addresses: i.vandeweerd@cs.uu.nl (I. van de Weerd), s.brinkkemper@cs.uu.nl (S. Brinkkemper), j.versendaal@cs.uu.nl (J. Versendaal).

0950-5849/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2010.03.002

key success factors that influence software process improvement [23]. However, in 2002, it was estimated that still 70% of software process improvement projects failed [27]. In addition, it has been suggested that it is necessary to link any software process improvement program to business goals [9].

To summarize the problem: Product software companies need to implement software product management practices, but at the same time, need to improve their processes to stay in pace with the growth of their organization. Since many software process improvement projects fail, we want to get more insight into this process.

A retrospective case study at a large ERP (Enterprise Resource Planning) vendor is carried out to investigate the method increments that took place over a period of 12 years and eleven countries. Our investigation focuses on the software product management processes requirements management and release planning. The reason for this is that these processes were well documented by the case study company, not only in the last years, but also in earlier years. In addition, the other software product management activities (part of product roadmapping and portfolio management), were at the case company not carried out by product managers. For example, issues concerning partnering and contracting were handled by a separate department, and product lifecycle management and roadmapping were carried out by the management of the company. Therefore, we narrow down our research scope to requirements management and release planning.

We first analyze the general *method increments*, as introduced [34], which are collections of method fragments that have been changed (inserted, modified, deleted) during a certain method adaptation. Secondly, we explore what the drivers for each method increment were, i.e. what circumstances or events in the organization triggered the method increments. This leads us to the following research question:

Which method increment types occur in incremental method evolution, and which general increment drivers can be identified?

In the next section, we first explain our research approach, including the meta-modeling technique. In Section 2, we describe the design of our retrospective case study, as well as the possible validity threats. Section 3 describes the results of the case study, as well as the analysis in which the method increment types are compared with the found method increments in the case study. Then, in Section 4, we describe the lessons learned from the case study. Section 5 presents an overview of related work. Finally, in Section 6, we describe our conclusions and future research.

2. Research approach

In this paper, we report about a retrospective case study. We chose this type of research for several reasons. Firstly, an exploratory case study makes it possible to get an in-depth understanding of a contemporary phenomenon where the investigator has little control over events [37]. Secondly, the retrospective nature of the case study made it possible to investigate the changes in an organization over a period of twelve years; a time period which is almost impossible to investigate through regular longitudinal research.

Our case study starts with retrospective interviews with several product managers and a document study on the actual deliverables of the used methods in the given time period. Examples of these deliverables are release plans, requirement documents and roadmaps. Based on the interviews and document study, we model the method increments in process-deliverable diagrams (PDDs), which are explained 2.2, and analyze them. Finally, we identify how the method increments found in the case study correspond to our earlier identified elementary method increment types.

In the following sections, we first describe our case study design. Then, in Section 2.2, we describe the meta-modeling technique we used to model the method increments. We conclude by describing the threats to validity.

2.1. Research site

We carried out a case study at Infor Global Solutions (specifically the former Baan company business unit, hereafter called 'Baan'), a vendor of ERP software. In 1978, Baan was established as a book-keeping consulting company. Over the years, the company changed from a consultant company to a software vendor for businesses. Baan was quoted on the Nasdaq stock exchange as an independent company from 1995 to 2000. The time period that is covered in the case study ranges from 1994 to 2006.

We analyzed the evolution of the software development process at Baan, with emphasis on product management activities. During this period, the development centers of Baan were located in eleven different countries in four continents, as is depicted in Fig. 1. All development centers used a shared infrastructure, consisting of a requirements database, online development method, and the document management systems for storing all development documents [4]. Around the years 1998 – 2001 about 1500 engineers, managers and support staff were employed at Baan R&D. Later,

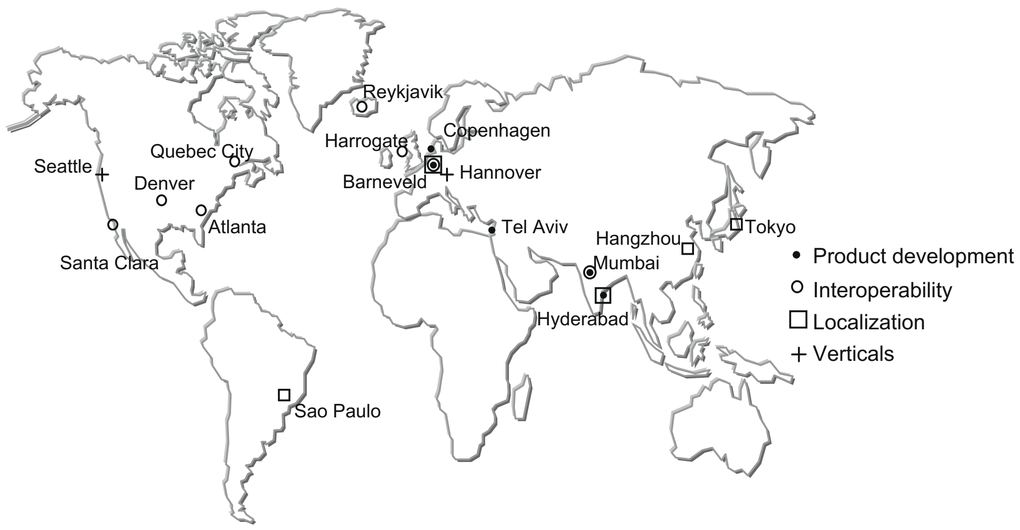


Fig. 1. Global distribution of R&D locations (situation August 2000).

the size was downscaled, which also implied that we had to include former employees in the case study.

The different locations had different main functions: *product development*, for developing Baan's ERP product line and the already integrated products of acquired companies; *interoperability*, to handle the integration of products of acquired companies with the main ERP product; *localization*, for localizing the products to international markets; and *verticals*, which focused on a specific market segment, e.g. automotive and aerospace. Note that the acquired companies also perform independent product development next to the interoperability. This is omitted in Fig. 1, as it is not relevant for the discussion. Also, sales and marketing, implementation services, and maintenance centers are not involved in this study, as these are not responsible for product management [30].

2.2. Case study design

Different sources are used to collect information. Firstly, several interviews were conducted with six former employees of the case company. The choice of these interviewees was based on their knowledge (they all played key roles in the Baan processes under investigation) and their availability. Two explorative 3-h interviews were conducted with the Process Engineer, who was responsible for the process improvement rollout in requirement management. Based on these interviews, the method evolution between 1994 and 2002 was modeled. This information was cross-checked by conducting 2-h follow-up interviews with five other employees of Baan, consisting of two former (Senior) Product Managers, a Director ERP Development, a Manager ERP Product Ownership and a Software Engineering Process Group (SEPG) Manager. In these interviews, also the method snapshots of 1994, 1996, 2003, 2004 and 2006 were identified and modeled.

Secondly, a document study was carried out. Documentation provided by the Process Engineer was used to complement and validate the results from the interviews. This documentation consisted of process descriptions, templates and examples of methods and work products used in the period 1997 until 2006. In total, 26 documents have been analyzed. From the period before 1997 no documentation was available. We focused on the following case study questions, related to software product management:

– Which snapshots can you identify in the method evolution?
– Which methods were used per stage? Which activities can be distinguished?
– Which deliverables resulted from these methods?
– Which process difficulties arose in this stage?
– Why was an increment needed?

2.3. Modeling method increments

For the analysis of method increments, we use process-deliverable diagrams (PDDs), a meta-modeling technique that is based on UML activity diagrams and UML class diagrams [26,32]. The resulting PDDs model the processes on the left-hand side and deliverables on the right-hand side (see Fig. 2).

We follow standard UML [20] conventions, but some minor adjustments have been made for modeling development processes. Firstly, deliverables can be *simple* or *compound*. Simple deliverables do not contain any sub-deliverables and are visualized with a rectangle. Compound deliverables contain one or more sub-deliverables. Compound deliverables can be open, visualized with an open shadow, to indicate that it contains sub-deliverables. The sub-deliverables can be shown in the same diagram, by using aggregation, or in another diagram (for example for space saving). Closed compound deliverables, visualized with a closed shadow, indicate that sub-deliverables exist, but are not relevant in this context. Similarly, open and closed activities are used in the diagram. Dotted arrows indicate which deliverables result from the activities. Further details on this modeling technique can be found in [32].

In [34,35], we have defined the notions of method evolution, snapshot and method increment. The PDD, visualized in Fig. 2, is called a *snapshot*, a model of the method as it was at a certain moment in time. The evolution of a method over time exists of a number of these snapshots. By comparing snapshots, method increments can be analyzed. In Fig. 2, we marked sub-activity 4 and its corresponding concept. We use this marking to show that

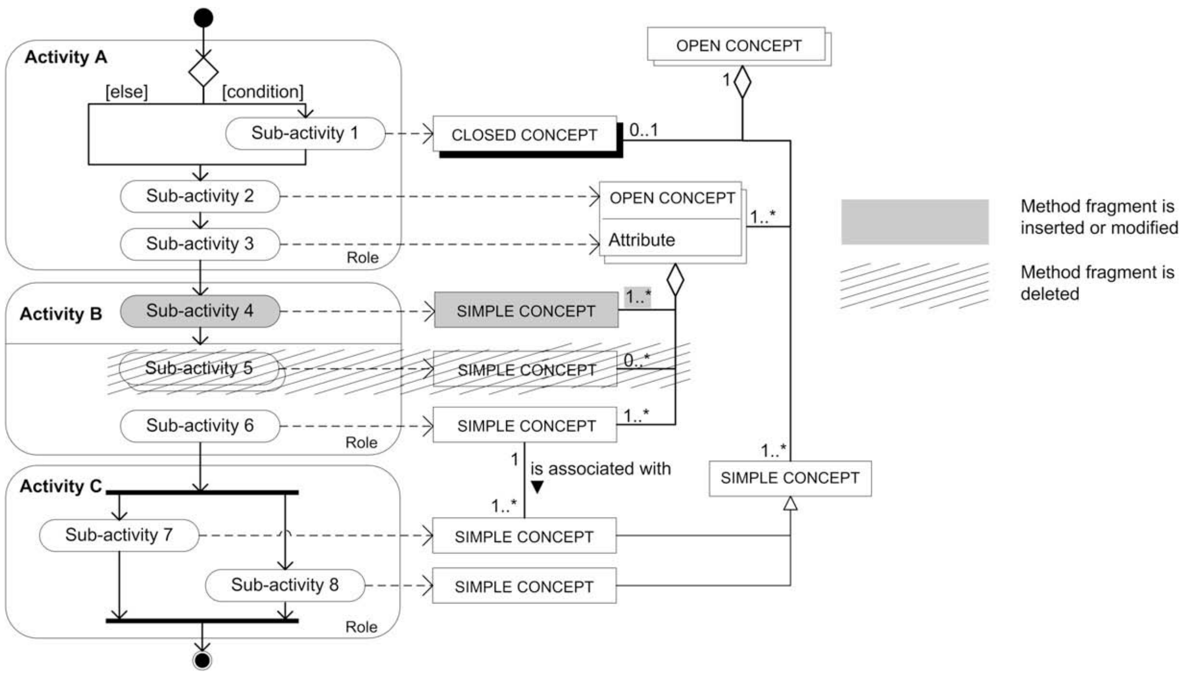


Fig. 2. Process-deliverable diagram.

this method fragment is inserted or modified compared to a snapshot earlier in time. Sub-activity 5 and its corresponding concept are also marked. This indicates that this method fragment is deleted, compared to the preceding snapshot.

Based on the meta–meta model of PDDs we identified a list of 18 elementary increment types.

- *Insertion* of a concept, property, relationship, activity node, transition, role.
- *Modification* of a concept, property, relationship, activity node, transition, role.
- *Deletion* of a concept, property, relationship, activity node, transition, role.

These elementary method increment types are used to decompose and analyze the method increments that are found in the retrospective case study.

2.4. Threats to validity

In exploratory research, the quality of the methodology should be judged on the basis of three types of validity [37]. *Construct validity* concerns the validity of the research method. We used multiple sources of data (interviewees and documents) to ensure that problems related to subjectivity and bias of data were avoided. Furthermore, we had key informants review the draft case study report.

Secondly, *external validity* concerns the domain to which the results can be generalized. We carried out the case study in the software product management domain in a product software company. We cannot say whether our findings apply to *all* globally operating product software companies. However, case study findings can be generalized to theory [15]. We are convinced that this study is an important contribution to the theory of incremental method evolution in software product management.

Finally, the *reliability* of the case study is concerned with demonstrating that the results of the study can be replicated. This threat is mitigated by maintaining a case study database that contains all the relevant information used in the case study. This case study database consists of interview notes, documentation and process-deliverable diagrams of all modeled methods. In addition, we used a case study protocol that has been used in earlier case studies to method increments in product software companies [35].

A few remarks should be made on the retrospective nature of the case study. A problem relating to this type of study is hindsight bias, which refers to the ability of people to reconstruct prior probabilities for an event after it has occurred [16]. In this study, it means that interviewees might have tended to regard the drivers that lead to the method increments as having been fairly predictable ("I knew it from the beginning..."), although this was in reality not the case. We minimized hindsight bias by using multiple interviewees and by backing their comments by documentation such as process descriptions and examples from the case study company.

3. Description of method increments

In 1985, Baan developed its first standard product: a salary software package. However, no requirements or designs were used and no documentation was written. This caused much frustration in the development process; improvement was needed. Some people started working on a standard method and, finally, in 1988 the first version of the *Baan Development Method* was created. As a result, developers started to document their software. In the early nineties, Baan counted 400 employees and had offices in America and Canada. It was necessary to define clear processes and methods. A Software Process Improvement (SPI) project was started, based on the Capability Maturity Model (CMM). The CMM level was estimated at level 1, so improvement was needed. In 1994, methods for software product management were defined for the first time. We use this as starting point for our analysis.

With the information gathered from the first interviews, we identified 14 method increments, as depicted in Table 1. For each method increment, we listed a description, date, and the countries that initiated the process change. Then, we analyzed the interviews and documents and modeled the 14 snapshots in PDDs, each representing a method that was used in a particular moment in time [33]. These snapshots were identified after a discussion with the interviewees. Based on the analysis, we also described the reason for every increment. In Section 4, we will further elaborate on this.

In the following sections we will analyze the increments. For the sake of brevity, we have only selected those increments with the most interesting method evolution. The other increments are described in [33]. Increments 1–6 and 12 and 13 are described in the sections below. We choose these method increments in order to be able to show all increment types. The first six increments illustrate the principle of including new fragments, and the later ones illustrate the deletion and modification of fragments.

For each method increment, we first describe the circumstances that instigated the increment, as well as the actions that were taken. Secondly, we provide a snapshot of the increment (expressed in a PDD). Thirdly, we give an explanation of the method increment. For clarification, we describe activities between hyphens (e.g. 'Write requirements document') and deliverables in capitals (e.g. requirements document). We only use this notation when we are explicitly referring to a PDD.

Table 1
Overview of method increments at Baan.

#	Increment	Date	Reason for increment	Initiating locations
1	Introduction requirements document	1994	Development management	Netherlands
2	Introduction design document	1996	Development management	Netherlands
3	Introduction version definition	1998, May	Development management	Netherlands and US
4	Introduction conceptual solution	1998, November	Business management	Netherlands and India
5	Introduction requirements database, division market and business requirements, introduction of product families	1999, May	Development management	Netherlands and India
6	Introduction tracing sheet	1999, July	Certification	Netherlands, US and India
7	Introduction product definition	2000, March	Departmental interfacing	Netherlands
8	Introduction customer commitment process	2000, April	Business management	Netherlands and US
9	Introduction enhancement request process	2000, September	Departmental interfacing	Netherlands and India
10	Introduction roadmap process		Departmental interfacing	Netherlands
11	Introduction process metrics	2002, August	Certification	Netherlands and India
12	Removal of product families and customer commitment	2003, May	Business management	US
13	Introduction customer voting process	2004, November	Business management	US
14	Introduction master planning	2006, October	Development management	Netherlands

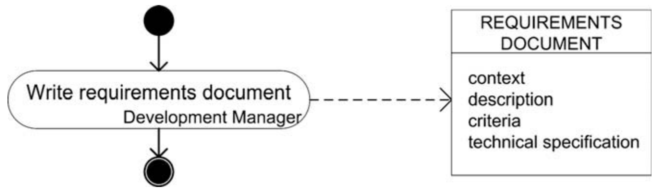


Fig. 3. Snapshot of increment 1.

3.1. Increment 1: documenting requirements

Before the snapshot depicted in Fig. 3, the *Baan Development Method* was used as a standard method to develop and document software. However, this method was restricted to development and testing stages, and no attention was given to requirements engineering.

In Fig. 3, increment 1 of the Requirements stage at Baan is visualized. The requirements were described by the Development manager in the REQUIREMENTS DOCUMENT, in order to separate the requirements from the design. This REQUIREMENTS DOCUMENT typically consisted of two pages and described mainly functional requirements. Key questions that were answered in this document were: what's it about, what should it do, and what are the criteria? In addition, the Development manager also described some high-level technical specification of the requirements of functionality.

3.2. Increment 2: separating requirements and design

In increment 1, Development managers were writing their technical specifications in the requirements document. Instead of describing the "*what*" question, the requirements document described the "*how*" question. This development driver caused the implementation of a method increment in which the requirements and design was being separated.

In Fig. 4, increment 2 of the requirements stage at Baan is visualized. We can divide this increment into two main adaptations. Firstly, the Technical specification is not described in the REQUIREMENTS DOCUMENT anymore but in another document, namely the DESIGN DOCUMENT. As is described in Section 1, this research focuses on requirements management and release planning. Therefore, we do not elaborate on this deliverable. The second change is the clear structure that has been introduced in the REQUIREMENTS DOCUMENT. In the REQUIREMENTS DOCUMENT, one or more REQUIREMENTS are described. Each REQUIREMENT has its own description and criteria.

Only one property in the concept REQUIREMENTS DOCUMENT has been deleted. The two other properties have been modified; they are now properties of a newly inserted concept: REQUIREMENT. Finally, a relationship has been added: the aggregation between REQUIREMENT DOCUMENT and REQUIREMENT.

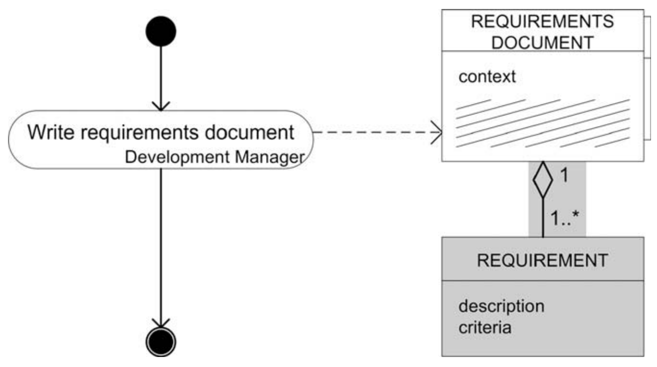


Fig. 4. Snapshot of increment 2.

3.3. Increment 3: version definition

In increment 2, the requirements were documented in a requirements document. However, much information was missing in this document and it was not focused on the release of the new product version.

To solve these problems, a team of three developers in Santa Clara, US (cf. 5.3–3), developed the version definition, which contained more information concerning the release. Note that the term 'version definition' is adopted from Baan's organizational terminology. In most literature, the term 'release planning' is used. The requirements were also being documented in more detail, and, furthermore, each requirement could now be assigned to a development group.

In Fig. 5, increment 3 of the Requirements stage at Baan is visualized. Looking at the process-side of the diagram, we can distinguish two main activities, namely Requirements management and Version definition. The following method fragments have been altered: First, the role has been changed. Instead of the Development manager, the Program manager is now responsible for the requirements management and release planning. The activity 'Write requirements document' is modified in 'Write version definition'. Furthermore, four activity nodes have been added: three new activities: 'Gather requirements', 'Review version definition' and 'Get version definition approval'; and one branch. This branch comes after 'Get version definition approval'. If this approval is obtained, the next activity can be started; otherwise the VERSION DEFINITION has to be reviewed again.

Along with these activity nodes, five new transitions are added to the method. At the deliverable side, several changes have been made as well: the concept REQUIREMENTS DOCUMENT is modified in VERSION DEFINITION, and multiple new properties have been added to this concept. Also, several new properties have been added to the existing concept REQUIREMENT. A new concept, GROUP, has been added along with two new constructions. One or more REQUIREMENTS are now owned by one or more GROUPS that have the responsibility for this REQUIREMENT. Finally, the relationship between REQUIREMENT and VERSION DEFINITION has been modified.

Concurrently with this method increment, the development processes of Baan, called Baan Development Method (BDM), were documented and put on the corporate intranet to support the worldwide distributed R&D departments [4] (cf. 5.3–1). In 1998, the BDM website was built in standard HTML, containing instructions, examples and templates for deliverables.

3.4. Increment 4: conceptual solution

Two main problems can be identified in method increment 3. Firstly, software engineers found it hard to read the version definition in order to build what is requested, and consequently did not build the precise features that were intended to be built. Also, the sales people found it hard to read the version definition and know exactly which new functionality was being built in the new product. Finally, it was difficult to store and trace requirements, since they were always part of a requirements document.

Two main improvements were made to solve the problems described above. Firstly, the conceptual solution was developed in the development center of Hyderabad, India (cf. 5.3–3). This document was used for internal and external communication related to requirements. Towards the customer, the requirement raiser, a conceptual solution provides guidance on the way in which

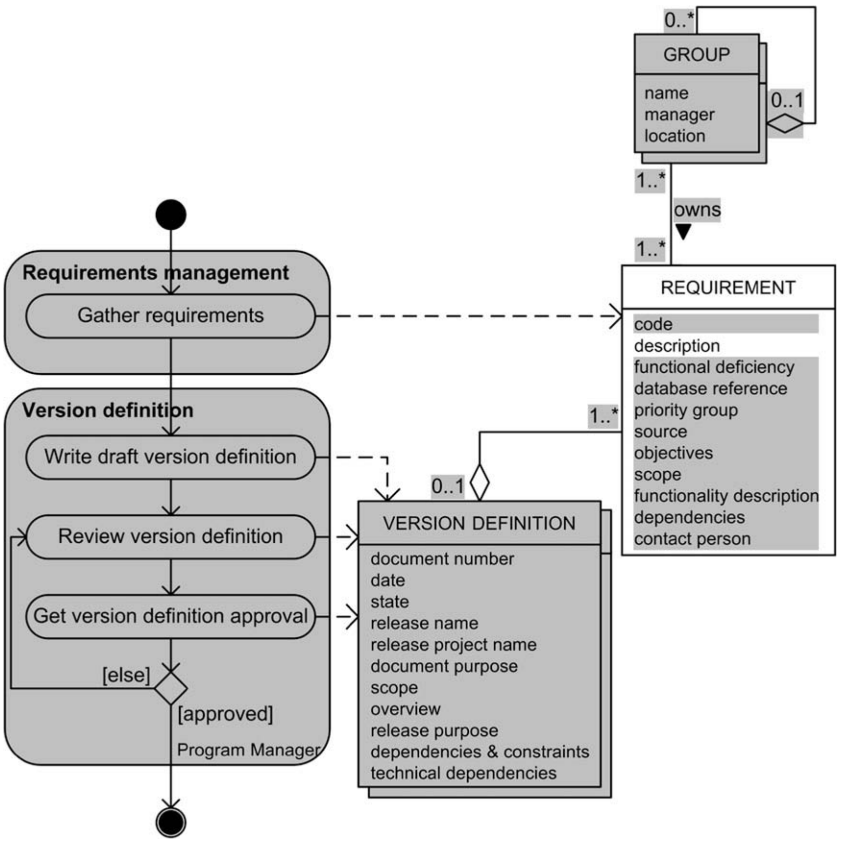


Fig. 5. Snapshot of increment 3.

requirements are elaborated and clarified in the perspective of the current Baan products. For the development department a conceptual solution provides input for design and realization of the concept.

After creating templates, instructions and examples, the method increment was first implemented in the main ERP development centers: Barneveld, Hyderabad and Mumbai. Later on, the acquired companies in Hannover, Quebec City, Tokyo and Santa Clara followed (cf. 5.3–4).

The second improvement concerned the separation of the requirements and the version definition. The version definition now only consisted of references to the requirements that are listed in the release table. In this way, requirements could exist without being part of a version definition and the version definition is readable again.

In method increment 4, the following changes have been made. First, a new activity node has been inserted: the activity 'Create conceptual solution'. This activity is open, i.e. it consists of several sub-activities. These sub-activities cannot be specified here due to space limitations. The full list of sub-activities is included in [33]. At the deliverable side, also several changes have been made. Three new concepts have been included. The RELEASE TABLE is as a reference table in the VERSION DEFINITION that refers to the REQUIREMENTS. In this way, the VERSION DEFINITION only has to include the references to the REQUIREMENTS instead of the entire descriptions. With the introduction of this concept, two relationships to VERSION DEFINITION and REQUIREMENT have further been inserted, and several attributes in REQUIREMENT have been modified. Secondly, the concept CAPACITY is inserted. With this concept the capacity of a GROUP is being administrated. Along with this concept, two relationships to GROUP and REQUIREMENT have been inserted. Finally, the concept CONCEPTUAL SOLUTION is inserted, which elaborates on the REQUIREMENTS, indicated by an association.

3.5. Increment 5: market and business requirements

In increment 4, no standard requirements gathering process was defined. External and internal stakeholders, such as customers, development, and consultants, sent their wishes to the Program manager, who added them to an Excel sheet as requirements. Secondly, the Program manager did not rewrite the requirements. Usually, they were described in the terminology of the customer, and multiple functionalities were often described in one requirement. At this time, hundreds of requirements were entering the company every month, so it was extremely difficult to keep the overview of the requirements database.

The next increment, presented in Fig. 6 shows a twofold solution to the earlier defined problems. The development center in the Netherlands developed a centralized requirements organization system in which a distinction was being made between market requirements and business requirements (cf. 5.3–1). Market requirements were requirements that are directly copied from the requirement raiser. In case a customer raised a certain requirement, it was documented as market requirement in the terminology of the customer. Then, business requirements were written by the product manager, which could be a redefinition of one or multiple market requirements. More information about this process can be found in [17]. The second part of the solution consisted of the use of the common component framework. Since the acquisition rate was quite high in this period, and multiple products were being acquired, several products and product lines exist that shared functionality. By identifying common components, and

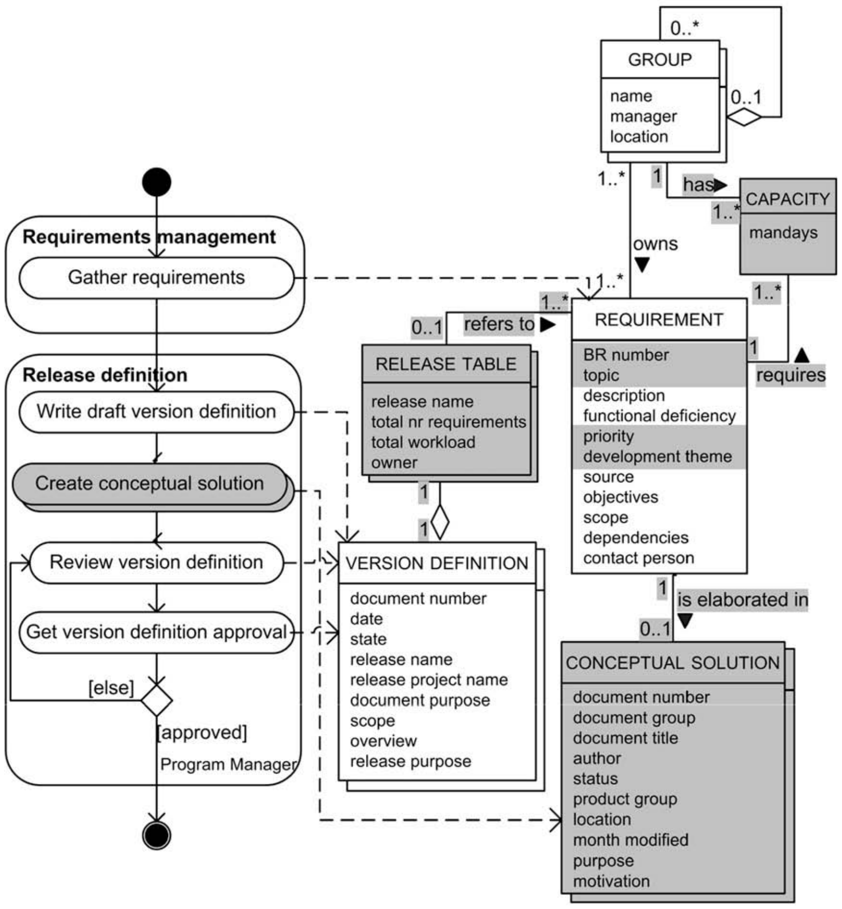


Fig. 6. Snapshot of increment 4.

relating that to the business requirements, the product managers tried to protect the company from double work.

As depicted in Fig. 7, several method fragments have been added in method increment 5. First, three new activities have been added, grouped in the 'Requirements management' main activity. The three activities are 'Create market requirement', in which a new MARKET REQUIREMENT, in the customer's terminology, is added to the requirements database; 'Maintain product line', in which COMMON COMPONENTS, or core assets, are identified and managed; and 'Create BR release independent', in which BUSINESS REQUIREMENTS are created, based on the stored MARKET REQUIREMENTS. Together with the introduction of the three activities, also several things changed at the deliverable side. REQUIREMENT is modified in BUSINESS REQUIREMENT. The concept MARKET REQUIREMENT is inserted to distinguish customer wishes from actual business or product requirements. Zero or more MARKET REQUIREMENTS are related to zero or more BUSINESS REQUIREMENTS. COMMON COMPONENT was added, with relationships to MARKET REQUIREMENT and BUSINESS REQUIREMENT. Finally THEME is inserted, which is only related to BUSINESS REQUIREMENT. Two extra small changes have been made, namely the introduction of an aggregation relationship for CONCEPTUAL SOLUTION and several extra attributes in CAPACITY.

In Fig. 8, a screenshot of the requirements database is illustrated. The picture shows a BUSINESS REQUIREMENT that is linked to several MARKET REQUIREMENTS.

3.6. Increment 6: tracing sheet

Increment 6 has a somewhat different driver for improvement than the previous method increments. This method increment was initiated in order to be CMMi certified. Several large customers demanded a certain CMMi level of Baan, so extra practices needed to be implemented to meet this demand.

In this case, the development center in the Netherlands decided to implement the tracing sheet, which purpose was to provide evidence of an association between a requirement and its source requirements, its realization, and its verification. Using the tracing sheet had several benefits: maintaining consistency between business requirements and conceptual solutions; checking the completeness of the work in progress; assisting in finding affected documents and components in case of a change request; and tracking the progress of the project.

In Fig. 9, increment 6 is depicted. In this snapshot, a large part of the concepts is omitted because of space limitations. The concepts that are not depicted in this snapshot, but are depicted in the snapshot of increment 5, have not been altered. The method increment consists of the addition of several method fragments. First, the activities 'Enter BR info in tracing sheet' and 'Enter CS info in tracing sheet' have been added to the process-side of the diagram. At the deliverable side of the PDD, two concepts have been added: TRACING SHEET, with four attributes, and TRACING TABLE, with ten attributes. These concepts are connected by an aggregation.

In Fig. 10, we illustrate a part of the tracing sheet. The example shows (among others) a sheet with three business requirements, with the description, linked market requirement, and some information concerning the conceptual solution in which the market requirement is detailed.

Note that increments 4, 5 and 6 were relatively small and introduced in a timeframe of nine months (see Table 1). These increments were deliberately kept small for the sake of easy implementation (cf. 5.3–2).

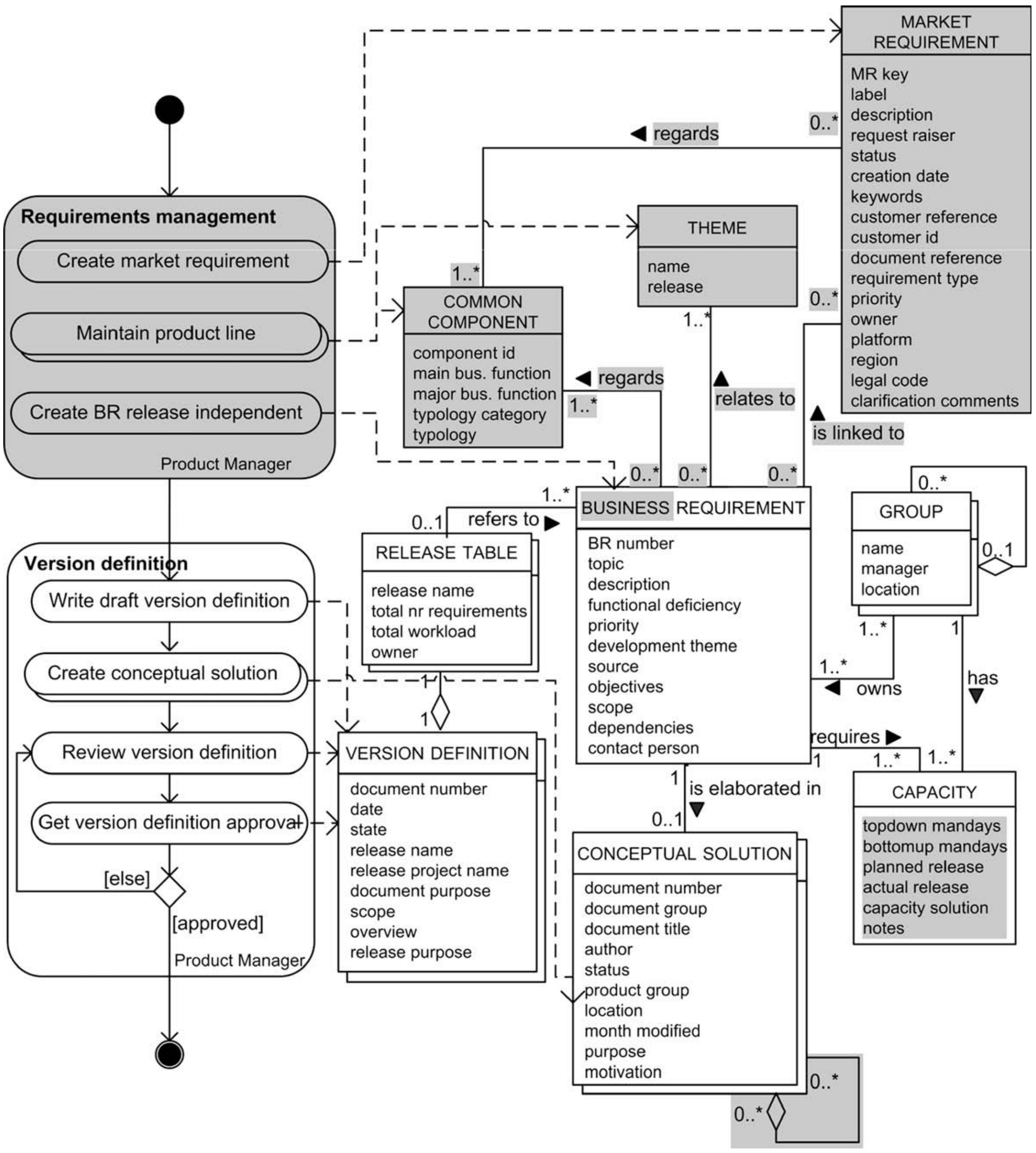


Fig. 7. Snapshot of increment 5.

3.7. Increment 12: omitting the common component framework

Method increment 12 takes place in 2003. In 2000, Baan got in serious financial problems and is finally taken over by Invensys, which was in that time Europe's second largest software company, just behind Germany's SAP.

Several problems concerning the common component framework are identified. First of all, product managers found it difficult to make the distinction between common and market-specific components. Secondly, in practice the developers did not use the common components, because they found it too much work. Thirdly, because of the many acquisitions in this period, and consequently the long range of acquired products, the common component framework was too difficult to maintain.

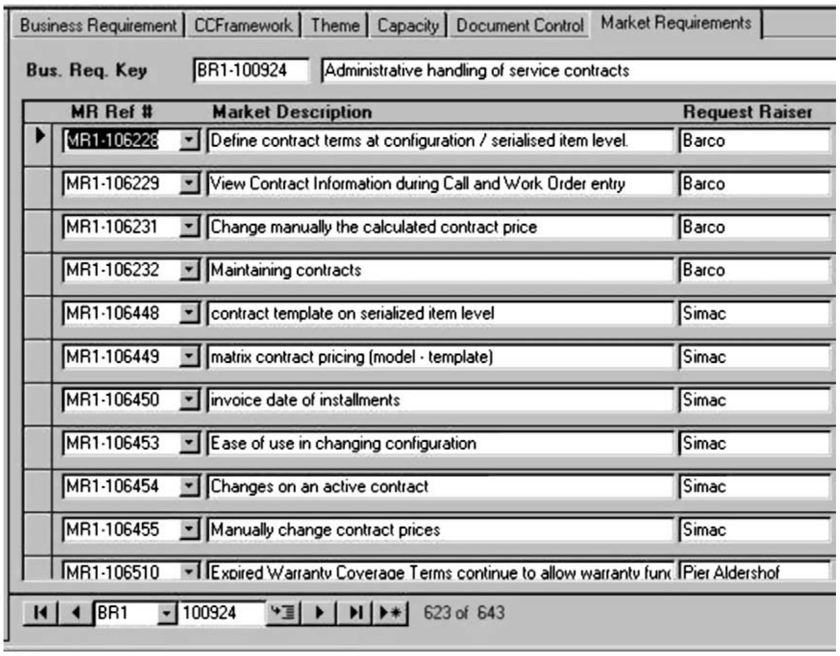
Altogether, the common component framework could not be maintained and management decided to eliminate the common component framework and only focus on the linkage between business requirements and market requirements.

In Fig. 11, the snapshot in increment 12 is depicted. One activity has been deleted, namely 'Maintain product line'. Also, two concepts are deleted: COMMON COMPONENT and THEME, together with their attributes and their relationships to MARKET REQUIREMENT and BUSINESS REQUIREMENT.

3.8. Increment 13: voting sheet

In June 2003, Invensys sold the Baan company to SSA Global, a large ERP vendor. A year later, method increment 13 took place. At that time, the requirements database had gotten so large that is was not useful anymore. At that moment several thousands of market requirements were stored and it was impossible to link them all properly to business requirements. The consequence was that customers complained because their demands were not met.

The solution was to leave the system of storing separate market and business requirements. Now, the market requirements are


Business Requirement | CCFramework | Theme | Capacity | Document Control | Market Requirements

Bus. Req. Key: BR1-100924 Administrative handling of service contracts

MR Ref #	Market Description	Request Raiser
MR1-106228	Define contract terms at configuration / serialised item level.	Barco
MR1-106229	View Contract Information during Call and Work Order entry	Barco
MR1-106231	Change manually the calculated contract price	Barco
MR1-106232	Maintaining contracts	Barco
MR1-106448	contract template on serialized item level	Simac
MR1-106449	matrix contract pricing (model - template)	Simac
MR1-106450	invoice date of installments	Simac
MR1-106453	Ease of use in changing configuration	Simac
MR1-106454	Changes on an active contract	Simac
MR1-106455	Manually change contract prices	Simac
MR1-106510	Expired Warranty Coverage Terms continue to allow warranty func	Pier Aldershof

BR1 100924 623 of 643

Fig. 8. Baan requirements database screenshot.

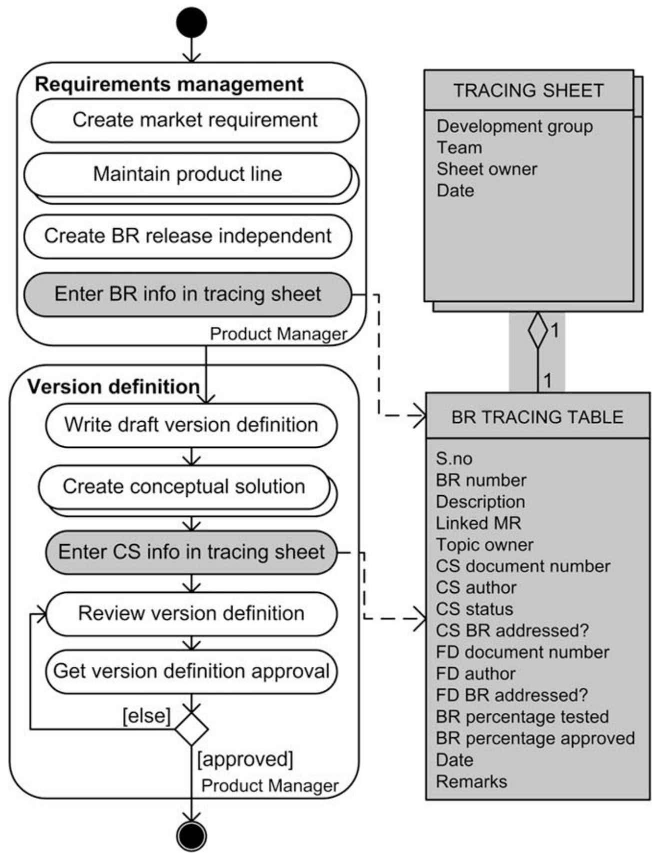


Fig. 9. Snapshot of increment 6.

stored in the voting list. During a customer voting process, the ones that receive the most votes are stored in the requirements database as business requirements, the ones that score average are saved for the next voting round, and the ones that receive almost no votes are deleted.

In Fig. 12, one extra activity is inserted, namely 'Carry out customer voting process'. Another activity is modified, namely from 'Create BR release independent' to 'Store selected MRs as BRs'. At the deliverable side, one extra concept is added, the VOTING SHEET. This concept has several new attributes and an aggregation relationship to MARKET REQUIREMENTS. Several attributes have been deleted, modified or inserted in the existing concepts. Finally, the relationship between MARKET REQUIREMENT and BUSINESS REQUIREMENT is changed from aggregation to generalization.

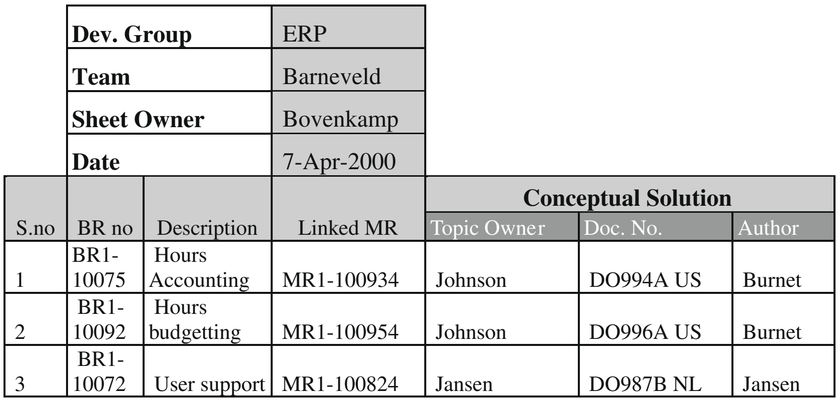
4. Analysis and lessons learned

Reflecting on the retrospective case study that we carried out at Baan, we identified several lessons learned. First, we give an overview of our method increment analysis. Then, we describe the lessons learned on incremental method evolution. Finally, we reflect on four lessons learned at Baan, concerning software product management in a global setting.

4.1. Method increments analysis

Looking at the modeled method increments, we can conclude that all increments found in the retrospective case study can be modeled with the 18 elementary method increment types that are listed in Section 2.2.

By counting the occurrences of elementary method increments in the series of analyzed method increments, we have constructed Table 2. Several things attract attention. Firstly, properties are most often inserted, modified or deleted. These kinds of method increments happen because of two reasons. First, it can be a side effect of another, more essential, change in the method. For example, in method increment 4, a release table is added. This leads to an inevitable change in the properties of the requirements concept, since this concept is now not part of the version definition anymore, but exists on its own, and is referred to in the release table. The other reason for an introduction, modification or change of attributes can be found in the experience of the users of the method. When a product manager or developer uses a certain template, it may become clear that one of the attributes is newly required or becomes



Dev. Group	ERP
Team	Barneveld
Sheet Owner	Bovenkamp
Date	7-Apr-2000

S.no	BR no	Description	Linked MR	Conceptual Solution		
				Topic Owner	Doc. No.	Author
1	BR1-10075	Hours Accounting	MR1-100934	Johnson	DO994A US	Burnet
2	BR1-10092	Hours budgetting	MR1-100954	Johnson	DO996A US	Burnet
3	BR1-10072	User support	MR1-100824	Jansen	DO987B NL	Jansen

Fig. 10. Part of the tracing sheet.

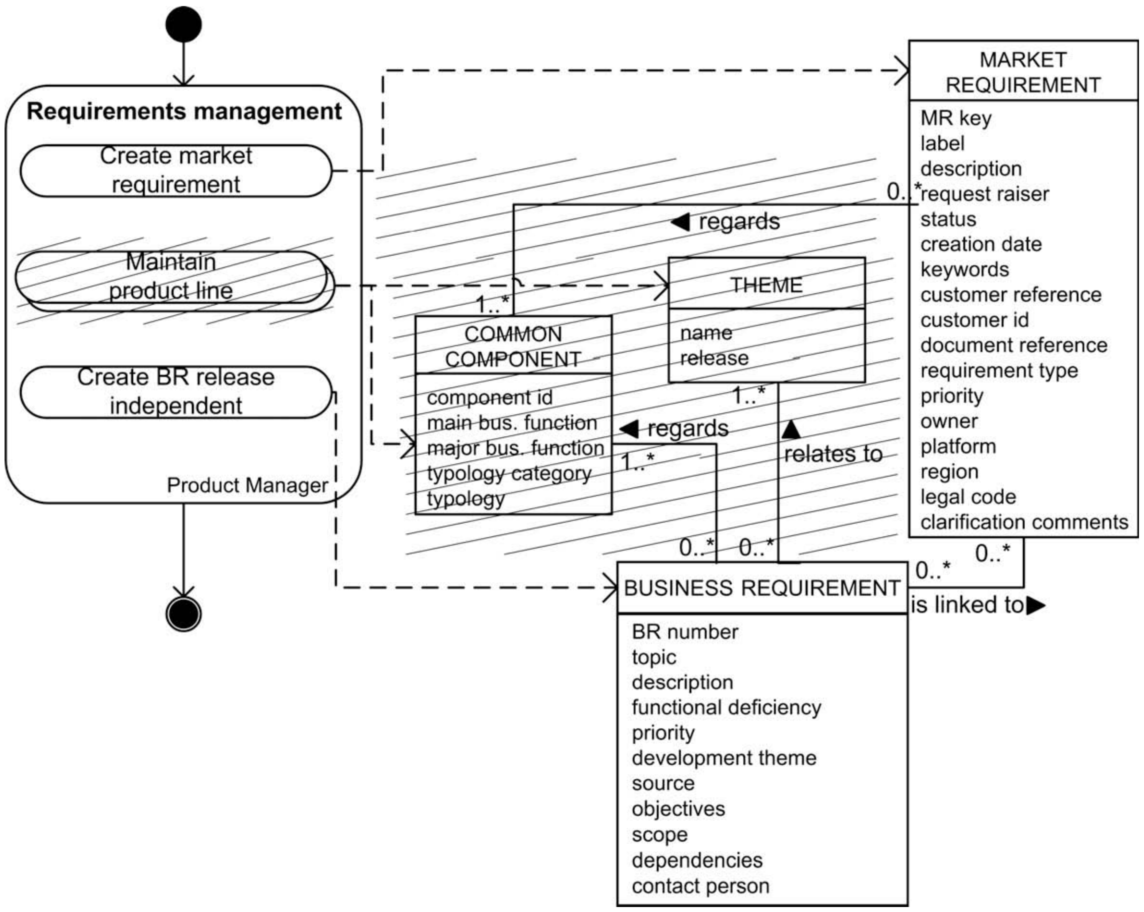


Fig. 11. Snapshot of increment 12.

redundant. These kinds of changes are easy to implement without changing the entire method rationale.

The second thing that attracts attention is that concepts and activity nodes are inserted regularly, but not often deleted. Consequently, the same holds for Relationships and Transitions. The reason for this is quite clear: during most of the years that are covered in the case study, Baan went through a large growth. Furthermore, during the period of downsizing after 2001, the method stays in place and is hardly scaled down in a similar proportion.

Role is the method fragment that changes the least. Two new roles were inserted: first the Program manager and later the product manager. These roles kept on existing, as well as the development role. Of course, the number of employees in these roles changed over time. At a certain moment 60 product managers were employed and at other times 10. However, the role of the product manager was never omitted.

4.2. Regarding incremental method evolution

Concerning incremental method evolution, we learned that increment drivers vary during evolution. The improvement of the requirements management processes related to the method increments were motivated by four different drivers:

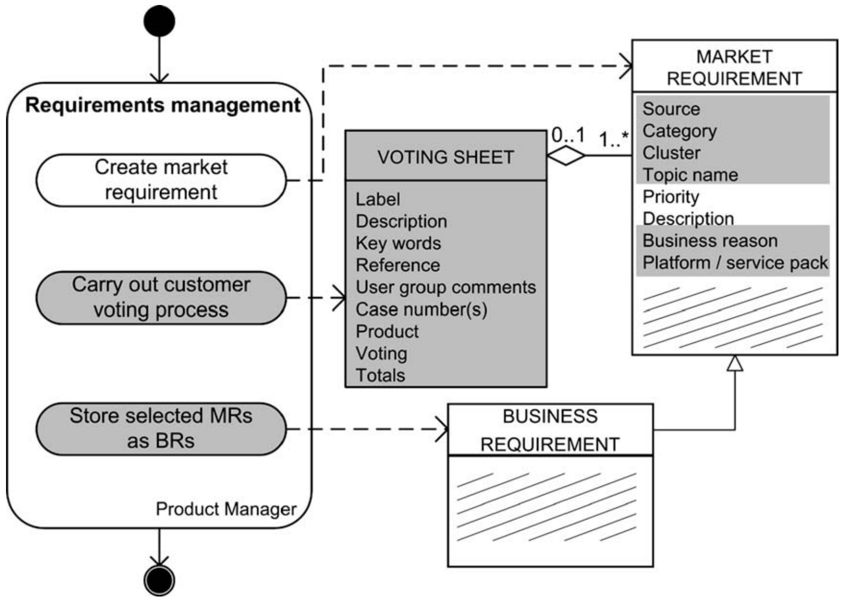


Fig. 12. Snapshot of increment 13.

Table 2
Overview of counted method increment types.

	Concept	Property	Relationship	Activity node	Transition	Role
Introduction	17	147	17	24	21	2
Modification	2	10	2	2	0	0
Deletion	3	38	5	2	1	0

1. *Development management*: The growth of the development effort requires that the role of the product manager is established, with expanding responsibilities for requirements management and release planning. As shown in Table 1, this driver caused five of the fourteen method increments.
2. *Business management*: The change of corporate business strategy requires process changes. The business management driver was responsible for four method increments.
3. *Departmental interfacing*: Other departments, such as marketing and customer services, require process adaptations for smooth interoperations and performance improvement. We found three method increments that were triggered by this driver.
4. *Certification*: The ambition to achieve higher maturity levels in CMMi requires the implementation of certain process extensions, although those processes may not strictly speaking be critical to the business. Two of the fourteen method increments were caused by the certification driver.

Usually a process change is due to a combination of drivers, but always one main motivation could be identified. Table 1 shows the variety in the main motivation for each method increment.

4.3. Regarding global Software Product Management

In the years of process improvement in requirements management, Baan gained a variety of experiences. From these, the following lessons learned were distilled, which can serve as guidance for other globally operating product software vendors.

4.3.1. Shared infrastructure is critical for rollout

The company-wide availability of and access to tools supporting the method is a critical success factor for the proper adoption of new method increments. In the Baan case these were the central requirements database, the development method on the intranet, and the document management systems for storing all development documents (cf. [4]).

4.3.2. Small increments facilitate gradual process improvement

Most organizations are not able to adopt large method increments in one step. Simple method adaptations such as for example the introduction of a new conceptual solution document (increment 5) or a new tracing sheet (increment 6), can easily be communicated to the product managers. Small increments usually require some simple instructions without a formal training program, and personal communication means (telephone, email) support the rollout to all teams worldwide.

4.3.3. Global involvement is critical

A centralistic process improvement effort will not be successful. Baan had a Software Engineering Process Group (SEPG) composed of representatives of all major development centers. The global rollout of method increments was facilitated by the local SEPG representatives. Furthermore, for some improvements a distributed project team was formed, the so-called Process Action Team (PAT). The PAT, consisting of experts in the improvement area, developed the method increment with templates, instructions and examples. In this way methodical content was accumulated with contributions from all global development locations. (cf. increments 3 and 4).

4.3.4. Acquired companies come later

Product software companies are known for their strong corporate cultures (cf. Google [30], Microsoft [8]). This also implies that the companies have their own process cultures. After acquisition it is important to support the existing culture and method for a while, and motivate the evolutionary adaptations toward a uniform method.

5. Related work

Several studies to evolution of methods and processes within the software domain have been carried out. To start with, Nejmeh and Riddle [18] argue that a company's context is the major influence on the definition and sequencing of process change cycles. Since all companies operate in different contexts, there is no "one size fits all" process evolution approach suitable for all companies. To cope with this, Nejmeh and Riddle developed a Process Evolution Dynamics Framework, which helps companies to cope with their process evolution in response to several drivers. As examples of these drivers, they mention customer desires, marketplace structure, personnel availability and capability, business goals, and available technology.

Ahn et al. [1] use a more generic structure to categorize increment drivers or, as they call it, evolution drivers. In their research, they propose a mechanism that supports the customization of software processes, based on Case Based Reasoning and a knowledge-based technique. The evolution drivers that they use are *improvement drivers*, such as complying with a certain CMM level, and *domain-specific drivers*, which can be any context driver, such as the use of a new technology.

The improvement drivers that Ahn et al. [1] identified are also used by Nguyen and Conradi [19]. They developed a categorization framework for process evolution which consists of six dimensions: where, why, what, when, how and by whom. The *why* dimension represents the major causes (or driver) and is classified in four categories: *correction*, *refinement*, *adjustment*, and *improvement*. In a case, study Nguyen and Conradi found that 35–40% of the recorded evolutions were requested or caused by the customer; 40% of recorded evolutions occurred because of resource underestimating; and 20% of the evolutions were a result of revising the plan document by changing initially planned activities.

Bandinelli et al. [2] use yet again another categorization mechanism. They divide evolution drivers within a software process into three categories of change: *incremental definition*, which is caused by the fact the it is impossible to define an entire software process model from the beginning; *changes in the environment or organization*, which may be caused by, for example, the introduction of new tools and changes in the strategy; *customization of the software process model*, which refers to the possibility of the process agents (e.g. the developers) to change the process when necessary.

Finally, some research has been conducted to method rationale and evolution. Rossi et al. [25] investigated the use of method rationale. They found that for organizations method rationale is a powerful mechanism in maintaining knowledge on systems development. By maintaining method rationale, i.e. preserving information on earlier method versions, changes and decisions, it can support future decision-making on the method; help method users to become familiar with the method; and make peculiarities and limitations of the currently used method more understandable. A drawback of this approach is that the true reasons of method changes are not always clear and sometimes controversial within the organization. Related to this is the research of Ocampo and Münch [21], who carried out an exploratory study to process evolution rationale. In this study they analyzed a database that was filled by process engineers with changes on software processes as well as justifications of the changes. They found a list of 11 explanations of changes. However, all explanations concerned procedural issues, such as ambiguous descriptions, non-compliant activities and improper sequences of activities.

In our study we found four main increment drivers: *development management*, *business management*, *departmental interfacing*, and *certification*. Several of these drivers are also mentioned in other studies. For example, Ahn et al. [1] mention improvement drivers for complying with a certain standard. This clearly resonates with our certification driver. However, many drivers do not directly map because we have a different focus. For example, Nejmeh and Riddle [18] describe customer desires as an increment driver. We do not describe these issues, because we only look at the Software Product Management function. Issues that are handled by other departments, such as marketing and sales, are categorized in the increment driver Departmental interfacing. In the same way, we can group e.g. Availability and Capability under the Development driver.

6. Conclusions

In this work, we carried out a research on incremental method evolution in a global software product management setting. We used a retrospective case study to analyze method increments. The results of this case study show that all increments found in the retrospective case study can be modeled with the 18 method increment types that we identified. However, in this case study, not all increments that we identified occurred.

In addition to the validation of the generic method increment types, we researched the general principles behind method increments. We found four increment drivers, in which we can place all method increments. These are development management, business management, departmental interfacing and certification.

6.1. Implications for practice

This retrospective case study shows the evolution of methods of what once was largest Dutch software vendor. We show how a company can cope with the its growth by changing its methods in small incremental steps. In addition, we described our lessons learned by reflecting on twelve years of implementing process improvements in the product management processes at Baan. Entrepreneurial opportunism in combination with changing economic situations requires product software companies to adapt their methods frequently. It is then critical to maintain up-to-date method documentation, including proper explanations of changes and notifications to the involved stakeholders.

6.2. Implications for research

Situational method engineering theory has focused mostly on constructing or adapting a method for a certain project. However, the *evolution* of methods that takes place in most companies is neglected. In this research, we show the mechanism of incremental situational method engineering. Each method increment is caused by an increment driver that arises from a changed company situation. By using method increments for software process improvement, companies can implement small, local changes in their processes. Accordingly, they are not forced to radical changes. Our experience is that by properly embedding these increments in the existing infrastructure and communicating them to the right knowledge workers, process improvements can be implemented evolutionary and successfully.

More empirical research is needed into the method execution problems and improvement needs in relationship to the size, structure and assembly of the method increment. Furthermore, the propagation of process changes to the corresponding transformation of deliverables and vice versa still requires adequate tool support.

6.3. Future research

Currently, we are working on the realization of the Product Software Knowledge Infrastructure [34,35]. With this infrastructure, product software companies can obtain a custom-made advice that helps them to improve their processes. The method increment types that we validated in this case study are used to implement assembly rules in the infrastructure. In addition, research is being done to the situational factors that influence the choice of software product management methods [3]. In the future, we plan to fill the method base with these situational factors, method fragments and assembly rules, and validate it at product software companies of different sizes and in different sectors.

References

[1] Y.W. Ahn, H.J. Ahn, S.J. Park, Knowledge and case-based reasoning for customization of software processes – a hybrid approach, International Journal of Software Engineering and Knowledge Engineering 13 (3) (2003) 293–312.

[2] S.C. Bandinelli, A. Fuggetta, C. Ghezzi, Software process model evolution in the SPADE environment, IEEE Transaction on Software Engineering 19 (12) (1993) 1128–1144.

[3] W. Bekkers, I. van de Weerd, S. Brinkkemper, A. Mahieu, The influence of situational factors in software product management: an empirical study, in: Proceedings of the 21st International Workshop on Product Management, Barcelona, Spain, 2008.

[4] S. Brinkkemper, Method engineering with web-enabled methods, in: S. Brinkkemper, E. Lindencrona, A. Sølvberg (Eds.), Information Systems Engineering: State of the Art and Research Themes, Springer Verlag, London, 2000, pp. 123–133.

[5] J.R. Bryson, Business service firms, service space and the management of change, Entrepreneurship and Regional Development 9 (2) (1997) 93–112.

[6] E. Ó Conchúir, H. Holmström Olsson, P.J. Ågerfalk, B. Fitzgerald, Benefits of global software development: exploring the unexplored, Software Process: Improvement and Practice 14 (4) (2009) 201–212.

[7] M.A. Cusumano, The Business of Software: What Every Manager, Programmer, and Entrepreneur Must Know to Thrive and Survive in Good Times and Bad, Free Press, 2004.

[8] M.A. Cusumano, R.W. Selby, Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People, The Free Press, New York, NY, USA, 1995.

[9] C. Debou, A. Kuntzmann-Combelles, Linking software process improvement to business strategies: experiences from industry, Software Process: Improvement and Practice 5 (1) (2000) 55–64.

[10] C. Ebert, The impacts of software product management, Journal of Systems and Software 80 (6) (2007) 850–861.

[11] K. el Emam, J.N. Drouin, W. Melo, SPICE: The Theory and Practice of Software Process Improvement and Capability Determination, IEEE Computer Society, Los Alamitos, CA, 1998.

[12] G. von Krogh, M.A. Cusumano, Three strategies for managing fast growth, Sloan Management Review 42 (2) (2001) 53–61.

[13] L. Krzanik, J. Simila, Is my software process improvement suitable for incremental deployment?, in: Proceedings of the 8th International Workshop on Software Technology and Engineering Practice (STEP '97), IEEE Computer Society, 1997, pp 76–87.

[14] M. Kwestel, M. Preston, G. Plaster, The Road to Success: How to Manage Growth, Wiley, 1998.

[15] A.S. Lee, R.L. Baskerville, Generalizing generalizability in information systems research, Information Systems Research 14 (3) (2003) 221–243.

[16] M.R. Leary, Hindsight distortion and the 1980 presidential election, Personality and Social Psychology 8 (2) (1982) 257–263.

[17] J. Natt och Dag, V. Gervasi, S. Brinkkemper, B. Regnell, A linguistic-engineering approach to large-scale requirements management, IEEE Software 22 (1) (2005) 32–39.

[18] B.A. Nejmeh, W.E. Riddle, A Framework for Coping with Process Evolution, LNCS, Springer, Berlin/Heidelberg, 2005, pp. 302–316.

[19] M.N. Nguyen, R. Conradi, Towards A Rigorous Approach for Managing Process Evolution, LNCS 1149, Springer, Berlin/Heidelberg, 1996. pp. 18–35.

[20] Object Management Group, UML 2.0 Superstructure Specification, 2004.

[21] A. Ocampo, J. Münch, Process Evolution Supported by Rationale: An Empirical Investigation of Process Changes, LNCS 3966, Springer, Berlin/Heidelberg, 2006. pp. 334–341.

[22] M.C. Paulk et al., The Capability Maturity Model: Guidelines for Improving the Software Process, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[23] A. Rainer, T. Hall, Key success factors for implementing software process improvement: a maturity-based analysis, Journal of Systems and Software 62 (2) (2002) 71–84.

[24] I. Richardson, K. Ryan, Software process improvements in a very small company, Software Quality Professional 3 (2) (2001) 23–35.

[25] M. Rossi, B. Ramesh, K. Lyytinen, J.-P. Tolvanen, Managing evolutionary method engineering by method rationale, Journal of Association of Information Systems 5 (9) (2004) 356–391.

[26] M. Saeki, Embedding metrics into information systems development methods: an application of method engineering technique, in: Proceedings of the 15th International Conference on Advanced Information Systems Engineering, LNCS 2681, Springer, Berlin/Heidelberg, 2003, pp. 374–389.

[27] Software Engineering Institute (SEI), Process Maturity Profile of the Software Community, Carnegie Mellon University, 2002.

[28] G. Stalk, P. Evans, L. Shulman, Competing on capabilities, in: D. Barnes (Ed.), Understanding Business: Processes, Routledge, NY, 2001, pp. 42–49.

[29] D. Stelzer, W. Mellis, Success factors of organizational change in software process improvement, Software Process Improvement and Practice 4 (4) (1998) 227–250.

[30] D.A. Vise, M. Malseed, The Google Story, Delacorte Press, New York, 2005.

[31] I. van de Weerd, S. Brinkkemper, R. Nieuwenhuis, J. Versendaal, L. Bijlsma, Towards a reference framework for software product management, in: Proceedings of the 14th International Requirements Engineering Conference, Minneapolis/St. Paul, Minnesota, USA, 2006, pp. 312–315.

[32] I. van de Weerd, S. Brinkkemper, J. Souer, J. Versendaal, A situational implementation method for web-based content management system-applications: method engineering and validation in practice, Software Process: Improvement and Practice 11 (5) (2006) 521–538.

[33] I. van de Weerd, S. Brinkkemper, J. Versendaal, Incremental method evolution in requirements management: a case study at Baan 1994–2006, Institute of Computing and Information Sciences, Utrecht University, Technical Report UU-CS-2006-057, 2006.

[34] I. van de Weerd, S. Brinkkemper, J. Versendaal, Concepts for incremental method evolution: empirical exploration and validation in requirements management, in: Proceedings of the 19th International Conference on Advanced Information Systems Engineering, LNCS 4495, Springer, Berlin/Heidelberg, 2007, pp. 469–484.

[35] I. van de Weerd, J. Versendaal, S. Brinkkemper, A product software knowledge infrastructure for situational capability maturation: vision and case studies in product management, in: Proceedings of the 12th Working Conference on Requirements Engineering: Foundation for Software Quality, Luxembourg, 2006, pp. 97–112.

[36] L. Xu, S. Brinkkemper, Concepts of product software, European Journal of Information Systems 16 (5) (2007) 531–541.

[37] R.K. Yin, Case Study Research: Design and Methods, Sage Publications Inc., 2003.