The supply chain of enterprise software
Postmus, D.

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2009

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):
7.1 Main findings

The main purpose of this thesis was to gain greater insight into some of the organizational and managerial aspects of the supply chain of enterprise software. We were especially interested in exploring whether it would be possible to apply some of the existing descriptive concepts from the analysis and design of the supply chain of physical goods to the supply chain of enterprise software or whether other constructs would be needed for describing and analyzing these kinds of networks. In particular, the following research objectives were identified in the introductory chapter of the thesis:

RO1 To develop a framework for the analysis and design of the supply chain of enterprise software.

RO2 To use the framework to (a) explore how the specific characteristics of the software industry enable and/or constrain the supply chain management initiatives of software companies, and (b) to develop quantitative models for assisting software companies in making appropriate design decisions.

Our main findings with regard to these objectives are discussed in the sub-sections below.

7.1.1 RO1: A framework for the analysis and design of the supply chain of enterprise software

By adapting some of the existing concepts from the field of operations management to the specific characteristics of the software industry, Chapter 2 and the first parts of Chapters 3 and 5 contributed to the development of a theoretical framework for the analysis and design of the supply chain of enterprise software. In particular, the following three elements were considered:

1. the product delivery strategy of the software company,
2. the structure of the software company’s internal supply chain, and

3. the actors involved in the software company’s supply network and distribution channel.

The product delivery strategy of the software company

To contribute to a better and systematic understanding of the different positions software companies can take on the continuum ranging from low flexibility and cost (standard software) to high flexibility and cost (bespoke software), two existing concepts from the design and manufacturing of physical goods were adapted to the new field of enterprise application software: the customer-order decoupling point (particularly its engineering design perspective) and the degree of customer specification freedom. The position of the customer-order decoupling point determines a software company’s investment in software assets prior to receiving a customer order. It was used as a primary characteristic to divide the product delivery strategies of software companies into three generic categories:

- engineer-to-order, i.e. the software development process is completely customer-order-driven,
- compose-to-order, i.e. customer-specific systems are composed from standardized components, and
- engineer-to-stock, i.e. the software development process is completely forecast-driven.

The degree of customer specification freedom was used as a secondary characteristic to split these three generic strategies into zero or more subtypes, depending on the types of customization that a customer is allowed to make. In particular, the following three dimensions of customer specification freedom were considered:

- accessing the system’s functionality through an application programming interface or web service,
- configuring the behavior of the software system (either by setting parameters or through the use of extension points), and
- adapting the behavior of the system.
7.1. Main findings

The structure of the software company’s internal supply chain

To provide concepts and tools for analyzing the structure of a software company’s internal supply chain, an analogy was made between the role of inventory in the production of physical goods and the role of reusable software assets in the development of software. In particular, it was argued that analogue to stock points in the production of physical goods, libraries of reusable software assets add flexibility to the software development process: a reusable software asset can be shared across several different software systems, provided that the asset has sufficient variation points to adjust it to the specific requirements of these systems. Systematic application of software reuse therefore allows software companies to divide their development processes into two or more separately controllable phases. The production of reusable software assets, often referred to as development for reuse, takes place in the first development phase, whereas concrete software systems are produced in the final development phase, also referred to as development with reuse. Intermediate phases exist when software reuse is applied recursively, i.e. when software systems are obtained by reusing and extending one or more existing software assets that have been developed with reuse as well.

The actors involved in the software company’s supply network and distribution channel

It was argued that the license agreement between the software company and the end user is a suitable criterion for the identification of a software system because it contains a description of the components that are considered to be part of the system. Based on this criterion, the following rule for demarcating the supply network of a software company was derived: a component supplier is a primary member of the supply network if the use of this component is covered by the license agreement between the software company and the end user. As far as the distribution channel of a software company is concerned, a distinction was made between resellers, which are mainly responsible for sales and implementation of the system at the customer, and independent software vendors, which, depending on the degree of customer specification freedom that is supported by the software company, develop interfaces with other software packages and develop add-ons for specific industries.
Chapter 7. General discussion

7.1.2 RO2: Application of the theoretical framework

The theoretical framework that we briefly summarized in the previous section provides researchers with a sound basis for exploring how organizations in the software industry organize and manage their supply chains. It may also serve as a starting point for the development of normative methods and instruments for assisting software companies in making appropriate design decisions. In the second parts of Chapters 3 and 5, we used our theoretical framework for the following two purposes:

1. to explore how software companies integrate quality management and requirements management across the supply chain (RO2a), and

2. to develop a quantitative model for assessing whether it is economically feasible to apply software reuse recursively (RO2b).

How do software companies integrate quality management and requirements management across the supply chain

In order to explore how software companies integrate quality management and requirements management across the supply chain, we conducted semi-structured interviews at two different companies in the enterprise resource planning (ERP) domain. One of them was a producer of ERP software, the other company was a reseller of enterprise information systems, including ERP software. The main findings of the case study are briefly summarized in the paragraphs below.

Quality management

Some companies in the distribution channel of an ERP manufacturer are specialized in developing industry-specific extensions. Since the quality of a customer configuration is partly determined by the quality of these extensions, both ERP manufacturers have started a certification program that forces these independent software vendors to implement the same quality standards as the ERP manufacturer. In addition, to prevent these extensions from breaking down after an update of the ERP system, one of the ERP manufacturers guarantees a forward compatibility of extensions that are developed with its software development kit. The other ERP manufacturer, in contrast, relies heavily on its resellers to detect and resolve compatibility problems before the update is deployed at the customer’s site.
7.1. Main findings

Requirements management
The ERP manufacturer from the first case study organizes monthly meetings with its database supplier to discuss the features that are going to be incorporated in the next release of the database management system. It has also created a web portal where resellers can exchange customer requirements. In the second case study, the reseller and the independent software vendor have integrated the requirements management process by exploiting customer-specific solutions as domain-specific add-ons when it turns out that there is a larger market for these solutions.

A quantitative model for assessing whether it is economically feasible to apply software reuse recursively
For manufacturers of physical goods, it holds that the number of phases in the production process is an important design decision. The same is true for the number of phases in the software development process: although software reuse has the potential of significantly reducing development time and costs, it does not always result in cost reduction because variations among related applications have to be accounted for. To assist software companies in determining the optimal number of software development phases, we quantified the costs that are associated with three different mechanisms for achieving software reuse: composition, black-box variation, and white-box variation. By using the resulting modeling elements, a quantitative model was constructed that can assist software companies in assessing whether it is economically feasible to apply software reuse recursively.

7.1.3 Additional material on related research topics
The main findings with respect to our research objectives were outlined in the previous two subsections. We conclude this overview by providing a brief summary of the results obtained in Chapters 4 and 6, which contain additional material on related research topics.

Chapter 4: Pay-per-use versus fixed-fee licensing
In Chapter 4, we compared two strategies for the pricing of packaged software: fixed-fee and pay-per-use licensing. It was assumed that the market consisted of a monopoly software vendor that was selling packaged software to customers who were homogeneous in marginal value of software use but
heterogeneous in level of use. In addition to obtaining the software package from the market, customers could develop the required software in-house. When in-house development costs are constant across customers, our results showed that the software vendor prefers pay-per-use licensing over fixed-fee licensing if in-house development is relatively expensive, whereas fixed-fee licensing is optimal if the cost of in-house development drops below a certain threshold value. When the assumption of a constant in-house development cost was relaxed by letting it vary among customers, it still holds that pay-per-use licensing is optimal if its average is relatively large. For low and medium values of the average cost of in-house development, however, fixed-fee licensing may no longer be optimal as the relative attractiveness of the two licensing strategies now depended on how dispersed the in-house development costs of individual customers were.

Chapter 6: A business perspective on web service composition

When the current trend towards delivering software functionality as a service over the Internet continues to develop, significant changes in the industrial organization of the software industry can be expected as traditional channel intermediaries such as resellers are no longer required. Instead, proponents of the software as a service paradigm envision a supply chain structure in which service aggregators combine the services from different service providers to create composite services that are capable of performing higher-level business transactions. In order to assist potential service aggregators in deciding whether or not to enter the market, Chapter 6 developed a quantitative model for estimating and predicting whether the accumulated benefits from offering a composite service are sufficiently large to compensate for the total cost of developing and operating the service. Our approach was to first develop a single-period version of the model, which was subsequently extended to include a multi-period decision horizon, keeping the basic terms and definitions unchanged. An example illustrated how the model can assist potential service aggregators in deciding whether or not to enter the market.

7.2 Reflection and suggestions for future research

In this section, a critical reflection on some of the methodological issues of this thesis will be undertaken. The opportunities for future research that follow from this reflection will be outlined as well.
7.2. Reflection and suggestions for future research

The typology of product delivery strategies in the software industry

The extent to which our typology of product delivery strategies in the software industry is “useful in aiding analysis in some way” can be assessed by considering the following three criteria (Gregor, 2006): (i) whether the dimensions are clearly defined, (ii) whether the dimensions are meaningful and natural, and (iii) whether important dimensions are not missing. We can say that our typology scores well on the first criterion as its two dimensions (i.e. the position of the customer-order decoupling point and the degree of customer specification freedom) have been extensively described in the second chapter of this thesis. The performance of our typology on the other two criteria is more difficult to assess: although we have tried to give credence to it by applying it to some examples, further research is required to empirically validate and test our classification framework. The knowledge that will be gained from applying our typology to several real-life case studies may lead to some refinements of the typology’s existing dimensions or may even result in the addition of one or more new dimensions.

The structure of the software company’s internal supply chain

In Section 1.2.2, we argued that when looking at supply chain structure, a distinction can be made between actors and activities. As far as the structure of a software company’s internal supply chain is concerned, we have focused on this latter aspect by making an analogy between the role of inventory in the production of physical goods and the role of reusable software assets in the development of software. In doing so, we were mainly concerned with determining the optimal number of software development phases. The phases themselves were treated as black-boxes: we were only interested in their inputs and outputs, not in their transforming resources (i.e. staff and equipment) and the assignment of tasks to these resources. This does not mean that task assignment is not important to consider. On the contrary: we expect that the way in which the different development phases are allocated to a company’s business units, and the related decision of where to locate these business units (e.g. on-shore or off-shore), may seriously affect the company’s overall performance, making it an interesting issue to explore in future research.
Using the license agreement as criterion for the identification of a software product

We have argued that a suitable criterion for the identification of a software product can be found in its legal definition. One should keep in mind, however, that the license agreement only covers the use of a specific version of a software product. In the course of time, a series of upgrades are produced to adapt this product to changing requirements. Although existing customers often require a separate maintenance agreement to receive these upgrades, they do become part of the basic license agreement for customers of the new version.

Upgrades are used to preserve the existing functionality and performance of a software system through bug fixes based on error reports from customers. They also serve to increase the system’s functionality and performance by adding new features and implementing performance upgrades (Wiederhold, 2006). The addition of new features causes the system to grow in size throughout its lifecycle. This may also result in changes to the supply chain structure, especially when part of the new code is provided by external suppliers. Using software licenses to identify a software product unambiguously therefore implies that the corresponding supply chain may evolve over time.

Using case study research to identify some of the supply chain management initiatives of software companies

To identify some of the supply chain management initiatives of software companies, we conducted semi-structured interviews at two companies from the domain of ERP software. As both companies are members of different supply chains, our case study can be seen as having an embedded multiple-case design (Yin, 2003): the cases are the two supply chains to which the companies belong; the embedded units are the different members of these supply chains.

For both cases, it holds that we collected data from just one of the embedded units (i.e. we conducted interviews at only one supply chain member). Although this gave us some insight into the different modes of supply chain coordination that were applied in these two cases, different supply chain members may have different perceptions on how activities are integrated and managed across the supply chain. Future research should therefore consider reducing the potential bias that results from analyzing the sup-
ply chain from only one perspective. A possible direction that could be taken is to start by conducting interviews at the software manufacturer and then to cross-validate this data by conducting interviews at several of the manufacturer’s first-tier suppliers and customers.

7.3 Concluding remarks

Compared to industries that are based on physical goods, the software industry in general and the domain of enterprise software in particular has several unique properties. These properties make it interesting to explore whether the established theory of supply chain analysis and design can be applied and adapted to the supply chain of enterprise software or whether other concepts and tools are needed for studying these kind of networks. The experiences gained from this thesis suggest that existing descriptive supply chain theory can indeed be fruitfully applied to describe and analyze the development, sales, and delivery of enterprise software. The existing prescriptive supply chain theory, in contrast, seems to be less applicable. This especially holds for methods and instruments that aim at reducing the unit cost of production as the marginal cost of producing a copy of a software system are already negligible.