Relationship Intimacy in Software Ecosystems: A Survey of the Dutch Software Industry

Joey van Angeren
Department of Information and Computing Sciences, Utrecht University
j.vanangeren@cs.uu.nl

Vincent Blijleven
Department of Information and Computing Sciences, Utrecht University
vbblijle@cs.uu.nl

Slinger Jansen
Department of Information and Computing Sciences, Utrecht University
s.jansen@cs.uu.nl

ABSTRACT

Software vendors depend on suppliers to provide the underlying technology for domain specific solutions. As a consequence, software vendors cooperate with suppliers to deliver a product. This cooperation results in supplier dependence, but also leads to opportunities. We present the results of an exploratory research based on twenty-seven case studies, identifying supplier strategies and resulting trade-offs. Strategies range from fully depending on large software ecosystem orchestrators to a minimal dependency strategy. Furthermore, we identify factors at play when selecting suppliers for different components. These factors include; ecosystem health indicators, product and license type and intensive support and maintenance flows. The results presented in this paper can be used by software vendors to assess their software supply network to review supplier relationships, but also for future research.

Categories and Subject Descriptors

I.6.5 [**Simulation and Modeling**]: Model Development; K.1 [**The Computer Industry**]

General Terms

Theory, Design, Management

Keywords

software ecosystem, supplier relationship, supplier selection, software supply network, product deployment context

1. INTRODUCTION

The Dutch product software industry is flourishing and is playing an important role in the Dutch economy. Various examples of successful products are computer games, navigation systems, administrative software and enterprise resource planning products. For the purpose of this paper, product software is defined as; "a packaged configuration of software components or a software-based service, with auxiliary materials, which is released for and traded in a specific market" [16].

As noted from the definition, software products are a configuration of numerous components. Often, a software vendor will not develop all of these components in-house, rather there will be a number of other organizations that supply them with hardware and software components, services and intellectual property vital for the products they offer. Because of this, software vendors become dependent on service providers and other software vendors in order to leverage their products to the customer. We refer to the network of actors that is the result of this phenomenon as a software ecosystem. A software ecosystem is defined as; "a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them" [10].

Software ecosystems can be studied from different scope levels. These different scopes involve different levels of abstraction and view software ecosystems and their interactions from different perspectives [8]. In the most detailed scope level, one specific software ecosystem is studied including the actors that are part of this ecosystem as well as the relationships amongst these actors. On the total opposite, a scope level addresses the organizational perspective and the relationships between different organizations and ecosystems. In this paper, we will use the middle scope level, in the form of Software Supply Networks, or in short SSNs. Within this scope level the network of hardware, software and service organizations is studied that cooperate to satisfy market demands [6]. This SSN can be used to provide insight into first-tier buyer-supplier relationships for one specific software vendor of interest, including the resulting flows and dependencies. Also, creating a Product Deployment Context (PDC) for this software vendor can provide further insights. A PDC describes the structure of software products and its direct running environment in a stack view, providing insight into how the product is composed out of different components [1] Combining these insights into how important a certain component is perceived to be for the portfolio of a certain product, including the grade of dependency on a certain supplier can be used to identify weaknesses. This can be useful to gain insight into the factors that are at play when selecting a supplier. Furthermore, this is valuable information to be aware of for a software vendor, to make the right decisions on a both strategic business as well as a software architectural level.

Because of the potential impact supplier dependencies and

MEDES'11 November 21-24, 2011, San Francisco, USA

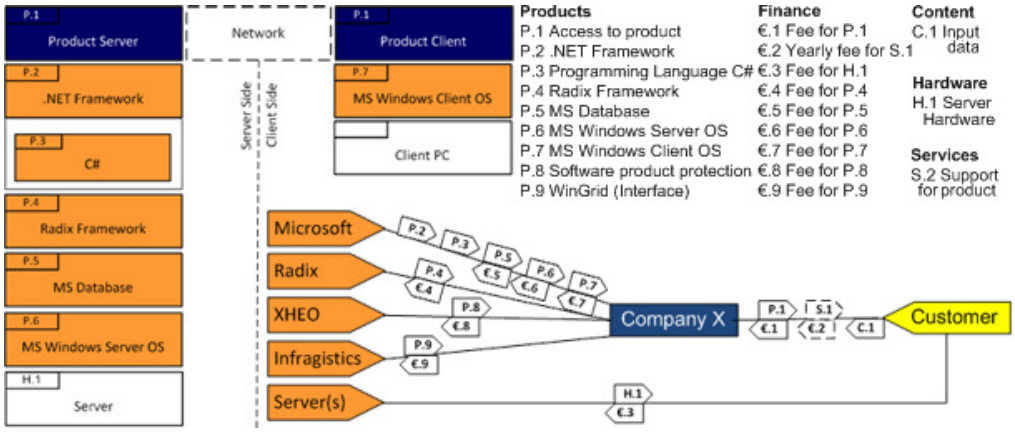

Figure 1: An example of a Product Deployment Context and Software Supply Network for a software vendor in the dataset

resulting trade-offs can have, more research needs to be carried out within this domain. In this paper we address supplier relationships from a portfolio perspective. We will propose a matrix to classify components and services and their perceived grade of importance for the final delivered product. This matrix will be utilized to perform a pattern analysis on the SSNs, PDCs and tables that describe the grade of intimacy of relationships with suppliers of Dutch product software companies. These data have been gathered through twenty-seven case studies. We will examine different supplier relationships, strategies and dependencies, based on the importance of the delivered component or service for the product leveraged by the Dutch software vendor. Furthermore, we will address the choices for supplier strategies from a software ecosystems perspective by addressing reasons to join an ecosystem as a customer to obtain software components or services, or even as a partner. Choosing for a certain supplier might result in a high level of dependence or lock-ins, having large implications for, for example, the business model of a company. Also, changing strategies or decisions about components or services by one of the suppliers can cause big implications and problems for the software vendor. On the contrary, intimate relationships with suppliers can result in advantages, such as shortened support and maintenance lines or secondary benefits. As a result, a software product company gets confronted with trade-offs.

The remainder of this paper continues with a description of the research approach in section two, in which we will elaborate on the research methods we employed as well as the case study participants selection process. In section three, we will present a matrix to classify components of a software product or its direct running environment. In the fourth section, we present the results of the case study concerning organizational size, business models and software ecosystems. An analysis and interpretation of the results presented in section four will be discussed in section five, including several benchmarks with similar studies that have been conducted in the past. In section six, we discuss encountered validity threats and make statements about generalization possibilities of the presented results. In the last section we draw the most important conclusions and provide suggestions for future research.

2. RESEARCH METHOD

In this paper we use empirical data gathered from the Dutch product software industry. We chose for a multiple case study design [17]. The data were gathered between September and November 2010. The total dataset consists of a total of twenty-seven case studies.

2.1 Data Collection

The data collecting process took place during the Product Software course at Utrecht University, which is part of the bachelor in Information Science curriculum. Twenty-seven couples of bachelor students selected a small to medium-sized Dutch product software company, where they wanted to conduct their research in order to get familiar with the Dutch product software industry. The main prerequisite for a software vendor to qualify for participation in these case studies was that their number of employees had to be at least ten. Furthermore, they had to be registered at the Dutch Chamber of Commerce. During two or three meetings, the teams of students gathered information about three key themes; organizational structure, business models and software ecosystems. Each assignment addressed one key theme, and each assignment was graded separately. The assignments did mainly consist of open question, this provides for a broader insight into drivers and motivations software vendors have for strategic decisions. All couples conducting the research used a similar approach when conducting the case studies.

The first part of the data collecting process was edged on gathering information about the participating software vendors. Information was gathered about the products they offer and the way in which the company is organized. This structure has been captured in an organizational structure, including details on the number of employees. The middle part of the case study was edged on business and revenue models. Data was captured by filling the business model canvas as defined by Osterwalder [12]. In this business model canvas, many components of the business model are addressed, for example, key partners, activities and revenue models. Furthermore, accompanied by a representative of the company, they filled in a SWOT Matrix [15] to identify the main strengths, weaknesses, opportunities and threats for the respective company.

Software ecosystems were addressed in the last part. For each of the companies, one product of interest, that the software vendor develops and delivers has been selected. For this product, the Software Supply Network and Product Deployment Context has been described and captured, according to the modeling techniques described by Boucharas, Jansen & Brinkkemper [1] and those described by Brinkkemper, Jansen & Van Soest [2]. An example from the dataset is shown in figure 1. The PDC describes a client-server product delivered by one of the product software companies that took part in this case study. The labels in the SSN, depicted at the right side of the figure match with those included in the PDC. The SSN also includes other suppliers of components, services and content. This way, the SSN gives an overview of first-tier buyer-supplier relationships and the exchange of products, services, data and money flows between these actors. Furthermore, in this part the perceived level of intimacy for each of the actors within the SSN has been collected in a table.

2.2 Selection Criteria

Because of the nature of the data collection process and to enhance the quality and integrity of the dataset, we formulated a number of inclusion criteria for each of the contributions. The contributions that were to be included into the dataset we used had to apply to the following criteria: (1) All three assignments have to be handed in and need to be accessible; (2) The average grade for the entire contribution has to be at least 7,5 on a scale of 1-10 where 1 is the lowest possible grade and 10 is the highest possible grade; (3) The grade for each assignment has to be at least 7 or higher; (4) Each assignment has to be entirely executed and complete; (5) Each company can be included only once into the dataset, and in case of a duplicate the one with the highest average grade will be included.

After applying the inclusion criteria to the initial dataset, in total seventeen out of twenty-seven contributions were included into the final dataset, 63% of the initial dataset. This dataset will be subject to the analyses that form the basis of the findings presented in this paper.

2.3 Data Analysis

The main research question of this paper is as follows; *"How does the perceived level of importance of a component, that is part of a software product, influence supplier selection"?* To be able to answer this research question and to benchmark with previous findings, we perform both qualitative and quantitative analyses on the empirical data gathered from the Dutch product software industry. Furthermore, notions from existing literature and about the Product Deployment Context will lead to the creation of a matrix that classifies product components.

A brief quantitative analysis is performed and employed to contextualize the dataset. We therefore elaborate on various organizational characteristics, such as organizational size and delivery models. Using these findings as a prerequisite, a qualitative analysis is performed to provide an answer to the research question. The SSN and PDC, together with the table that describes the perceived level of intimacy with each of the actors within these networks, will be subject to a pattern analysis. Through this analysis, we identify multiple supplier strategies and their resulting trade-offs. The matrix that will be presented in section 3 is then utilized for further analysis. The analysis is employed to identify factors that influence supplier selection and to determine to what extent these factors are perceived as important when selecting certain types of components.

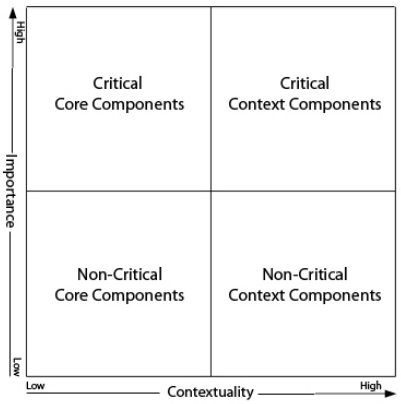

Figure 2: A matrix for the classification of components within a product deployment context

3. RELATED LITERATURE

Companies need to monitor their relationships with suppliers and be aware of the influence these suppliers can exert over them. Maloni & Benton [11] emphasize the need for power awareness within supply chains in order to construct integrated, high performance buyer-supplier relationships. When looking at these buyer-supplier relationships, Software Supply Networks differ from traditional supply networks because traditional supply networks, and associated supply chain management, lack attention for maintenance flows between different actors [9], This flow is important within the product software industry. Therefore, in this paper Software Supply Networks are subject of study.

Within the research domain of software ecosystems, Jansen, Finkelstein & Brinkkemper [10] summarized an array of research challenges. One of these challenges is to direct more attention to Software Supply Networks. However, until now little research has been directed at supplier relationships within these software supply networks. Popp & Meyer [13] identifies different categories of suppliers, based on the type of software or services they deliver and the exchange of goods, intellectual property and money that takes place between buyers and suppliers. Examples of these roles are an OEM or open source supplier. Jansen, Brinkkemper & Finkelstein [7] already noted that the intimacy of these supplier relationships is related to the role a certain component fulfills within a product context. However, they did not classify these components to address relationship intimacy for different component type suppliers.

4. A MATRIX FOR THE CLASSIFICATION OF COMPONENTS WITHIN A PRODUCT DEPLOYMENT CONTEXT

Product software is constructed out of multiple components. Most relevant components are either hardware or software components. Apart from that, additional services and the inclusion of added value by the software vendor will

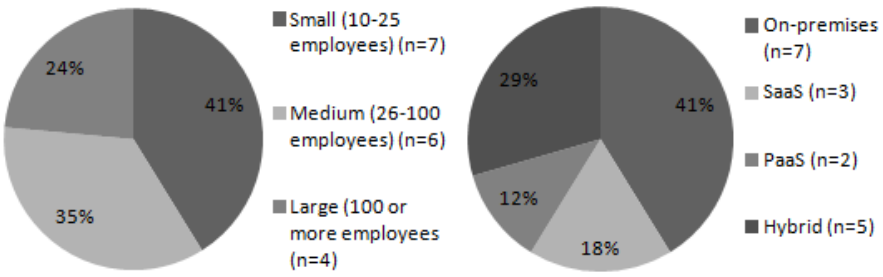

Figure 3: Contextualization of the dataset in terms of company size and delivery models

result in a final software product. As already elaborated on by Jansen, Brinkkemper and Finkelstein [7], some components obtained from suppliers are more easily replaced than others. An interface grid, for example, can be replaced easily within most software products for a product with similar functionality from another supplier without heavily affecting the end product. Totally migrating a software product to be compatible with another operating system on the other hand, is a challenging and time consuming process. Based on this notion, we developed a matrix that classifies components that are at play within the running environment of a software product, as described in PDCs.

The matrix in figure 2 distinguishes four types of components from an architectural perspective, that are part of the software product or its direct running environment. First, we distinguish between core and context components. When speaking of core components, we refer to the fundamental building blocks of a software product that are vital to allow the software product to be run and provide no value-added functionalities to the customer. Core software components are regarded as the heart of a software product. Context software components then, are software components that add specific values to the product. For example, certain functionalities that make the product unique to a customer or that add additional functionality to the product. Unlike core software components, context components are not a necessity for the software product to be run.

When examining core and context components, we can make a distinction between critical and non-critical core and context components. Components are critical if they cannot be easily interchanged with another component and contribute significantly to the added value of the software product and its resulting overall functionality. A simple PDF plug-in, for example, provides an additional functionality for the product and is therefore a context component, but since it is easily interchangeable by a substitute with equal functionality from a different supplier it is considered as non-critical.

The proposed matrix complies with the PDC modeling approach defined by Boucharas, Jansen & Brinkkemper [1]. Within the stack view of a product and its direct running environment, a distinction can be made between optional and required components. Typical examples of optional components again are plug-ins or components that are easily interchangeable with another component. Furthermore, the place a product has within the stack view depicted in a PDC, typically products that are lower on a stack are core components (e.g. operating systems, frameworks, databases), as well as the description that always accompanies a PDC provide for a last distinction within the four identified categories.

Because the PDC provides insight into product components and the running environment in a stack view that is based on a certain level of abstraction, this matrix is created from a simplified architectural perspective. Since factors like the degree of coupling and the lines of code necessary to incorporate a certain component into the final product cannot be measured by assessing the PDC, they are not considered as factors influencing the classification of components. Since we are dealing with PDCs within our dataset, this simplification step will not have an influence on the results.

5. RESULTS

To conceptualize the dataset, we divided the participants in three distinct company size categories. Companies having ten up to and including twenty-five employees are categorized as small. In addition, companies with an employee amount of twenty-six up to and including one hundred have been categorized as medium-sized companies. The last category is for companies categorized as large, who have more than one hundred employees. This categorization of participants based on company size differs from the one proposed by the Dutch Chamber of Commerce, because that categorization does not provide enough detail for this research. An overview of the result of this company categorization is displayed in figure 3.

Furthermore, the software delivery model for each company within the dataset is relevant. 55% of the companies sell their software products as traditional (on-premises) software, where the product is both installed and running on computers of the person or organization using the software. In addition, 41% of these companies also offer a software as a service (SaaS) solution as an alternative next to their traditional solution, we refer to this as a hybrid delivery model. Additional revenue streams for on-premises software will often come from subscriptions to maintenance and support. Furthermore, three participating companies are dedicated SaaS solution providers. The remaining two companies solely offer a platform as a service (PaaS) solution. An overview of solution types offered in terms of percentage can be seen in figure 3.

The core of the dataset is formulated around Software Supply Networks, Product Deployment Contexts and a table that describes the perceived level of intimacy with each supplier that is part of an SSN. Furthermore, a textual description accompanies these artifacts to provide insight into the motivation for the decisions made by the product software company. Out of this data, we distinguish four main

supplier strategies.

- Product integration with a hardware component supplier
- Depending on a large software ecosystem orchestrator
- The inclusion of open source components
- Minimal supplier dependence strategy

Software vendors are confronted with trade-offs when decisions are made to opt for one or more of these strategies. Choosing to be less dependent, for example, can result in having to dedicate more resources to development or having fewer benefits. An overview of the strategies and its resulting trade-offs are collected in table 1.

Several companies choose to integrate their products with a hardware component supplier. For example, a large company indicated that due to the complex implementation process of its product, the choice was made for a firm integration with a single hardware component supplier. This to streamline the implementation process and thereby reducing complexity. Companies need to make a trade-off between having a streamlined integration process by working with a hardware supplier with whom they have built up an intimate relationship and thus becoming dependent on this vendor, or having a less streamlined product integration process without being dependent on the hardware supplier, since in a lot of cases it is easily replaceable by another one.

Some companies choose to become fully dependent on large software ecosystem orchestrators. In a lot of cases this is regarded as an opportunity rather than a threat. Various reasons for this choice have been given, such as; *"continuation of the product and support is more or less guaranteed, since we anticipate the company will still exist in ten years from now"* or *"the company offers attractive benefits to its customers"*. One of the main reasons companies choose to rely on large software ecosystem orchestrators, is to benefit from niche creation within the ecosystem. In case of heavily depending on a large software ecosystem as a supplier, it becomes common to join its partnership model. This model brings additional benefits for both parties, can provide education for employees of the participant and can shorten direct support and maintenance lines to the supplier. Another trade-off needs to be made here, whether to rely on a large software ecosystem orchestrator and benefit from this relationship, or to remain as independent as possible.

Software vendors indicate to be reviewing their current supplier relationships because of the advent of open source software components as alternatives for proprietary components. Various reasons for this were given, such as; *"we advocate the inclusion of open source software components as we do not want to be subjected to vendor lock-ins"*, *"we contribute to open source software ourselves, to influence the way the open source project is heading"*. With the benefits from open source also come new supplier selection challenges, such as carefully examining licenses of open source software to avoid liability issues. One company stated; *"we do not want to include open source components in our product, as this makes us prone to liability issues for which we do not want to be held accountable"*. Having to leverage continuous support and maintenance to customers is perceived as an additional drawback of open source. A software vendor indicated; *"we associate the inclusion of open source components with increased responsibility in terms of maintenance and support. This responsibility now ends up at our company"*. Most open source communities just deliver their products rather than providing additional services. As a consequence, more responsibilities end up with the software vendor.

The larger companies within the dataset strive for less supplier dependence. Critical components for the leveraged product are developed in-house if resources are available, as one large company indicated; *"we try to develop as many components in-house as we can, to reduce the dependency on component suppliers"*. A trade-off is made between the advantage of not needing any additional resources because of being dependent on a supplier, or developing components in-house and thus decreasing direct supplier dependency. This in the contrary to smaller companies that rely on basic vital components from suppliers. They create some additional value on top of this, in an attempt to serve a niche. Apart from the mentioned suppliers, some software vendors are dealing with service providers and intellectual property providers. An example of this is that most SaaS vendors indicate to maintain intimate relationships with their hosting providers. They cannot permit themselves to have downtime or security breaches, so short support and maintenance lines are needed.

6. ANALYSIS

In the results we distinguished four types of supplier strategies and their trade-offs, but did not yet discuss the factors that are at play when selecting strategies for certain components. The component classification matrix serves as a basis for the analysis of the results. The goal of this analysis is to capture the influence of the type of a software component, as defined in the matrix, on supplier selection criteria and supplier relationships.

With regard to the **perceived level of intimacy**, a relationship with a component supplier can be classified as either intimate, familiar or unfamiliar. In this sense an intimate relationship can be, for example a partnership with a supplier, but also just having regular and direct contact with the supplier. On the contrary, an unfamiliar relationship with a supplier is applicable for normal buyer-supplier relationships in which the software vendor just obtains a product or service. As noted in section 4, software vendors indicate to select suppliers based on their capability to supply their components without disruptions, their company size or reputation. Furthermore, they strive for participation in partnership models of large software ecosystems. They are keen on intimate relationships with these organizations, especially when it concerns a critical core component or a critical context component supplier. An example of this is the relationship with a platform host that forms the basis of the end product.

Indicated motives for supplier selection, for example, when choosing to depend on a large software ecosystem show parallels with ecosystem health indicators. Ecosystem health is defined by Iansiti & Levien [4, 5] as an overall performance indicator of an ecosystem. This is further defined by three determinants; robustness, productivity and niche creation. For supplier selection, especially **robustness** is relevant, since it indicates the capability of an ecosystem to face and survive disruptions. Software vendors indicate that for them, **continuity of a software ecosystem and**

Table 1: Supplier selection strategies and their resulting trade-offs

Strategy	Trade-off	
Product integration with a hardware component supplier	Y	(+) Streamlined integration process by working with an "intimate hardware supplier" (-) Become dependent on a supplier
	N	(+) Independent of hardware supplier (-) Less streamlined integration process
Depending on large software ecosystem orchestrator	Y	(+) Benefit from the participation in a partnership model of a large software ecosystem orchestrator (+) Benefit from niche creation (+) Direct contact & support lines with the supplier (-) Become dependent on a large software ecosystem orchestrator
	N	(+) Remain independent (-) No benefits from niche creation (-) Less partnership model possibilities (-) Indirect contact & support lines with the supplier
Inclusion of open source components	Y	(+) Ability to steer an open source software project into a desired direction (+) Less license fees (-) Possible liability issues (-) More support and maintenance responsibilities
	N	(+) Avoid liability issues (+) Less support and maintenance responsibilities (-) Few strategic influence on the development of components
Minimal dependency on suppliers	Y	(+) Develop components in-house to decrease direct supplier dependencies (-) More resources required
	N	(+) No additional resources required (-) Remain dependent on suppliers

its products is vital when selecting a supplier for the most valuable components of a product. Software vendors therefore opt for stability, by selecting software ecosystems to join, as a customer or partner, based on these simple indicators. Related to this, are some of the indicators to measure ecosystem health, as defined in [3]. The **visibility of a software ecosystem within the market** is indicated as another important reason to choose for a certain supplier, especially for large ecosystems that provide core components. This is also reflected by quotes of software vendors such as; *"the supplier has a strong market position"*. Furthermore, **niche creation** is an important reason to join a software ecosystem. This is especially the case when selecting a platform that forms the basis for the product. A medium sized software vendor, for example, chooses to fully depend on Microsoft or SAP, providing additional functionality on top of one of their products or platforms to serve a niche within the ecosystem.

Choosing for different **product types**, in the form of open source components, becomes more interesting for software vendors. Examining **open source licenses** therefore, becomes prominent in the supplier selection process. This examination is performed regardless of the type of component, since inclusion of any component with a restricting open source license, can have serious consequences for the rights of use or redistribution for the total software product. Ruffin & Ebert [14] discuss several major legal aspects and three major risks with regard to the inclusion of open source software and how to mitigate them during product development. The first risk concerns directly integrating open source software into the source code, and then reproducing or selling the product without permission of the licensor. As a consequence, the licensor might claim damages or force the product vendor to terminate production, delivery and sale. Secondly, open source software is often a compilation of code from many sources. Because of these many sources, it is not an easy task to identify which parts, if at all, relate to protected intellectual property rights. Infringement of patents of third parties or other intellectual property rights might be the consequence of being unaware of including open source software that is protected by rights. Third, when an open source tool is used to, for instance, generate output that contains tool-created comments and a specific structure, the resulting work might be considered a derivative work. In this case, the owner of the tool and thus the copyright holder will be given certain rights to the resulting product. In all three discussed risks, an examination by an expert (e.g. judicial expert) often becomes emergent. As a consequence of these risks, software vendors indicate to be holding back when choosing for the inclusion of open source components.

Several participants primarily belonging to the large company size category mentioned that for certain components they prefer open source over closed source. The reason they gave was that using open source components leads to less supplier dependence which makes it an attractive alternative. They did not, however, mention anything about possible copyright violations due to the inclusion of open source software protected by intellectual property rights, making them vulnerable for the associated major risks as described in the beginning of this section. It is questionable whether the participants are aware of the major risks concerning the inclusion of open source software within their own commercial products. While this aspect is critical for all components, only a few software vendors indicate to employ strict policies about which open source licensed components can be included into a product.

Table 2: Classification of factors influencing supplier relationships and selection per product component type

Category	Factor	Critical core component	Non critical core component	Critical context component	Non critical context component
Supplier related factors	Perceived level of intimacy	Intimate	Familiar	Intimate	Unfamiliar
	Continuity	Y	Y	Y	N
	Visibility within the market	Y	Y	N	N
	Niche creation	Y	N	N	N
Product related factors	Product & license type	Y	Y	Y	Y
	Support & maintenance	Y	N	Y	N

Support and maintenance flows play a decisive role when selecting both critical core and context component suppliers. For critical components a well organized support and maintenance flow is essential. Therefore, continuity of maintenance and support and direct lines with suppliers are identified as important triggers for supplier selection. These maintenance and support interactions make supply management in the (product) software industry different from this practice in other industries [7]. As a result, partnering with these suppliers becomes interesting to shorten these lines, definitely when these support and maintenance flows are part of the additional services that a software vendor offers to create additional revenue streams.

Table 2 is the result of classifying the factors described in this section. By taking the selection criteria as described into account, we created a classification table to identify which factors are important when selecting a supplier for a certain type of component and what the average grade of intimacy is for this supplier relationship. It is important to note, that selection criteria related to functionalities or perceived added values of a certain component and costs are not included in this table, since they will be applicable for all categories and are therefore trivial.

7. DISCUSSION

In this paper, we addressed supplier selection from a software ecosystems and portfolio perspective. We also addressed component classification, supplier strategies and the way in which the perceived level of importance of a component influences factors that are at play when selecting suppliers and strategies. The data used for this analysis have been gathered by means of case studies, conducted by bachelor students during the Product Software course that is part of the bachelor in Information Science curriculum at Utrecht University. Because of the nature of the data collection process, we chose for a data selection process to enhance the validity of the final dataset. Nevertheless, the generalizability of the results is limited, because of the nature of the data gathering process. Apart from that, twenty-seven companies make up only a small part of the total Dutch product software industry. Furthermore, we cannot state that the Software Supply Networks include all the suppliers or that the Product Deployment Contexts contain all components. Some software vendors may not be aware of some small (open source) components that have, consciously or unconsciously, been incorporated into the end product.

The next step to take to generalize the findings presented in this paper, is carrying out a large sample survey with product software companies and perform an analysis on the dataset, similar to the one employed in this paper. Also, case studies using unique or representative cases [17] can contribute to provide a higher level of generalization of the findings presented in this paper. The same also applies for the proposed matrix. Further validation, from a PDC or system architectural perspective, is needed.

8. CONCLUSION

In this paper, we addressed supplier selection and strategies from a software ecosystems and portfolio perspective, using data gathered through twenty-seven case studies with Dutch product software companies. Software products consist of multiple components, only part of which developed in-house. We created a matrix that classifies components that are part of the direct running environment of a product. First of all, a distinction between core and context components was made. Core components are the essential building blocks of the product while context components provide additional functionalities resulting in added-value. In a second level of decomposition, we distinguished between critical and non-critical components. A critical component is a component that is not easily interchangeable or that adds significant added value to the product. The presented matrix can be valuable from both a Product Deployment Context perspective as well as a system architectural perspective.

Four main supplier strategies were identified. Some software vendors choose to become fully dependent on a large software ecosystem orchestrator. While increasing supplier dependence, it brings opportunities because of a perceived guarantee of continuation of support as well as offered additional benefits. On the total contrary, software vendors opt for a minimal dependence. Software vendors also indicated to be reviewing current supplier relationships because of the advent of open source. Having to leverage continuous support and maintenance, however, is experienced as a drawback of open source. In addition, open source software licenses need to be reviewed to avoid risks associated with the inclusion of open source components within a product.

Software vendors employ criteria when selecting suppliers. Critical core component supplier relationships have a high perceived level of intimacy compared to non-critical component suppliers. In addition, suppliers are selected based on their software ecosystem health, a performance indicator of an ecosystem. Also, support and maintenance plays a role when selecting both core and context component suppliers, especially when a component is classified as critical.

More research needs to be employed to provide further evaluation and validation. Furthermore, studies need to be addressed on software ecosystem selection to gain insight into mechanisms that are at play when deciding on what ecosystem to join.

9. REFERENCES

[1] V. Boucharas, S. Jansen, and S. Brinkkemper. Formalizing software ecosystem modeling. In *Proceedings of the 1st International Workshop on Open Component Ecosystems*, pages 41–50, 2009.

[2] S. Brinkkemper, I. van Soest, and S. Jansen. Modeling of product software businesses: Investigation into industry product and channel typologies. In *Information Systems Development*, pages 307–325. Springer, 2009.

[3] E. den Hartigh, M. Tol, and W. Visscher. The health measurement of a business ecosystem. In *Proceedings of the European Network on Chaos and Complexity Research and Management Practice Meeting*, 2006.

[4] M. Iansiti and R. Levien. *The Keystone Advantage: What the New Dynamics of Business Ecosystems Mean for Strategy, Innovation, and Sustainability.* Harvard Business School Press, 2004.

[5] M. Iansiti and R. Levien. Strategy as ecology. *Harvard Business Review*, 82(3):68–78, 2004.

[6] S. Jansen, S. Brinkkemper, and A. Finkelstein. Providing transparency in the business of software: A modeling technique for software supply networks. *Advances in Information and Communication Technology*, 243:677–686, 2007.

[7] S. Jansen, S. Brinkkemper, and A. Finkelstein. Component assembly mechanisms and relationship intimacy in a software supply network. In *15th International Annual EurOMA Conference, Special Interest Session on Software Supply Chains*, 2008.

[8] S. Jansen, S. Brinkkemper, and A. Finkelstein. Business network management as a survival strategy: A tale of two software ecosystems. *1st International Workshop on Software Ecosystems*, 505:34–48, 2009.

[9] S. Jansen, A. Finkelstein, and S. Brinkkemper. Providing transparency in the business of software: A modelling technique for software supply networks. In *Proceedings of the 8th IFIP Working Conference on Virtual Enterprises*, pages 677–686, 2007.

[10] S. Jansen, A. Finkelstein, and S. Brinkkemper. A sense of community: A research agenda for software ecosystems. In *ICSE 09: Proceedings of the 31st ICSE Conference on Software Engineering*, pages 187–190, 2009.

[11] M. Maloni and W. C. Benton. Power influences in the supply chain. *Emerald Management Reviews*, 21:(3):49–73, 2000.

[12] A. Osterwalder and Y. Pigneur. *Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers.* Wiley, 2010.

[13] K. Popp and R. Meyer. *Profit from Software Ecosystems: Professional Edition, Business Models, Ecosystems and Partnerships in the Software Industrys.* Books on Demand GmbH, 2010.

[14] C. Ruffin and C. Ebert. Using open source software in product development: a primer. *IEEE Software*, 21(1):82–86, 2004.

[15] J. L. Ward and J. Peppard. *Strategic Planning for Information Systems.* Wiley, 3rd edition edition, 2002.

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

[17] R. K. Yin. *Case Study Research: Design and Methods.* Sage Publications, Inc, 2008.