TECHNISCHE UNIVERSITÄT MÜNCHEN

Dr. Theo Schöller-Stiftungslehrstuhl für Technologie- und Innovationsmanagement

IP Modularity in Software Products and Software Platform Ecosystems

Josef Waltl

Vollständiger Abdruck der von der Fakultät für Wirtschaftswissenschaften der Technischen Universität München zur Erlangung des akademischen Grades eines Doktors der Wirtschaftswissenschaften (Dr. rer. pol.) genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Stefan Minner
Prüfer der Dissertation:	1. Univ.-Prof. Dr. Joachim Henkel
	2. Univ.-Prof. Dr. Oliver Alexy

Die Dissertation wurde am 24.01.2013 bei der Technischen Universität München eingereicht und durch die Fakultät für Wirtschaftswissenschaften am 12.02.2013 angenommen.

Life is like riding a bicycle. To keep your balance you must keep moving.

Albert Einstein (1879 - 1955)

Acknowledgements

First of all, I would like to thank Prof. Dr. Joachim Henkel for his outstanding support as a thoughtful supervisor. His enthusiasm and energy for the research on IP modularity were always inspiring to me and guided me throughout my whole dissertation.

This research project would never have been successful without the support from the researched software companies. For SugarCRM I would like to thank Elena Annuzzi, Nick Halsey, John Mertic and Clint Oram. For Salesforce.com the credits go to Kimia Poursaleh and for SAP special thanks goes to Dr. Karl-Michael Popp for his excellent support and all the effort he put into our joint research. I would also like to acknowledge the interview partners in a company whose name has to be kept undisclosed for the excellent support and the willingness to share their insights, despite the additional effort to control each statement for non-disclosure of confidential company information.

Furthermore sincere thanks are due to the team of the Dr. Theo Schöller Chair of Technology and Innovation Management for the inspiring time and the pleasant cooperation.

Special credits go to Christoph Krauß and Kristina Schreiner for their help in data collection and their valuable comments on my analysis results.

Finally, this dissertation would not have been possible without the never-ending support from Marianne, Josef , Martin and Claudia.

Table of Contents

List of Appendices

List of Figures

List of Tables

List of Abbreviations

AGPL	Affero GPL
APIs	Application Programming Interfaces
BSD	Berkeley Software Distribution
CDO	Chief Development Officer
CEO	Chief Executive Officer
CIO	Chief Information Officer
CRM	Customer Relationship Management
CTO	Chief Technology Officer
DSM	Design Structure Matrix
FOSS	Free- and Open Source Software
GNU	GNU’s Not Unix
GPL	GNU General Public License
IP	Intellectual Property
IPRs	Intellectual Property Rights
ISV	Independent Software Vendor
JAR	Java Archive
MIT	Massachusetts Institute of Technology
OLS	Ordinary Least Squares
OSS	Open Source Software
SaaP	Software as a Product
SaaS	Software as a Service
SCRM	SugarCRM
SFDC	Salesforce.com
SI	System Integrator
VIF	Variance Inflation Factor
VP	Vice President

Zusammenfassung

Die Arbeit untersucht die Auswirkungen einer modularen Softwarearchitektur, die durch die Optimierung von Wertaneignungsmechanismen entstanden ist (IP Modularität), auf Softwareprodukte und Softwareplattform-Ökosysteme. Ein System ist IP-modular, wenn die internen Modulgrenzen so gezogen werden, dass die jeweiligen Module ausschließlich Elemente enthalten, die in Bezug auf geistige Eigentumsrechte identisch behandelt werden können. Diese Rechte können in Form von Lizenzrechten, Urheberrecht aber auch informell, zum Beispiel durch Geheimhaltung von Quellcode, in Erscheinung treten.

Die Ergebnisse dieser Dissertation basieren auf einer detaillierten qualitativen Fallstudienanalyse von Softwareprodukten und -plattformen sowie einer quantitativen Studie in zwei Plattform-Ökosystemen.

Durch das Aufzeigen eines direkten Zusammenhanges von IP-modularer Produkt- oder Plattformarchitektur mit dem entsprechenden Geschäftsmodell zur Wertaneignung erweitern die Ergebnisse der Arbeit die bestehende Literatur zu IP Modularität. Es wird gezeigt, dass eine IP-modulare Architektur eine partielle Offenheit erlaubt, die verteile Wertschöpfung begünstigt. Zusätzlich wurde die frühe Berücksichtigung von Anforderungen mit Bezug auf geistiges Eigentum in der Anforderungsanalyse (Requirements Engineering) von Softwaresystemen als wesentlicher Treiber zur Verhinderung von zeit- und kostenaufwändigen Re-Modularisierungen identifiziert.

Die Ergebnisse der quantitativen Studie zeigen, dass die Ausprägungen IP-modularer Plattformarchitekturen, durch die Möglichkeit zu größerer Offenheit, bei gleichzeitiger Sicherstellung der Wertaneignung, die Plattformattraktivität für Designer plattformspezifischer Zusatzapplikationen erhöhen können.

Zusammenfassend zeigt die Arbeit die Verbindung von Management geistigen Eigentums, Softwarearchitektur und der jeweiligen Geschäftsmodelle von Softwareprodukt oder -plattform Anbietern auf.

Abstract

This dissertation examines the impact of Intellectual Property (IP) modular architecture on software products and software platform ecosystems. A software system is IP modular when its module boundaries separate parts of a system that have to be treated differently with respect to IP. The IP status is then homogeneous within each module, but may differ between modules. IP rights can be formal IP such as licensing contracts or copyright, but also informal IP like keeping the source code secret.

The presented results in this dissertation are based on a detailed qualitative case study analysis of two software products and two software platforms and on a quantitative study of two software ecosystems.

The results extend the existing literature on IP modularity by demonstrating a direct association between IP modular product or platform architecture and the related business models. The analysis also shows that the early consideration of IP-related requirements in the requirements engineering process of software systems can prevent costly and time-consuming re-modularizations.

The quantitative analysis in two software ecosystems shows that IP modular platform architecture, which can allow increased openness while still maintaining value appropriation, can increase a platform's attractiveness for complementors.

To summarize, this dissertation demonstrates the connections between IP management, software architecture and the respective business models of software product or platform providers.

1 Introduction

This dissertation explores the implications of the new concept of Intellectual Property (IP) modularity (Henkel and Baldwin, 2010, 2011) for the software product and software platform ecosystems domain. A system is IP modular when the module boundaries separate the parts that need to be treated differently with respect to IP. The IP status is accordingly homogeneous within each module, but it can differ between modules. IP rights can be formal IP, such as licensing contracts or copyright, but they can also be informal IP, such as keeping source code secret. This research endeavor focuses exclusively on the software domain because IP is the core asset of each software business, and the modularity of software systems can be adapted to a variety of requirements.

More broadly, this dissertation links research on IP modularity with research concerning software platforms (Gawer and Cusumano, 2002; West, 2003; Boudreau, 2010), multi-sided markets (Eisenmann *et al.*, 2006), software ecosystems (Jansen and Cusumano, 2012), software business models (Osterwalder, 2004; Weill *et al.*, 2005) and software requirements engineering (Wiegers, 2003; Chung and do Prado Leite, 2009).

To motivate the research on IP modularity Henkel and Baldwin (2010) consider, among others, the case of the video game Counter-Strike (Jeppesen and Molin, 2003). When the software publisher Valve Software released the video game Half-Life in 1998, it divided its codebase into two different modules. Valve Software put the game engine under a proprietary license and kept its source code secret, whereas it made the remaining application source code available to users under a broad license that allowed users to modify and share the code. Within approximately one and a half years of the original release of Half-Life, users generated the game Counter-Strike, which surpassed the success of the original game and created significant additional revenue for Valve Software, given that Counter-Strike players had to license and reuse the Half-Life game engine. This example shows the potential benefits that IP modular design could have for companies in the software domain, where module boundaries are flexible and technological entry barriers for value co-creators are low.

Already initial interviews in the early stages of this research project with industry practitioners have revealed that IP modular design is of special relevance in software platforms, which by nature face the challenge of optimizing value creation in the whole ecosystem of complementors and value appropriation for the platform providers. The exchange with managers from software platform companies confirmed the link between their IP modular platform designs and their business strategies; the managers also confirmed the need for a better understanding of these IP mechanisms on a more conceptual level.

In the software domain, there are many examples of IP modularity, but little is known about the exact reasoning that led to those IP modular designs. To my knowledge this dissertation presents the first empirical study to shed light on the effects of IP modularity in the software domain. The main research objective of this dissertation is formulated as follows:

Research objective: *What are the effects of IP modularity on software products and software platform ecosystems?*

This dissertation aims to answer the main research objective through three different perspectives. First, there is the perspective of the providers of software products. Second, there is the perspective of the providers of software platforms. Third, there are the complementors who generate additional applications for software platforms. Based on these perspectives, the structure of this dissertation is as follows:

Section 2 introduces IP modularity as the main theoretical concept of this dissertation. This section also presents the basics of system design and introduces the concept of modularity and the main drivers of modularizing technical systems. The section concludes by presenting IP modularity in software systems with concrete examples.

Section 3 presents the research methodology applied in this dissertation. First, in section 3.1 a hybrid research approach is identified as the most suitable method, given the current progress in the research field. In section 3.2 a detailed description of the case study methodology builds a solid methodological foundation for the in-depth qualitative research conducted in this dissertation. This section not only describes the applied research methodology, but it also guides the reader through the case selection process. A thorough understanding of this section is therefore vital for the interpretation of the case results. Finally, section 3.3 describes the quantitative analysis.

Section 4 addresses IP modularity from the perspective of a software product provider. First, the section presents the basics of software product design and software business models. The analysis of an engineering software case for outgoing IP modularity and of a data management software case for incoming IP modularity build the empirical foundation to answer the research questions regarding IP modularity in software products:

- Why are software products modularized with regard to IP considerations?
- How do the intended effects of IP modular product design relate to the real effects?

Finally, in this section, a cross-case comparison of the identified effects uncovers the similarities of both cases and leads to the formulation of additional propositions about the impact of IP modular software product design.

Section 5 uncovers the impact of IP modular design on software platform design from a software platform provider's perspective. Based on a review of the core concepts of software platforms and related ecosystems, two case studies on popular software platforms form the empirical basis to answer the research questions of this section:

- Why are software platforms modularized with regard to IP considerations?
- How does IP modular platform design influence the cooperation between platform providers and complementors?

The findings from the first case on SugarCRM confirm prior findings by Henkel and Baldwin (2010) and lead to the formulation of the research hypotheses for later quantitative tests. The second case on SAP NetWeaver PI suggests how the IP modular design can increase the attractiveness of a proprietary software platform. Finally, the cross-case analysis uncovers the common effects of IP modular platform design on platform development, platform attractiveness for complementors and platform attractiveness for end-users.

Section 6 follows up on the findings of the previous section with a quantitative research approach. It tests the findings from the qualitative case analysis on the levers of platform attractiveness. In this section, the perspective switches from the platform providers to the complementors. The analysis aims to answer the following research question:

- How does IP modularity influence the attractiveness of a software platform from a complementor's perspective?

The analysis is based on a platform attractiveness model[1] that describes the variables that influence the platform attractiveness in the platform provider setting and in the complementor setting. With reference to the qualitative findings from Section 5 and propositions on outgoing IP modularity from Henkel and Baldwin (2010) two hypotheses are formulated and tested with a regression analysis.

Finally, Section 7 draws conclusions for two target audiences. The first audience is the scientific community, for which this dissertation embeds the results in the broader discussion on related streams of research and makes suggestions for further research. The second audience are practitioners, such as general managers in software companies, ecosystem managers or software architects, for whom the managerial implications of IP modular design are discussed.

All results presented in this dissertation are based on my own work unless stated otherwise. All results from other researchers are carefully referenced. However, it is my deepest belief that creative ideas do not only sparkle in the mind of a single researcher. To reflect this belief and to recognize the efforts of my co-authors in earlier publications on the topic and other members of the research community to critically review my results, I purposely use "we" to present the results of this dissertation.

[1] The model is based on the Master's thesis of Schreiner (2012) that I initiated and supervised. For the analysis in this dissertation the original model was adapted and simplified.

2 The concept of IP modularity

In this chapter we introduce the concept of IP modularity as the central topic of this dissertation. We start with a basic description of design and introduce modularity as a design concept. Subsequently, we show the impact of modular design on value appropriation with a special focus on the software domain. Finally, we present IP modularity as a means to combine the benefits of modularity with the goal of value appropriation.

2.1 The basics of design

To understand modular design, it is important to first understand the basics of design. According to Baldwin and Clark, design is a complete description of an artifact (Baldwin and Clark, 2000, p. 21). The artifact can be a physical object or, as in the context of this dissertation, a virtual object such as a software source code. The design specifies all parameters of an artifact. All interdependencies between the design parameters define the design structure of an artifact (Baldwin and Clark, 2000, p. 21).

This design structure can be visualized with a tool named design structure matrix (DSM), invented by Steward (Steward, 1981) and refined by Eppinger (Eppinger, 1991). The use of design structure matrices can be illustrated with the simple example of the design of a mug (Baldwin and Clark, 2000, p. 21). The design parameters are named from P1 to P10, and the interdependencies are displayed with *X* marks in the DSM. These marks refer to an *is input to* relationship and mean that the specification of one column design parameter affects the design of the corresponding row design parameter (Eppinger, 1991, p. 285).

Figure 1 a) shows such a hierarchical relationship in the mug example. The manufacturing process influences the possible height but not vice versa. For the design process of the mug, this influence implies that manufacturing process can be specified first, regardless of the height. Once the manufacturing process is defined, the height cannot be chosen without restrictions. The design process is strictly sequential.

a) Hierarchy

		P3	P4
Manufacturing process	**P3**	▪	
Height	**P4**	X	▪

b) Interdependence

		P1	P2
Material	**P1**	▪	X
Tolerance	**P2**	X	▪

Figure 1 – Design structure (based on Baldwin and Clark, 2000)

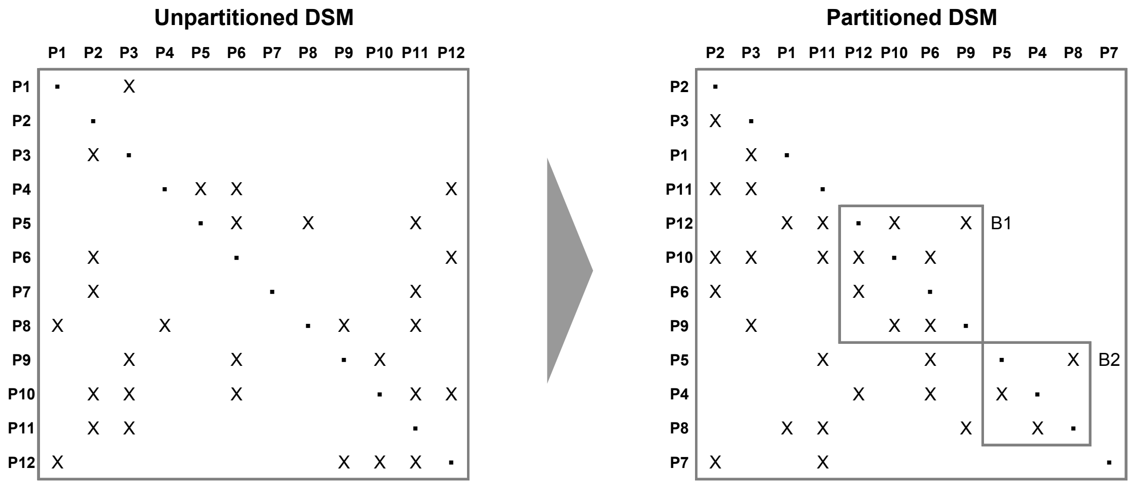
If the design parameters mutually depend on each other, the relationship is interdependent as shown in Figure 1 b). Here, the material and the tolerance depend on each other. A change in the specification of one parameter requires the designer to adapt the specification of the other parameter. Design parameters can also be fully independent from one another, as in the case of the height and tolerance in the mug example. The full design structure matrix of the mug example is shown in Figure 2 which presents an input-output table of design parameter choices (Baldwin and Clark, 2000, p. 41).

		P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Material	**P1**	▪	X	X			X	X	X		X
Tolerance	**P2**	X	▪	X			X	X	X	X	X
Manufacturing process	**P3**	X	X	▪			X	X	X	X	X
Height	**P4**			X	▪	X			X		X
Vessel Diameter	**P5**		X	X	X	▪	X	X	X		
Width of Walls	**P6**	X	X	X	X	X	▪	X	X		
Type of Walls	**P7**	X	X	X		X	X	▪	X	X	
Weight	**P8**	X		X	X	X	X	X	▪	X	
Handle Material	**P9**	X	X	X				X	X	▪	X
Handle Shape	**P10**	X	X	X	X					X	▪

Figure 2 – Design structure matrix of a mug (based on Baldwin and Clark, 2000)

For the design process, it is optimal if all marks are below the diagonal. The matrix is then called lower triangular, and the design process can be strictly sequential (Eppinger, 1991, p. 285). In this case, the designer can specify one design parameter after another without cycles in the design process.

Unfortunately, in reality this lower triangular form is rarely found. Thus, the DSM has to be reordered to bring as many marks as possible below the diagonal or in blocks around the diagonal. With this reordering, the design structure can be brought toward the lower triangular form, and the design process becomes more sequential (Steward, 1981; Eppinger, 1991). Figure 3 shows an example of this reordering process.



	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
P1	▪		X									
P2		▪										
P3		X	▪									
P4				▪	X	X						X
P5					▪	X		X			X	
P6		X				▪						X
P7		X					▪				X	
P8	X			X				▪	X		X	
P9			X			X			▪	X		
P10		X	X			X				▪	X	X
P11		X	X								▪	
P12	X								X	X	X	▪

	P2	P3	P1	P11	P12	P10	P6	P9	P5	P4	P8	P7
P2	▪											
P3	X	▪										
P1		X	▪									
P11	X	X		▪								
P12			X	X	▪	X		X				
P10	X	X		X	X	▪	X					
P6	X				X		▪					
P9		X				X	X	▪				
P5				X			X		▪		X	
P4					X		X		X	▪		
P8			X	X				X		X	▪	
P7	X			X								▪

Figure 3 – DSM-partitioning example (based on Eppinger *et al.*, 1994, p. 3)

Within this re-arranged design structure matrix, the two blocks B1 and B2 can be identified. B2 is dependent on the input of B1; however, B1 does not depend on B2.

By re-arranging these design parameters, the building of blocks and the resulting input relationships, the basics for modular design are laid. Following-up on this example, the next section will present the details of modular systems design.

2.2 Modularity in technical systems

"In a world of change, modularity is generally worth the cost." (Langlois, 2002, p. 24)

In the previous section, the two blocks B1 and B2 emerged from the partitioning of the design process (see Figure 3). These blocks show strong interdependencies within themselves but no interdependencies among each other (Steward, 1981, p. 72; Eppinger *et al.*, 1994, p. 3). This relationship of strong coupling within elements and loose coupling across elements makes up modularity; or, as Baldwin and Clark describe:

“A Module is a unit whose structural elements are powerfully connected among themselves and relatively weakly connected to elements in other units. Clearly there are degrees of connection, thus there are gradations of modularity.” (Baldwin and Clark, 2000, p. 63)

Modularity is essential to the design of complex systems, considering that we have limited mental capabilities and can only process sub-segments of such systems at a time (Baldwin and Clark, 2000, p. 63). A modular system can only be superior over an integral system if its modularization is precise, unambiguous, and complete (Baldwin and Clark, 1997, p. 86).

The modularization of a system is based on a set of *visible design rules* across the whole system and *hidden design parameters* within each module. The visible design rules are fixed throughout the whole design process and limit the degrees of freedom a designer has. In contrast, the hidden design parameters increase the efficiency within the design process because they prevent unnecessary cycles (Baldwin and Clark, 1997, p. 86).

The visible design rules itself comprise of *architecture*, *interfaces* and *integration protocols* (Baldwin and Clark, 2000, p. 77).

First, the architecture specifies the modules of a system, built upon the primary purpose of each module. The architecture maps the modules in the design structure to the different functions of a product (Ulrich, 1995, p. 422).

Second, the interfaces define how these modules work together. The interfaces can be of physical nature similar to a hinge that connects a car door to the car body or, as is more relevant for this dissertation, of virtual nature (Ulrich, 1995, p. 422). Those virtual interfaces in the software domain describe the possible interactions among software modules. These interfaces can be manifold from simple function calls in object-oriented programming to sophisticated Application Programming Interfaces (APIs) across several software technologies.

Third, the integration of all modules into a functioning system needs to be tested with a defined integration protocol. Therefore, each module is tested against a defined standard that ensures conformity to its design rules (Baldwin and Clark, 1997, p. 86).

Hidden design parameters do not affect the modular system beyond the borders of a specific module. They describe the information that is not visible outside the module and therefore can be chosen late in the design process (Baldwin and Clark, 1997, p. 85).

In general, a module should hide as much information as possible to provide for a flexible inner structure. This concept of information hiding (also named black box concept) is of special relevance for the generation of complex software systems (Parnas, 1972, p. 1056; Yourdon and Constantine, 1979, p. 22). Software engineers generate modules that can be fully integrated into larger systems without knowledge of the inner operations of these modules. In addition to making a module's inner design more flexible, information hiding ensures that IP is kept within the boundaries of a module.

If the described design rules are followed, modularity provides a number of benefits: Single modules can be changed without the need of changing other modules (Ulrich, 1995, p. 423; Baldwin and Clark, 1997, p. 85). In that case modularity enables concurrent or asynchronous development of complex systems (Eppinger *et al.*, 1994, p. 1; Ethiraj *et al.*, 2008; LaMantia *et al.*, 2008, p. 5). Modularity also increases the flexibility to customize or alter a system (Schilling, 2000, pp. p. 312; Langlois, 2002, p. 24).

However, modularity does not come free of cost. Compared to an integral system, additional efforts are required to define the design rules, the architecture and the test of the entire system (Langlois, 2002, p. 23). Additionally, the modules may not work together properly in a poorly designed system. Nevertheless, modularity entails enormous flexibility for designers and users of a modular system (Baldwin and Clark, 1997, p. 85). As stated in the beginning of this section, as long as a system is subject to change, modularity is generally worth the additional cost (Langlois, 2002, p. 24).

The modular structure of virtual systems can much easier be adapted throughout their lifetime than in physical systems (Parnas *et al.*, 1985). Thus, software systems can easier be re-modularized ex-post, even if this typically entails additional cost and risks. The decision for or against such a modular design is a managerial choice that can have a significant impact on a product's business performance. Therefore, the modular product design is a constant element of managerial decisions in software companies (MacCormack *et al.*, 2006, pp. 1027–1028).

Modularity generates option value (the right but not an obligation) to adapt an architecture (Baldwin and Clark, 2006, p. 1126). This option allows managers in software companies to consider opening-up a product for other partners and generating a platform or to implement open source software (OSS) business models. Recent research shows that modularity is of special importance for OSS projects and can attract developers to join a project (Lerner and Tirole, 2002, p. 220; Baldwin and Clark, 2006, p. 1116).

Figure 4 shows that the ideal modular design is always a compromise between different rationales. *Distributed development* and *flexibility in design* are the two most common rationales software architects have to balance.

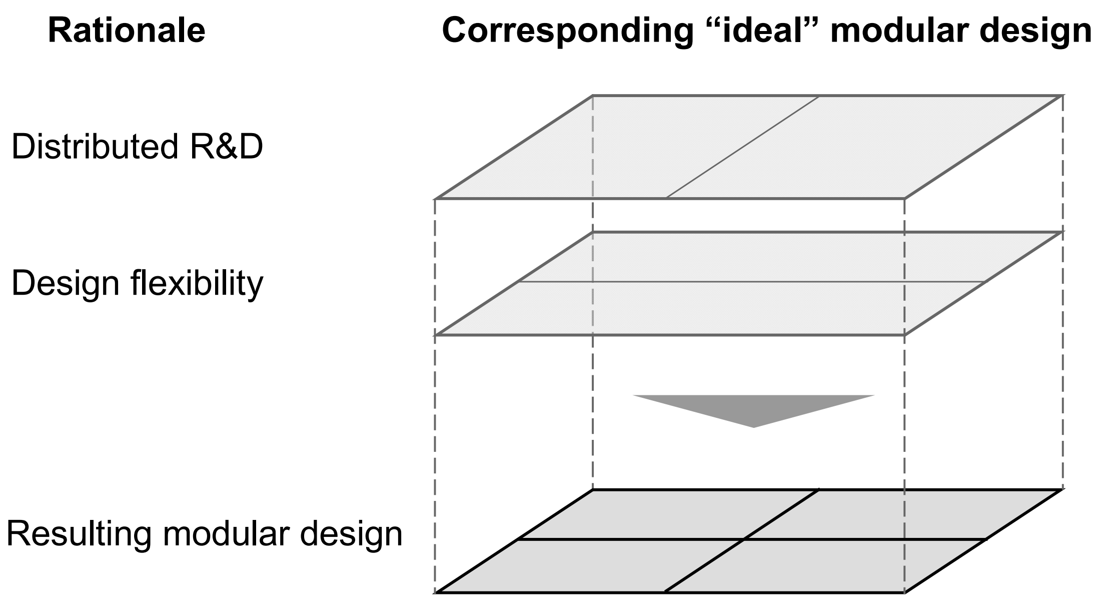


Figure 4 – Rationales for modular design (based on Henkel, 2011)

Distributed R&D refers to the requirement of having several modules whose development can be split up across geographically different R&D facilities (MacCormack *et al.*, 2006, p. 1027). Vice versa, recent research shows that distributed software development organizations are more likely to produce modular designs. This phenomenon is named *mirroring hypothesis* (Sanchez and Mahoney, 1996).

Modularity keeps complex software flexible. It allows a designer to change a subset of modules and re-use other modules to adapt a product to new requirements. In addition, customers benefit from modularity because they can mix and match modules that best satisfy their needs (Baldwin and Clark, 1997, p. 85, 2000, p. 59).

2.3 IP modularity

A designer of a complex software system significantly invests before being able to appropriate returns from his innovation. In his fundamental work on value appropriation, Teece (1986) shows that firms often fail to appropriate value from their innovations while customers, immitators and other industry players benefit.

As described in the previous chapter, modularity allows different parties to work on modules independently. These parties can be various R&D departments within a firm but also other organizations. As such, modular architecture enables the splitting of innovation across firms, eventually resulting in an overall higher innovation activity within the whole domain (Sanchez and Mahoney, 1996, p. 74). However, modular systems also entail the risk of imitation of modules by third-party providers (Baldwin and Clark, 1997, p. 86; Ethiraj *et al.*, 2008).

Baldwin (2007, pp. 179–180) exemplifies this risk of an innovator failing to profit from innovation with IBM's introduction of the System/360. By modularizing a former integral system, IBM triggered the development of substitutes for some of the IBM System/360 modules by other companies. These companies could supply IBM customers with peripheral devices with a better price-performance ratio than that of IBM's own products, resulting in a lower value that IBM could appropriate from its innovation. Even worse, IBM lost control of the platform, and the biggest value share was appropriated by its complementors.

The right strategy for value appropriation is of special relevance for software platform providers that face the conflict between value appropriation and value creation in their ecosystem (Cusumano, 2010, p. 43). Further details on IP modularity in software platform ecosystems will be presented in Section 5.

This tension of value creation and value appropriation could be managed with the concept of IP modularity as introduced first by Henkel and Baldwin (Henkel and Baldwin, 2010, 2011). For this dissertation, we define a modular software system as follows:

> *"**IP modular software system:** A software system is IP modular if its technical module boundaries coincide with the boundaries of parts defined by homogeneous IP rights. These IP rights not only comprise formal IP such as patents, copyrights and licensing contracts but also informal IP such as secrecy*

(realized, e.g., by not disclosing a program's source code)." (Waltl et al., 2012, p. 95)

This definition introduces IP management to optimize value appropriation as an additional rationale for the modularization of a software system. Figure 5 shows a case where value appropriation is the dominant criterion for the modularization of a software system (illustrated with the bold line in the resulting modular design). For the research purposes of this dissertation, we focus on such cases to learn more about the decision-making rationale.

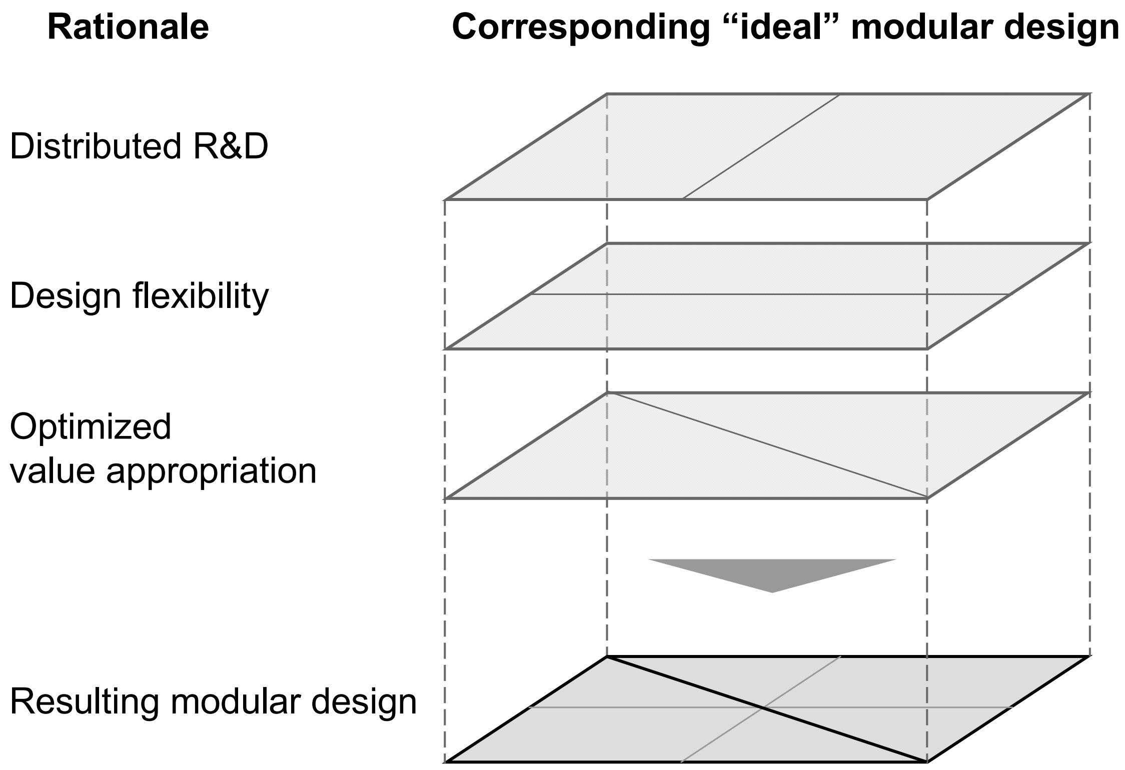
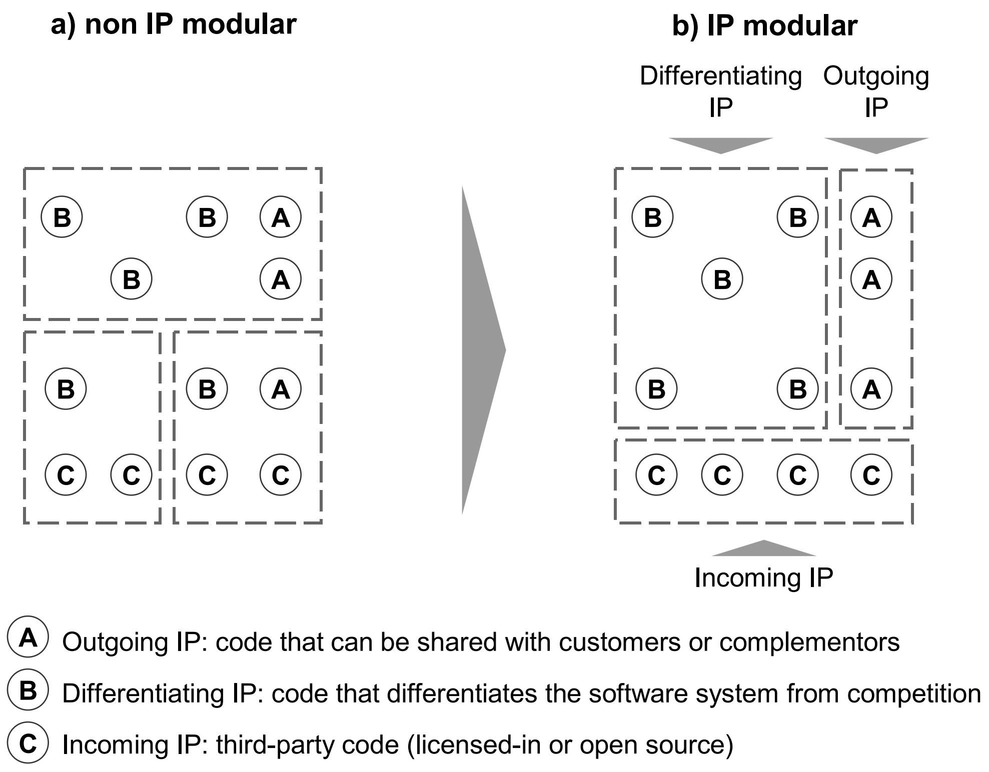


Figure 5 – Optimized value appropriation as rationale for modularization

Intellectual property is the core asset of software product companies because their business model is based on allowing customers to use their IP for a certain compensation (Popp, 2011, p. 27). Many modern software systems are highly dependent on externally developed components because it is not economically efficient to develop everything in-house. Thus, modern software systems comprise of own developed IP but also of third-party IP.

To illustrate IP modularity, we can imagine a software system that is modularized based on a certain rationale such as distributed R&D or flexibility as shown in Figure 6 a).



a) non IP modular
b) IP modular
Differentiating IP
Outgoing IP
Incoming IP
A Outgoing IP: code that can be shared with customers or complementors
B Differentiating IP: code that differentiates the software system from competition
C Incoming IP: third-party code (licensed-in or open source)

Figure 6 – IP incompatibility

In the non IP modular case shown above the system consists of linked elements of which each carries a certain IP status of *Outgoing IP*, *Differentiating IP* or *Incoming IP* (Henkel and Baldwin, 2010, p. 10). *Outgoing IP* could be code that can be shared with customers or complementors to improve cooperation. *Differentiating IP* is code that differentiates the software system from competing systems that is essential for a company's business model and motivates customers to pay for the software system. *Incoming IP* is coming from third-parties, which could be licensed-in or open source code. We call such modules that include elements with different IP status *IP incompatible*.

This *IP incompatibility* can lead to strategic disadvantages. For example, if the source code of all artifacts labeled with *A* should be provided to a customer or a complementor, two modules are affected. As shown in Figure 6 a), the release of all *A* would also include *B* artifacts. By releasing B artifacts, a company would restrict its options to appropriate value for these elements that differentiate its offering from competition. Likewise, if the third-party IP marked with *C* should be exchanged in the non-IP modular case, two modules would be affected.

Modular software design with regard to IP can prevent the described risks. Figure 6 b) illustrates an IP modular software design with homogeneous IP statuses within each module. All artifacts with *Differentiating IP (B)* are clustered within a separate module.

The example also allows us to introduce the difference between incoming IP and outgoing IP as first described by Henkel and Baldwin (2010):

> *"With incoming IP [...] the IP under consideration is owned by some party external to the focal firm. The owner of the IP has determined the IP status of the knowledge under consideration, and the focal firm must accept the bundle of rights and possibilities it has been granted.*
>
> *With outgoing IP, the focal firm owns the IP under consideration, and is free to award to each element the IP status it finds most advantageous." (Henkel and Baldwin, 2010, p. 12)*

Henkel and Baldwin (2010) analyze the theoretical conditions under which IP modularity is beneficial and thus, assuming rational behavior on behalf of system architects, most likely to be observed. In particular, they derive propositions for incoming and outgoing IP (Henkel and Baldwin, 2010, pp. 19–21) :

Proposition 1: [Value Co-creation] *The greater the potential for value co-creation in the surrounding ecosystem, the more advantageous is outgoing IP modularity. Module boundaries should go between the open and proprietary IP.*

Proposition 2: [Distributed Co-creators] *The more distributed, numerous, and anonymous the co-creators of value, the more advantageous is outgoing IP modularity.*

Proposition 3: [Complexity] *The more complex the downstream system, the more advantageous is outgoing IP modularization. The module boundaries should separate the IP that helps to integrate the focal component into the system from that which contributes to the component's internal performance.*

Proposition 4: [Customization] *The greater and more varied the need for downstream adaptations, the more advantageous is outgoing IP modularization. The module boundaries should separate the IP that serves as the basis for modification from the IP supporting the proprietary "core" modules.*

Proposition 7: [Holdup Risk] *Incoming IP modularity is advantageous when the focal firm faces the risk of holdup from suppliers of incoming IP. The module boundaries should go between the firm's own IP and the incoming IP.*

These propositions are the starting point for the case study research in this dissertation as presented in the following sections.

3 Research methodology

This section describes the applied research methodology. Based on the research goals, we identify the optimal research approach. We provide a detailed description of all steps in the applied research approach and use concrete examples for illustration.

Section 3.1 addresses the principals of selecting the appropriate research methodology for management field research. Section 3.2 provides a detailed description of the qualitative approach employed for case study research – which is the core source of insight for this dissertation. Finally, Section 3.3 provides an overview of the methods used for quantitative hypotheses-testing applied for this dissertation.

3.1 Selection of a hybrid research approach

As described in the previous section, there is little empirical evidence on the concept of IP modularity in general and even less on IP modularity in software systems. Therefore, to our knowledge, this dissertation is the first empirical study to identify the strategic implications of IP modularity for software products and platforms.

Because of the novelty of this research field, selecting the appropriate research approach is highly important. Edmondson and McManus (2007) note that methodological fit is a key quality criterion in management field research that must be considered prior to the start of a research project. This type of fit is defined as internal consistency among the various elements of a research project, such as the research question and methodology.

In general, management field research can be divided into three categories that correspond to different types of theory. First, nascent field research focuses on completely new topics. In this type of research, primarily qualitative methods are applied to generate new theory (Eisenhardt, 1989, p. 536). Second, intermediate research aims to extend existing theory through the utilization of a mix of qualitative and quantitative methods. Third, mature research focuses on testing and detailing established theories using quantitative methods.

According to Edmondson and McManus (2007), methodological fit follows a mean tendency for these three types of theory and the type of data to be researched (see Figure 7).

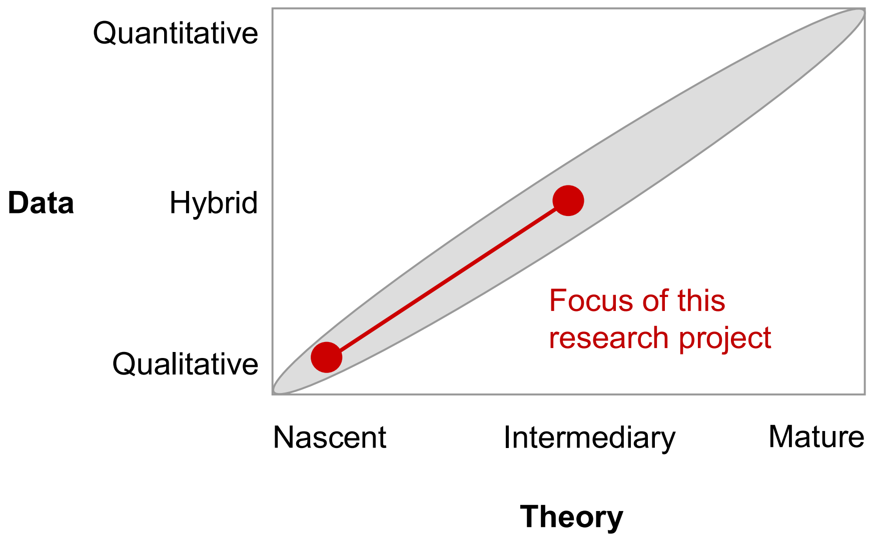


Figure 7 – Methodological fit
(based on Edmondson and McManus, 2007)

As for this research project, a mature scientific literature exists for each of the relevant research fields (modularity, platforms, value appropriation and software business models). We enter uncharted territory in combining these research fields with the new concept of IP modularity (Henkel and Baldwin, 2010, 2011). In particular, there is no existing theory on the impact of IP modularity on software products, platforms and platform ecosystems. Therefore, this research project can be categorized as research located between nascent and intermediary field research. It has a strong focus on qualitative research methods followed by first tests of the identified hypotheses through the use of quantitative methods. Edmondson and McManus (2007) describe the benefit of such a hybrid approach for scientific insight and rigor as follows:

> *"The combination of qualitative data to help elaborate a phenomenon and quantitative data to provide preliminary tests of relationships can promote both insight and rigor." (Edmondson and McManus, 2007, p. 1165)*

Applying this research approach, we begin with a detailed analysis of existing research in the focus research fields to derive basic research questions. Based on this analysis, the effects of IP modularity on software products and platforms are analyzed using qualitative methods. Finally, the effects of IP modularity on software platform attractiveness for providers of complementary software products are verified using quantitative methods (Mahoney, 2006, p. 230). Figure 8 shows the applied hybrid research process, which includes initial qualitative and subsequent quantitative analysis.

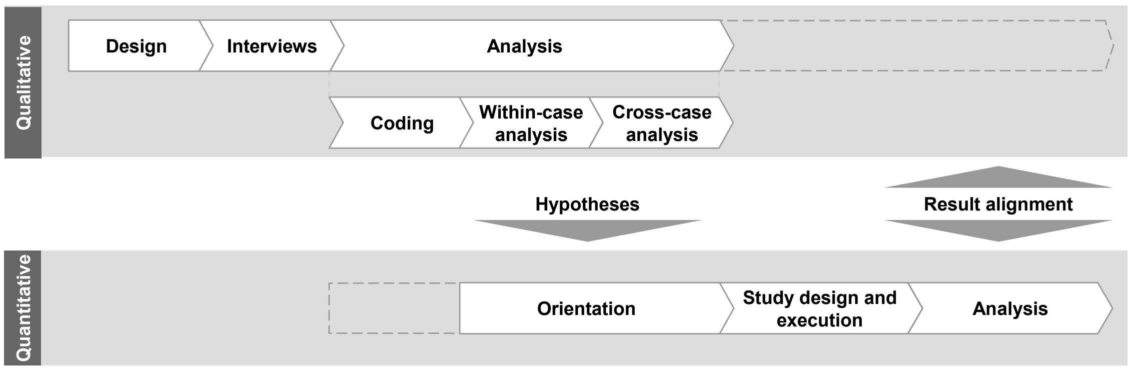


Figure 8 – Hybrid research process

For our novel research field, the qualitative research facilitates shaping of the research hypotheses that are tested using quantitative analysis. Finally, the results of hypotheses-testing are aligned with the expected results obtained from the initial qualitative analysis.

3.2 Case study research

Having identified qualitative research approaches as most suitable for the initiation of the research endeavor, the appropriate method must be chosen. The literature on qualitative research lists five methods: experiment, survey, archival analysis, history and case study (Yin, 2009, p. 8). The selection is based on the research question type, the degree of control over behavioral events and the degree of focus on contemporary events.

Given the research objectives of this dissertation, the case study method is identified as the best fit, as the research questions begin with “how” and “why” (see Section 1) and we have no control over the contemporary real-life events we study. Using this selection logic, we adhere to the work of Yin (2009), who describes a case study as follows:

> *"1. A case study is an empirical inquiry that investigates a contemporary phenomenon in depth and within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident.*
>
> *2. The case study inquiry copes with the technically distinctive situation in which there will be many more variables of interest than data points, and as one result relies on multiple sources of evidence, with data needing to converge in a triangulating fashion, and as another result benefits from the prior development of theoretical propositions to guide data collection and analysis." (Yin, 2009, p. 18)*

Case studies are becoming increasingly popular in management field research and information systems research (Dubé and Paré, 2003, p. 599). This trend has manifested itself in an increasing number of scientific publications in high-quality journals based on case study research in the last decade (Bansal and Corley, 2011, p. 234). One of the reasons for this growing popularity is that case study research is one of the best manners of generating novel theory that bridges the gap between qualitative evidence and mainstream deductive research (Eisenhardt and Grabner, 2007).

In its essence, a case study attempts to illuminate decisions. The final result reveals why particular decisions were made, how they were implemented, and the results of these decisions (Yin, 2009, p. 17). Case study research can be employed for various purposes: to provide description, to test theory and to generate new theory (Eisenhardt, 1989, p. 535). In the context of this dissertation, we focus on the generation of new theory that extends the existing research on IP modularity in the software domain.

Generating high-quality case study research is not trivial, and quality evaluation does not follow clear rules. However, a set of best practices has emerged in recent years. First, published papers follow a consistent structure (Pratt, 2009, pp. 858–861; Bansal and Corley, 2011, p. 234) . Typically, the sections are as follows: introduction, literature review, methods, findings and discussion. Second, there is an increasing tendency to code data. Third, the findings are illustrated through the use of detailed tables, graphs, diagrams and organizing figures. Finally, propositions are increasingly applied to display the study's theoretical contribution. The final report should be written in a concise manner and should read like a well crafted story (Pratt, 2009, pp. 858–861) . In addition, best practices are described best by Eisenhardt and Grabner (2007, p. 30):

"[...] challenges can be mitigated through precise language and thoughtful research design: careful justification of theory building, theoretical sampling of cases, interviews that limit informant bias, rich presentation of evidence in tables and appendixes, and clear statement of theoretical argument." (Eisenhardt and Grabner, 2007, p. 30)

Additionally, pitfalls such as reinventing the wheel and providing no significant contribution to existing theory can diminish the quality of a case study. These hurdles can be overcome by clearly pinpointing which theoretical conversations are affected by the results of the study and the current status of these conversations (Pratt, 2009). How the researcher engaged with the phenomenon of interest must be fully transparent, and evidence for the conclusions must be provided (Bansal and Corley, 2011, p. 236).

Deficiencies in methodology and rigor are critical for any type of social science research. These deficiencies are particularly problematic in qualitative case studies because they form the starting point for further quantitative research. Additionally, recommendations for practitioners regarding real-life management situations could lose relevance without rigor. Therefore, four criteria for ensuring validity and rigor in case study research should be followed, best described by Gibbert *et al.* (2008, pp. 1466–1469) and Yin (2009, pp. 40–45) and presented in the following paragraphs.

Internal validity is achieved when the research conclusion can be defended with compelling logical reasoning and plausible arguments. A clearly communicated research framework and constant comparison of the empirically identified patterns with those predicted can increase internal validity. We ensure internal validity through the use of a detailed description of the applied research framework.

During the data collection phase, *construct validity* must be considered. Construct validity is defined as the extent to which a study investigates what it claims to investigate. Researchers must identify the correct operational measures for the concepts that are studied (Yin, 2009). Furthermore, the triangulation of results through the use of data from multiple data collection methods is a measure to ensure construct validity (Gibbert *et al.*, 2008, pp. 1466–1469) . In this qualitative research, we endeavor a rich set of secondary data sources (see Table 4), such as company internal documents, analyst reports and field notes, which complement the data obtained from case interviews.

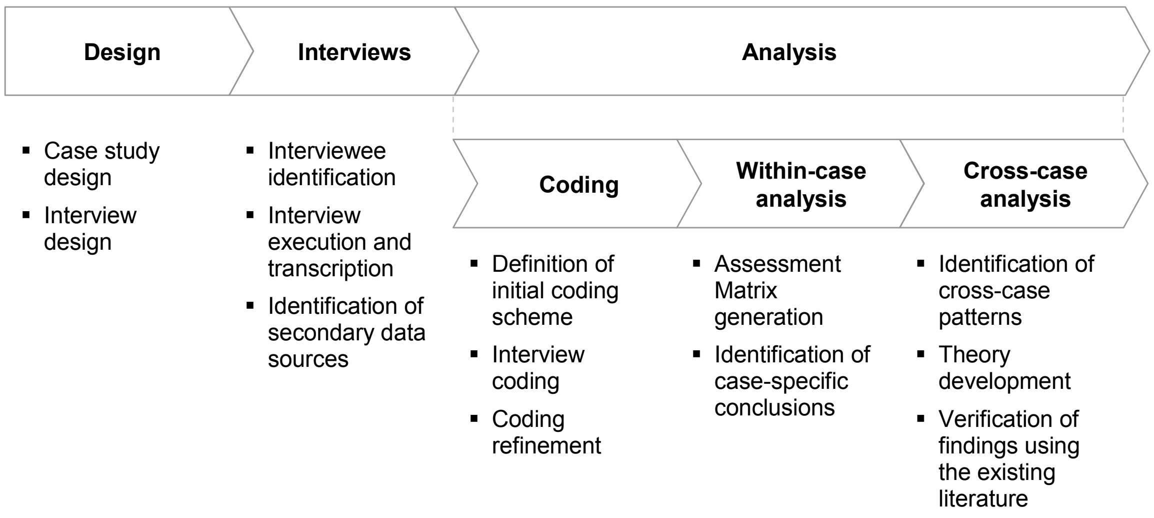
External validity measures to what extent the findings can be generalized beyond the concrete context of the study. Case studies can provide a good basis for analytical generalization when results are derived using cross-case comparison. Additionally, researchers should provide a clear rationale for case selection and describe the case context in as much detail as possible (Gibbert *et al.*, 2008, p. 1468). In this dissertation we ensure external validity with a clear case selection approach as shown in Figure 13.

A case study's *reliability* can be increased when all research steps are displayed in a highly transparent manner such that subsequent researchers can draw the same conclusions when they replicate the study. We ensure the reliability by providing all relevant data in the appendix of this dissertation. Following that a replication of the analysis is likely to lead to identical outcomes (Yin, 2009).

The literature provides a number of methods of generating novel theory from case study findings. The most fundamental method is grounded theory research, as introduced by Glaser and Strauss (2008). The basis for this method is the continuous comparison of data and theory, beginning with data collection. New theoretical categories emerge solely from evidence and an incremental approach to case selection and data gathering (Eisenhardt, 1989, p. 534). Grounded theory is best suited to gaining an understanding of the process how actors construct meaning out of intersubjective experience (Suddaby, 2006, p. 636). To identify patterned relationships between social actors, it is important to elevate data to a conceptual level through the use of a focused research process. The application of a grounded theory approach leads to a detailed case study description but does not entail the formulation of concrete testable hypotheses (Glaser and Strauss, 2008).

As our research is based on preexisting theory, we apply a positivist research approach (Eisenhardt, 1989, p. 546; Eisenhardt and Grabner, 2007; Dubé and Paré, 2003; Gephart, JR., 2004, p. 456). Through the contrasting of preexisting understandings, as well as observations made during concrete cases we want to further elaborate existing theory (Greenwood and Suddaby, 2006, p. 33).

To conclude this section, we argue that in management field research, case studies are a powerful tool for the generation of novel theory. However, the generation of high-quality case study reports is not trivial and is susceptible to criticism with regard to a number of possible pitfalls (Bansal and Corley, 2011). Thus, in this dissertation, a series of measures are taken to ensure quality in the derived results. The research process follows a three-phase process consisting of *design*, *interviews* and *analysis* (see Figure 9):



Design
Interviews
Analysis
Case study design
Interview design
Interviewee identification
Interview execution and transcription
Identification of secondary data sources
Coding
Within-case analysis
Cross-case analysis
Definition of initial coding scheme
Interview coding
Coding refinement
Assessment Matrix generation
Identification of case-specific conclusions
Identification of cross-case patterns
Theory development
Verification of findings using the existing literature

Figure 9 – Qualitative research process (based on Yin, 2009)

This process is based primarily on the work of Yin (2009), Eisenhardt (1989) and Miles and Huberman (1994). The following sections provide a detailed description of this process and its application in this dissertation.

3.2.1 Case design

As an initial step in the research process, the case design is specified. This specification provides a logical plan for progressing from the initial research questions to their corresponding answers. During this step, the research questions, possible propositions and the unit of analysis are defined (Yin, 2009, p. 27). Additionally, the research cases are identified based on a selection logic derived from the research topics.

Case study design

At the least, the basic research questions must be broadly specified at the beginning of a qualitative research endeavor. Throughout the research process, the research questions are adapted and finalized based on additional findings through the use of an iterative process (Eisenhardt, 1989, p. 536). It is common to derive these initial research questions from the existing literature.

The basic research questions (see Section 1) of this dissertation are formulated based on prior research on IP modularity (Henkel and Baldwin, 2010, 2011) and software platform ecosystems (West, 2003; Gawer, 2009; Eisenmann *et al.*, 2009; Boudreau, 2010; Cusumano, 2010) but are phrased in an open way to prevent the development of result bias from unilateral questioning:

Research topic one (see Section 4): IP modularity in software products

- Why are software products modularized with regard to IP considerations?
- How do the intended effects of IP modular product design relate to the real effects?

Research topic two (see Section 5): IP modularity in software platform ecosystems

- Why are software platforms modularized with regard to IP considerations?
- How does IP modular platform design influence the cooperation between platform provider and complementors?

Research topic three (see Section 6): The impact of IP modularity on platform attractiveness

- How does IP modularity influence the attractiveness of a software platform from a complementor's perspective?

We apply the following research framework to generate new theory on the effects of IP modularity in software products and software platform ecosystems (see Figure 10).

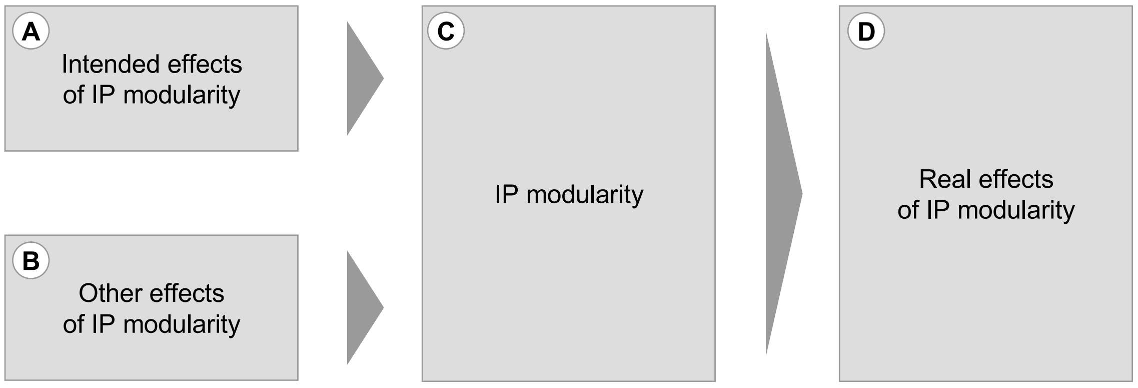

A
Intended effects of IP modularity
B
Other effects of IP modularity
C
IP modularity
D
Real effects of IP modularity

Figure 10 – Research framework

The arrows in Figure 10 show how the elements influence each other. The characteristics of an IP modular system (C) are influenced by intended (A) and other effects (B). The real effects (D) are influenced by the characteristics of the IP modular system (C).

Beginning from this point, the overall goal for research topic one and two is to generate cause and effect propositions with regard to IP modularity. For research topic one in the incoming IP modularity case we also aim to verify Henkel and Baldwin's holdup proposition (Henkel and Baldwin, 2010, p. 26):

Proposition 7: [Holdup Risk] *Incoming IP modularity is advantageous when the focal firm faces the risk of holdup from suppliers of incoming IP. The module boundaries should go between the firm's own IP and the incoming IP.*

While studying research topic three, the effects of IP modularity on software ecosystem complementors, basic case study propositions guide us through the research (Yin, 2009, p. 28). We base our research on Henkel and Baldwin's (2010) four propositions with regard to outgoing IP modularity (Henkel and Baldwin, 2010, pp. 19–21) :

Proposition 1: [Value Co-creation] *The greater the potential for value co-creation in the surrounding ecosystem, the more advantageous is outgoing IP modularity. Module boundaries should go between the open and proprietary IP.*

Proposition 2: [Distributed Co-creators] *The more distributed, numerous, and anonymous the co-creators of value, the more advantageous is outgoing IP modularity.*

Proposition 3: [Complexity] *The more complex the downstream system, the more advantageous is outgoing IP modularization. The module boundaries should separate the IP that helps to integrate the focal component into the system from that which contributes to the component's internal performance.*

Proposition 4: [Customization] *The greater and more varied the need for downstream adaptations, the more advantageous is outgoing IP modularization. The module boundaries should separate the IP that serves as the basis for modification from the IP supporting the proprietary "core" modules.*

After defining the basic research questions and identifying the primary research propositions, the research cases must be defined. The overall goal is to select cases that allow for generalization of the findings beyond the specific case. Drawing an analogy to quantitative hypotheses-testing, we can think of all possible cases that can answer the basic research questions as a population. This concept of a population facilitates formation of the boundaries for the generalization and applicability of the emerging theory (Yin, 2009, p. 15).

In a case study research project, a sample is drawn from this population. In contrast to quantitative research, a random selection is neither necessary nor preferable. Cases are chosen so that theoretical saturation can be achieved. When theoretical saturation occurs, the addition of one more case would not result in more insight regarding the identified theory. Additionally, the selection of polar cases can facilitate the identification of findings that can be generalized (Eisenhardt, 1989, p. 537). There is no fixed number of cases that should be included in a research project. The number of cases required to generate robust findings depends on what the researcher wants to answer (Pratt, 2009, p. 856). Eisenhardt (1989, p. 545) is more concrete and states that four to ten cases is sufficient for the generation of excellent results.

The third component of a case study research design is the unit of analysis. The exact definition derives from the primary research questions and can be an event, an entity or a single person (Yin, 2009, p. 29). For the qualitative research portions of this dissertation, we define the unit of analysis as follows:

Unit of analysis: *A specific software product or platform whose modular structure is influenced by considerations regarding the management of intellectual property rights.*

This definition of the researched entity is consistent with prior research on software product and platform modularity (Baldwin and Clark, 2000; Sosa *et al.*, 2004; Baldwin and Clark, 2006; MacCormack *et al.*, 2006; LaMantia *et al.*, 2008; Baldwin and Woodard, 2009). Using this definition of the unit of analysis, we ensure that our findings are generalizable and can be aligned with existing theory.

In this dissertation we follow a multi-case design approach, as this approach is likely to deliver more robust and compelling results and aid the researcher in drawing a more precise theoretical picture (Eisenhardt and Grabner, 2007, p. 27). A multi-case study can be compared to the analysis of multiple experiments. Similarly to the experimental research requirement for careful experiment preparation, multi-case research requires careful case selection. Case selection follows a replication logic, not a sampling logic, as in quantitative research. Replication logic is aimed toward the duplication of the results with altered conditions, resulting in a more robust theory.

To answer the research questions addressed in this dissertation, a theoretical replication approach is favorable, as the juxtaposition of two contrasting cases can generate a theory that is both more holistic and more robust (Eisenhardt, 1989, p. 537).

Case studies can follow a holistic or an embedded design approach. A holistic approach is favorable when the unit of analysis is holistic in itself and no logical subunits can be defined. However, in certain situations, the holistic approach risks the generation of overly abstract or general results. In such cases, an embedded design including several sub-cases is useful (Yin, 2009, p. 50). As in this dissertation, the primary unit of analysis – a software product or platform – is holistic in itself, the study primarily follows a holistic case design. To summarize, the qualitative research design applied in this dissertation can be described as a holistic multi-case approach, as illustrated in Figure 11.

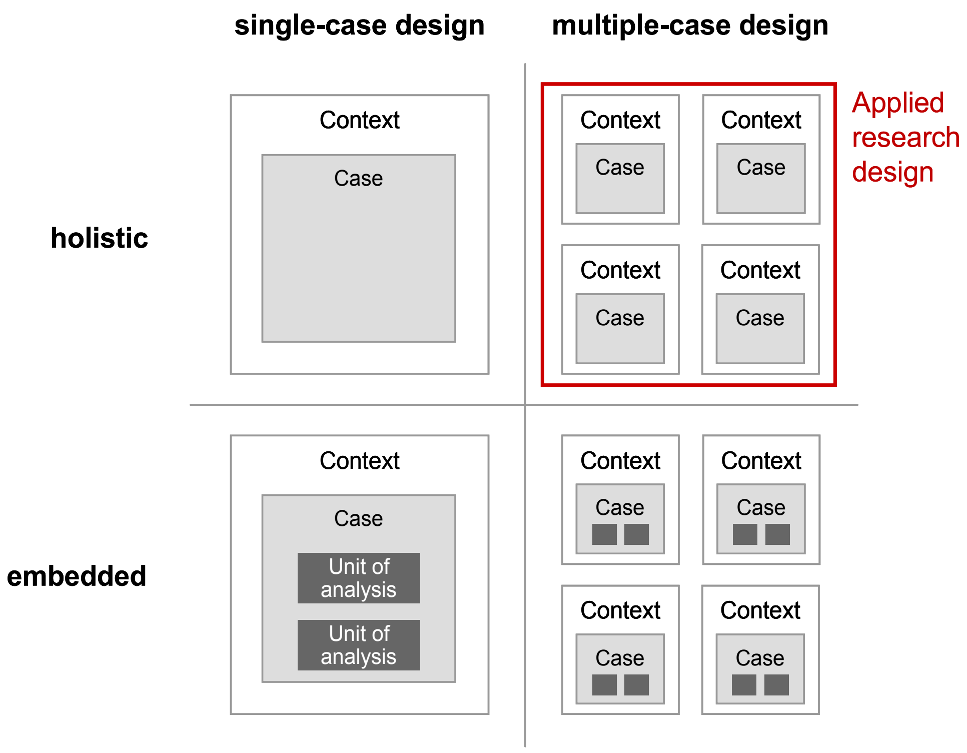


Figure 11 – Case study research design (based on Yin, 2009, p. 46)

With the research design utilized here, the cases are selected based on the specification of the respective research context. For this dissertation, the case study context for software product cases (research topic one, Section 4) and software platform cases (research topic two, Section 5) differs.

Within each of these sub-contexts, incoming IP modularity and outgoing IP modularity cases differ (see Section 2.3). With regard to incoming IP modularity, there is no differentiation between software product and software platform.

As for outgoing IP modularity, the cases differ with regard to the respective vendor's IP strategy. For this dissertation, an IP strategy is defined as restrictive when the product or platform is opened to customers or complementors with a strictly defined API. In contrast, an IP strategy is considered open when additional IP is provided by the platform vendor. This IP can be source code access based on special licensing conditions (West, 2003, p. 1276) or the provision of additional insights into internal data structures. This classification is important with regard to the selection of polar cases, which is recommended by Pettigrew (2010, pp. 274–276) for comparative case studies.

However, it must be mentioned that a gray-scale picture of product/platform openness is transformed into a black and white one. By their nature, software platforms in particular cannot follow an excessively restrictive IP strategy but also cannot be completely open, which would result in total surrender of a platform vendor's control (Boudreau, 2010, p. 1850). Figure 12 displays the resulting matrix and the relevant area for case selection.

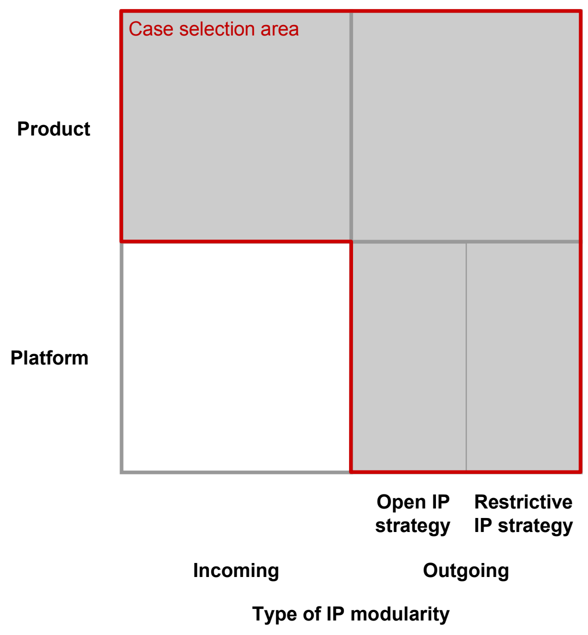


Figure 12 – Generic case selection matrix

During concrete case selection, we follow a structured approach to identify the most promising cases. First, a long list of twelve cases from five companies that experience IP modularity was generated (see Table 1).

No.	Case	Company	Importance of IP modularity	Type	IP strategy
1	Linux graphic drivers	Suse Linux GmbH (part of Attachemate Corp.)	medium	product	open
2	Engineering software	- confidential -	high	product	restrictive
3	Embedded Linux	- confidential -	medium	product	open
4	Data management software	- confidential -	high	product	open
5	Collaboration software	- confidential -	medium	product	open
6	Visualization software	- confidential -	medium	product	restrictive
7	Visualization toolkit	- confidential -	low	product	restrictive
8	Storage technology	- confidential -	low	product	open
9	SAP NetWeaver PI	SAP AG	high	platform	restrictive
10	SAP Business ByDesign	SAP AG	medium	platform	restrictive
11	SugarCRM	SugarCRM Inc.	high	platform	open
12	Force.com	Salesforce.com Inc.	N/A	platform	restrictive

Table 1 – Long list of possible research cases

In any case study, the disclosure of as much as possible case-specific information is most desirable, as the reader's trust in the case results can thereby be increased (Yin, 2009, p. 181). In this dissertation, we face the challenge of being unable to name one company for confidentiality reasons. This company is a multinational technology company with significant software businesses in several business units.

This anonymization does not limit the quality of our results because only the names of individuals and specific products are affected, but all case-relevant data can be used without any restrictions. Furthermore, the agreed-to confidentiality allowed the company representatives to be more open and share additional internal documents.

Based on initial consultations with our company contacts, basic case descriptions were developed and the importance of IP modular design for product or platform success was rated by the researcher team in cooperation with the company representatives. Therefore, we identified the strategic performance indicators for the specific product or platform. Then, we discussed the impact of IP modular design on the indicators with the company representatives (see questions 14 to 16 in Figure 14). We rated importance as high when the respondents categorized IP modular design as a vital precondition for product/platform success.

From the initial long list of cases (see Table 1), the final selection of concrete cases is aimed toward the identification of cases that cover both incoming and outgoing IP modularity for software products and follow different IP strategies for software platforms.

Finally, those cases on the long list in which the importance of an IP modular strategy is rated highest were selected. Figure 13 provides the cases that were selected based on this logic.

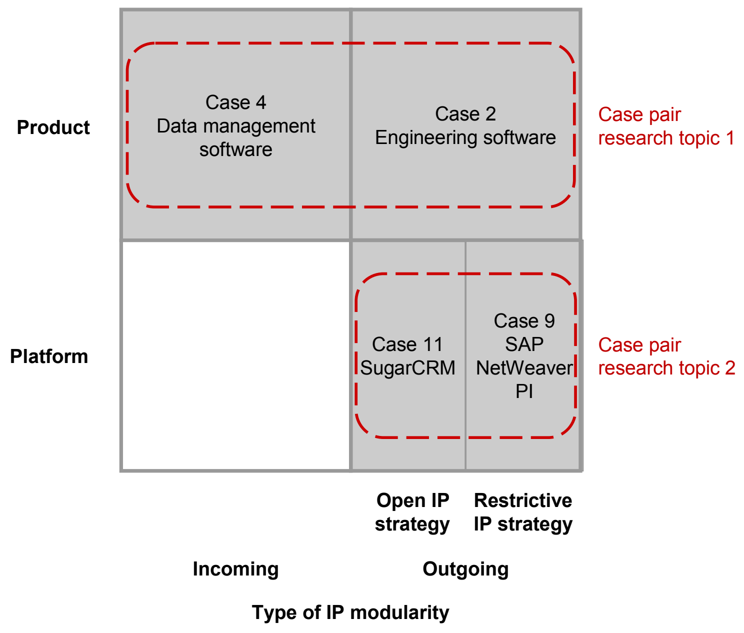


Figure 13 – Final case selection matrix

There is no special case for incoming IP modularity within the platform case, as it does not differ from the product case in terms of its characteristics.

The case selection process results in a total of four selected cases, which is viewed as sufficient for the generation of valid theory (Eisenhardt, 1989, p. 545). As this section aims to focus on a general introduction to the applied case study research methodology, the selected cases will not be described in detail here. Extensive descriptions of the selected cases can be found in Sections 4, 5 and 6, which address the case study results.

Interview design

As interviews provide the primary data source for case study research, thorough preparation is essential. Therefore, a set of guiding questions is defined with the purpose of maintaining the investigator's focus during the case interview (Yin, 2009, p. 86). Unlike in quantitative research, these questions act as guides for the case interviews and do not have to be answered in a linear manner. Furthermore, one of the primary benefits of case study research is that unexpected findings may arise from semi-structured case interviews.

There is no clear rule regarding the level of detail of these guiding questions. Often, a qualitative inquiry begins with a set of initial research questions and is then extended during the course of the research:

> *"Although early identification of the research question and possible constructs is helpful, it is equally important to recognize that both are tentative in this type of research. No construct is guaranteed a place in the resultant theory, no matter how well it is measured." (Eisenhardt, 1989, p. 536)*

For this study, two interview guidelines are used, one for software product and platform providers (research topics one and two) and one for providers of complementary applications (research topic three).

As shown in Figure 14, the interview guideline for software product and platform providers is based on the initial research framework displayed in Figure 10 and includes additional questions on the demographics and the role of the interview partner.

Introduction	1.	Interview setup
	2.	Interviewee details: academic background, job project title & function, job with regard to case
	3.	Brief description of our research goals
C IP modularity	4.	Validate and review prepared case basic description
	5.	What role did IP play as a criterion for modularization?
	6.	Was there a trade-off between IP modularity and other types of modularization?
	7.	Did the level of IP modularity that was established in the product imply higher cost and/or technical drawbacks?
	8.	Who brought forward arguments related to IP in the process of designing the modular architecture?
A Intended effects	9.	What are the intended effects of IP modularity in this particular case?
	10.	How could the effects be ranked in importance?
	11.	Which strategic advantages were in this case obtained from IP Modularity?
B Other effects	12.	In addition to IP what were other drivers leading to this particular modularization?
	13.	How are IP considerations and other drivers for this particular modularization interrelated?
D Real effects	14.	What are the performance indicators of this particular product/technology/platform?
	15.	Did the particular (re-) modularization affect product/technology/platform performance?
	16.	How did the particular (re-) modularization prevent risks for this product/technology/platform?
Wrap-up	17.	What are other factors related to IP Modularity in this particular product/technology/platform?
	18.	Which other stakeholders were involved?
	19.	Debriefing

Figure 14 – Interview guideline for software product and platform providers

The interview guideline for ecosystem partners, as shown in Figure 15, is based on the four guiding propositions for this qualitative research project, which are aimed toward validation of the initial conceptual findings regarding outgoing IP modularity arrived at by Henkel and Baldwin (Henkel and Baldwin, 2010).

Introduction	1.	Interview setup
	2.	Interviewee details: academic background, job project title & function, job with regard to case
	3.	Brief description of our research goals
Product	4.	How did your product evolve and why did you choose your platform provider?
	5.	How does your interaction with the platform look like? *(Proposition 1)*
	6.	How does your current interface/contract with the platform provider look like *(Proposition 2)*
	7.	How complex is your complementary product? *(Proposition 3)*
	8.	How does your product extent/adapt the platform? *(Proposition 4)*
	9.	What is your offering for other platforms?
Platform & ecosystem	10.	What are the intended effects of IP modularity in this particular case?
	11.	How could the effects be ranked in importance?
	12.	Which strategic advantages were in this case obtained from IP modularity?
Wrap-up	13.	What is your overall perspective on the ecosystem management of your platform provider?
	14.	Which other stakeholders were involved?
	15.	Debriefing

Figure 15 – Interview guideline for ecosystem partners

Figure 14 and Figure 15 present basic interview guidelines. To enable structured text analysis (see Section 3.2.3), these guidelines[2] have not been altered significantly throughout the research process. However, minor adaptations have been implemented to incorporate first findings for the benefit of the study as a whole, a method noted by Yin (2009, p. 90):

> *"... the existence of such an outline should not imply rigid adherence to a predesigned protocol. In fact, case study plans can change as a result of the initial data collection, and you are encouraged to consider these flexibilities – if used properly and without bias – to be an advantage of the case study method ." (Yin, 2009, p. 90)*

The interview guideline leads us through a case interview in terms of content. Overly strict adherence to the given structure might ease the interview analysis, but valuable information could be lost by disrupting the interview partner's current thought formation. During the interviews, we had to find the appropriate balance between focus and openness to maximize the results. To mitigate this risk, both interview guidelines have been pre-tested in simulated interview situations with a second researcher.

3.2.2 Case interviews

After the case interviews are set up, data generation is prepared from both a content and an execution perspective: First, the interviewees are identified based on their roles and importance to the case. Second, the interviews are scheduled with the interviewees. Third, the interviews are executed and transcribed. Fourth and finally, each interviewee is asked for additional data that support the discussed topics and opinions. In addition to the data provided by the interviewees, publicly available documents such as internal documents, analyst reports and findings from fellow researchers are gathered.

[2] Some scholars refer to the interview guideline as an interview outline. For this dissertation, these terms are used synonymously.

Interviewee identification

We begin our case analysis for each case with a series of pre-interviews with our *primary contact persons* in each respective case. The result is a basic case description (see Section 3.2.1) and a list of possible additional interviewees. This list is generated with the assistance of the primary contact person for the respective case and based on a predefined list of roles in a typical software development organization. Table 2 lists the basic role descriptions used for the pre-interviews.

Role	Description and responsibilities
General manager	Responsible for overal product performance
R&D manager	Responsible for R&D budget
Staff function manager	Supports responsible management using special analysis Typical functions: strategy, business development,..
Software architect	Responsible for architecture decisions for specific product
Developer	Involved in product development
Legal expert	Expert involved in all IP-related issues
Sales manager	Responsible for specific product sales
Ecosystem manager	Manages partner ecosystem

Table 2 – Interviewee role description

The primary contact person then identified possible interview partners based on the role descriptions. In addition, at the end of each interview, all interviewees were asked to name additional stakeholders (question 18 in Figure 14 and question 14 in Figure 15).

Interview execution and transcription

A total of 30 interviews were conducted for the four cases between May 2011 and September 2012. The majority of these interviews were conducted with general managers and R&D managers. All interviewees' job descriptions fit to one of the roles provided in Table 2, but not all roles apply to each case.

Overall, more than 27 hours of interviews, which include 13 in-person and 17 phone interviews, form a solid empirical basis for the case study analysis in this dissertation.

The aim was to conduct each interview as a personal interviews because such a method allowed us to gather additional information, such as field notes (Eisenhardt, 1989, p. 538) and direct observations (Yin, 2009, pp. 101–114) . Therefore, we made a significant effort to conduct in-person interviews in five locations in Germany and in two locations in the United States of America.

To generate a common understanding of the concepts and research objects, we sent interviewees supporting material prior to the actual interview. Table 3 provides an overview of all interviews for the selected cases.

No.	Case	Company	Date	Interviewee role	Type
1	Engineering software	- confidential -	2011-07-19	Software architect	Phone
2	Engineering software	- confidential -	2011-08-05	Developer	In-person
3	Engineering software	- confidential -	2011-08-08	R&D manager	In-person
3	Engineering software	- confidential -	2011-08-08	General manager	In-person
4	Engineering software	- confidential -	2011-08-26	R&D manager	In-person
5	Engineering software	- confidential -	2011-08-31	Software architect	In-person
6	Engineering software	- confidential -	2011-09-01	R&D manager	In-person
7	Data management software	- confidential -	2011-05-03	R&D manager	In-person
8	Data management software	- confidential -	2011-07-28	Developer	Phone
9	Data management software	- confidential -	2011-07-29	R&D manager	Phone
10	Data management software	- confidential -	2012-05-24	General manager	Phone
11	SugarCRM	SugarCRM Inc.	2011-08-12	General manager	Phone
12	SugarCRM	SugarCRM Inc.	2012-08-25	Sales manager	In-person
13	SugarCRM	SugarCRM Inc.	2012-09-19	General manager	In-person
14	SugarCRM	SugarCRM Inc.	2012-09-19	Legal expert	In-person
15	SugarCRM	SugarCRM Inc.	2012-09-19	R&D manager	In-person
16	SugarCRM	SugarCRM Inc.	2012-09-19	Staff function manager	In-person
17	SugarCRM	SugarCRM Inc.	2012-09-23	General manager	In-person
18	SugarCRM	KINAMU AG	2011-12-07	General manager	Phone
19	SugarCRM	Insignio CRM GmbH	2011-12-12	General manager	Phone
20	SugarCRM	MyCRM GmbH	2011-12-16	General manager	Phone
21	SugarCRM	SugarCRM Inc.	2012-01-24	R&D manager	Phone
22	SugarCRM	SugarCRM Inc.	2012-03-01	Staff function manager	Phone
23	SAP	SAP AG	2011-08-22	Staff function manager	Phone
24	SAP	SAP AG	2011-09-01	Staff function manager	Phone
25	SAP	SAP AG	2011-10-06	Software architect	Phone
26	SAP	SAP AG	2011-11-09	Software architect	In-person
27	SAP	Information Builders Corp.	2012-03-09	Software architect	Phone
28	SAP	SAP AG	2012-03-09	Ecosystem manager	Phone
29	SAP	SAP AG	2012-03-13	General manager	Phone
30	SAP	Itelligencene AG	2012-03-14	Software architect	Phone

Table 3 – List of case interviews

The interviews were conducted primarily in the mother tongues of the interviewees. To prevent case analysis bias, the interviews have not been translated into one language. The two interviews that could not be recorded because of confidentiality reasons were summarized by the interviewer(s) in interview write-ups and aligned with the interviewees. For confidentiality reasons, all person, company, product and location-based data in the interview transcripts for Case 2 and Case 4 were anonymized. In total, 313 pages of interview transcripts and write-ups were generated for the case database.

Secondary data sources

In addition to the interview transcripts, our analysis is based on data such as official presentations and internal documents. Therefore, each interviewee was asked to share additional documents. Additionally, external data sources such as conference presentations and analyst reports are added to the case database (see Table 4).

No.	Case	Additional information	Type
1	Engineering software	List of components affected by IP modularization	Internal document
2	Engineering software	Architecture introduction	Internal document
3	Data management software	The use of free and open source software	Conference presentation
4	Data management software	Third-party software adoption and management process	Internal document
5	SugarCRM	Dow Jones company report 2011	Analyst report
6	SugarCRM	The Forrester Wave™: CRM Suites For Midsized Organizations	Analyst report
7	SAP NetWeaver PI	List of API components	Internal document

Table 4 – Secondary data sources

This rich set of secondary data sources enables the use of a data triangulation approach during the analysis phase, increasing the construct validity of the findings (Yin, 2009, p. 116).

3.2.3 Case analysis

The analysis of case data is the most important and difficult step in generating theory from case study research. In this dissertation, we derive our findings from multiple analyses: First, the categories obtained from the iterative coding process, second, the results of the within-case analysis and, third, the results of the cross-case comparison.

The most important criterion for high-quality case analysis is the close linkage of data and conclusions, which can be assured with a detailed documentation of all analysis steps (Eisenhardt, 1989, p. 539) (see Figure 9), as provided in the following paragraphs.

Definition of initial coding scheme

As suggested by Miles and Huberman (1994, p. 58), the initial coding scheme is derived from the interview guidelines (see Figure 14 and Figure 15), which are based on the basic research design. Thus, the coding scheme, the interview guideline and the research framework are aligned so that the questions for research topics one to three can be answered:

Research framework	Root node	Interview question
C - IP modularity in research case	Q04 - Case review and validation	Validate and review prepared case description
	Q05 - IP as modularization criterion	What role did IP play as a criterion for modularization?
	Q06 - Modularization trade-offs	Was there a trade-off between IP modularity and other types of modularization? If so, in what respect?
	Q07 - Costs and drawbacks	Did the level of IP modularity that was established in the product imply higher cost and/or technical drawbacks?
	Q08 - Stakeholders and decision power	Who brought forward arguments related to IP in the process of designing the modular architecture? And - if this was the outcome - who finally decided to base modularization strongly on IP considerations?
A - Intended effects	Q09 - Intended effects	What are the intended effects of IP modularity in this case?
	Q10 - Effects ranking	How could these effects be ranked in importance?
	Q11 - Strategic advantages	Which strategic advantages were in this case obtained from IP modularity?
B - Other effects	Q12 - Non-IP modularization drivers	In addition to IP what were other drivers leading to this particular modularization?
	Q13 - Interrelations IP- and other drivers	How are IP considerations and other drivers for this particular modularization interrelated?
D - Real effects	Q14 - Platform performance indicators	What are the performance indicators of this particular product/technology/platform ?
	Q15 - Modularization impact on performance	Did the particular (re-)modularization affect product/ technology/platform performance? If so, how strongly?
	Q16 - Modularization impact on risk mitigation	How did the particular (re-)modularization prevent risk for this product/technology/platform? If yes, how strong?

Table 5 – Initial coding scheme

This approach results in logical consistency throughout the entire research process and increases the construct validity of this qualitative research project.

Interview coding

In the course of case study research, massive amounts of data are collected, and addressing this complexity is the key challenge (Miles and Huberman, 1994, p. 56). The risk of being overwhelmed by the mass and complexity of data is inherent, as Pettigrew (2010) describes:

> *"The result is death by data asphyxiation – the slow and inexorable sinking into the swimming pool which started so cool, clear and inviting and now has become a clinging mass of maple syrup."* (Pettigrew, 2010)

Structured coding has become a best practice for mitigating this risk (Bansal and Corley, 2011, p. 234). Coding is the process of attaching units of meaning (codes) to portions of the original data such as interview transcripts. Coding reduces complexity and maintains the context of the single parts (Miles and Huberman, 1994, p. 56). The information portions can be of different sizes, from full paragraphs to single words. In addition, a coding scheme, consisting of different categories, is required for the categorization of the various portions of information and that the researcher is to be able to quickly find the portions relating to a specific research question (Miles and Huberman, 1994, p. 56).

As the interviews typically do not strictly follow the interview guideline, there is a degree of overlap among the answers given. Thus, multi-coding of certain parts of the original data is required.

To ease the coding process, the standard software package NVivo from QSR International Pty Ltd. has been used to support the iterative coding process (Yin, 2009, p. 128). In this software, categories are referred to as nodes. Thus, for this dissertation, "category", in terms of the coding scheme, and "node" are used synonymously.

As proposed by Eisenhardt (1989, p. 538), multiple investigators were involved in coding and reviewing to increase confidence in the findings.

Coding refinement

Because the entire qualitative research process is iterative in nature, this initial coding scheme can be adapted throughout the research process. The idea is that the researcher becomes familiar with the case as stand-alone entity (Eisenhardt, 1989, p.

541; Miles and Huberman, 1994, p. 65). Revising the initial coding scheme is a prerequisite for the identification of unexpected findings:

> *"Researchers with start lists know that codes will change; there is more going on out there than our initial frames have dreamed of, and few field researchers are foolish enough to avoid looking for these things." (Miles and Huberman, 1994, p. 63)*

The coding approach applied for this dissertation combines both inductive and deductive elements. In this approach, deductive elements form the basic research framework (see Figure 10), and the case study propositions are derived from prior studies. The coding scheme is extended when new constructs that emerge throughout the coding process. With regard to these new constructs, the data are approached without preconceived propositions, which allows the new theory to emerge from the data in a pure manner (Glaser and Strauss, 2008, p. 46).

The constructs that emerge from the data during this iterative process are the basis for further analysis and key elements of the cross-case comparison.

Within-case analysis

The importance of the within-case analysis is driven by the serious need to reduce the staggering volume of data without losing vital details (Eisenhardt, 1989, p. 540).

The emerging categories, as described in the preceding paragraphs, form the first indications of the identified theoretical constructs. The within-case analysis investigates how the identified constructs are interrelated. A common method of within-case analysis is the use of data displays (data abstractions in matrices) that facilitate the development of plausible reasons for why things are occurring (Miles and Huberman, 1994, p. 90).

Assessment matrix generation

Miles and Huberman (1994, pp. 90–141) provide a variety of assessment matrices (displays) that assist the qualitative researcher during within-case analysis: partly ordered displays, time-ordered displays, role-ordered displays and conceptually-ordered displays.

Based on the specific criteria for each display, we identify the partly ordered checklist matrix as a perfect fit for our research approach:

> *"A checklist matrix is a format for analyzing field data on a major variable or general domain of interest. The basic principle is that the matrix includes several components of a single, coherent variable, though it does not necessarily order the components." (Miles and Huberman, 1994, p. 105)*

Checklist matrices are well suited to the exploration of new domains (Miles and Huberman, 1994, p. 109) – which is the case for our research endeavor on IP modularity – and for new field research in the information systems domain (Lee and Cheung, 2004, p. 7).

We collect comparable data from all key respondents and process it in the specific format of the predefined interview guideline. Looking ahead in the analysis process, the checklist matrix is suitable for within-case analysis and, furthermore, can easily be customized for cross-case analysis purposes (Miles and Huberman, 1994, p. 105).

We combine the checklist matrix with the notion of roles that differ among the interviewees on the list (see Table 2). This dimension is added because in the course of our research, we experienced that the roles of the respondents significantly influence their views on IP modularity.

By adding the notion of roles, we extend the checklist matrix using elements of a role-ordered matrix (Miles and Huberman, 1994, p. 123). The resulting checklist matrix shows the identified categories on the Y-axis. The interviewees and their roles are shown on the X-axis. The cells are populated with an "x" or the evaluations "Low", "Med" or "High." "x" indicates that the category – in this case, a type of intended effect – is mentioned but not evaluated in terms of its importance. The evaluations "Low", "Med" or "High" are based on the concrete statements in the coded quotes. To minimize bias resulting from subjective judgment, all evaluations have been reviewed by a second researcher.

Identification of case-specific conclusions

The overall idea of a detailed within-case analysis is to familiarize oneself with the case and allow the case-specific pattern to emerge prior to the cross-case analysis (Eisenhardt, 1989, p. 540). The assessment matrix and the original data are always closely linked. Thus, the assessment matrix acts as a vehicle for the identification of interrelations between identified constructs, but the original case data are required for an understanding of these interrelations and the generation of valid theory (Miles and Huberman, 1994, p. 101). In our results, we are explicit and demonstrate how these interrelations are identified to ensure case study reliability (Miles and Huberman, 1994, p. 109).

Generating meaning from the sheer mass of case data is the most difficult element of case study research. Unfortunately, this element is also the least structured element (Yin, 2009, p. 127). Miles and Huberman (1994, pp. 245–261) describe 13 tactics for generating insights from data displays (Miles and Huberman, 1994, pp. 245–261) . Yin (2009, pp. 136–160) provides five elementary analytic techniques: pattern matching, explanation building, time series analysis, logic models and cross-case synthesis (Yin, 2009, pp. 136–160) .

These analysis techniques support our search for meaning in the case data. A key challenge is the identification of the appropriate set of techniques for analyzing the data for a specific case. For within-case analysis, Yin (2009, p. 136) identifies the pattern-matching approach as the most useful technique. Additionally, Miles and Huberman (1994, p. 106) explicitly suggest the use of a pattern matching tactic for the analysis of checklist matrices. Patterns are similarities and differences across categories within the given context of an assessment matrix (Miles and Huberman, 1994, p. 246). If the identified similarities and patterns match the predicted ones, solid conclusions can be drawn and, through further analysis, logic models can be generated (Yin, 2009, p. 137).

In our case, the research framework is tested and extended using identified constructs that emerge during the coding process. With this approach, we identify new theoretical constructs and draw our conclusions by reflecting on them in relation to the predefined research framework and the case study propositions (see Section 3.2.1). This approach allows us to draw initial conclusions and generate novel theory from single-case analysis (Yin, 2009, p. 149).

Identification of cross-case patterns

The initial within-case analysis provides us with valid findings and a sound understanding of each case. However, the real power of case-study analysis lies in the comparison of cross-case patterns, as it increases the reliability of the findings and reduces the risk that the findings will be idiosyncratic (Yin, 2009, p. 156; Miles and

Huberman, 1994, p. 172). Eisenhardt (1989, p. 540) frames the rationale for cross-case analysis as follows:

> *"Overall the idea behind these cross-case searching tactics is to force investigators to go beyond initial impressions, especially through the use of structured and diverse lenses on the data." (Eisenhardt, 1989, p. 541)*

We utilize three basic analysis tactics. First, we search for similarities within a certain set of categories. In our case, a similarity could for example exist in terms of the intended effects of IP modularity. Second, we select pairs of cases and list similarities and differences among these cases. Third, we separate the analysis in accordance with type of data source (Eisenhardt, 1989, pp. 540–541) .

For this dissertation, we compare polar case pairs of software products and platforms, as shown in Figure 16.

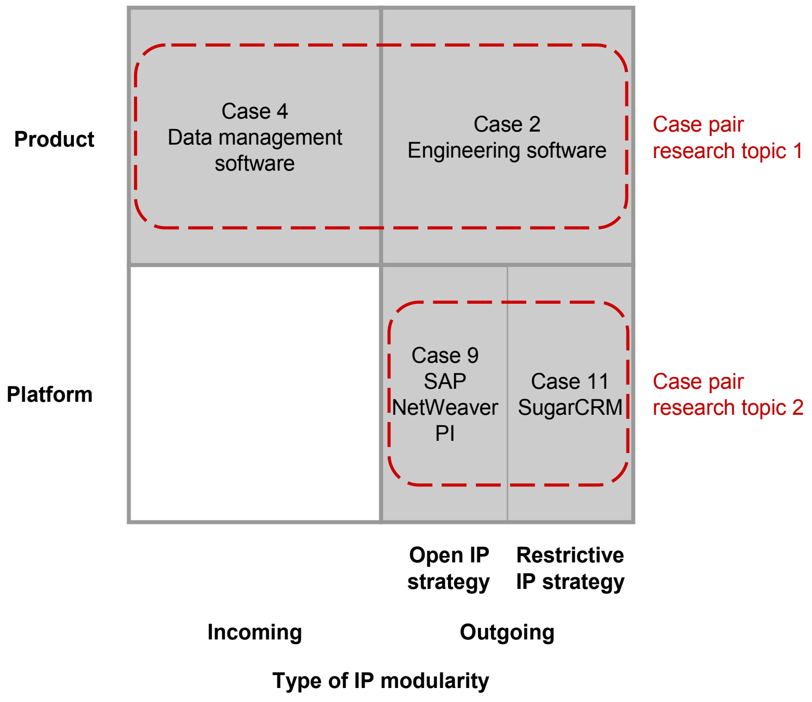


Figure 16 – Case pairs for cross-case analysis

We compare the effects of outgoing IP modularity for an engineering software product (Case 2) with the effects of incoming IP modularity for data management software (Case 4). On the software platform side, we juxtapose SugarCRM with an open IP strategy (Case 11) and SAP NetWeaver PI (Case 9) with a restrictive IP strategy.

The concrete findings obtained using cross-case analysis emerge through the use of the iterative research approach with regard to the concrete case pair and are described in detail in Sections 4 and 5.

Theory development

The overall goal of this qualitative research project is the extension of the existing theory on IP modularity with novel findings derived from empirical evidence. These new findings can be concepts, conceptual frameworks, propositions or mid-range theories (Eisenhardt, 1989, p. 545).

Based on the findings obtained using the within-case and cross-case analysis, we formulate new theory by constantly refining the definition of the identified constructs and compiling evidence across cases (Eisenhardt, 1989, p. 541). The permanent comparison of emerging theory and case data allows us to shape our identified constructs and verify the relationships between them.

We finally aim to generate novel findings and formulate concrete propositions that extend the existing literature on IP modularity.

Verification of findings with existing literature

The research design of this dissertation allows us to extend the existing literature on IP modularity based on within-case and cross-case analysis. The internal validity and generalizability can be further enhanced by linking the results to the existing literature (Eisenhardt, 1989, p. 545).

If the findings are congruent with the extant literature, a higher conceptual level can be achieved. However, conflict between the findings and existing research is also beneficial to the quality of the case-results. In such a case, the researcher is forced to enter a more creative and frame-breaking mode of thinking, which often results in deeper insight into the emergent theory (Eisenhardt, 1989, p. 544).

The comparison of the findings with the existing literature defines the boundaries of generalization of the results of this case study project (Miles and Huberman, 1994, p. 279). For this dissertation, we juxtapose the emerging theory on IP modularity in software products and platforms (Sections 4 and 5) with existing research conducted by Henkel and Baldwin (2010); Baldwin and Henkel (2011).

3.3 Quantitative research

Based on the qualitative findings regarding the impact of IP modularity on software platform ecosystems (see Section 5), we follow our hybrid research approach (see Section 3.1) and conduct a quantitative survey to complement our qualitative findings.

Qualitative methods focus on the verification of established theories through the testing of concrete hypotheses (Creswell, 2003; Mahoney, 2006; Edmondson and McManus, 2007)[3]. Similarly to qualitative research, quantitative research follows a structured research process. For our research endeavor, we apply a three-step approach based on Flick (2011):

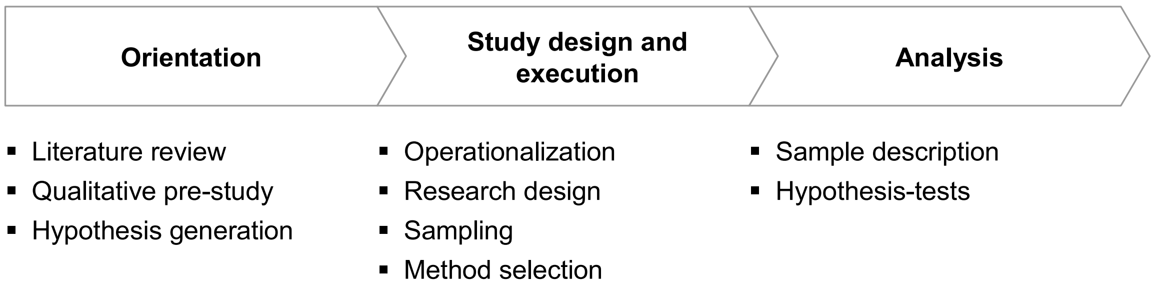


Figure 17 – Quantitative research process

The following sections provide the methodological details of the three phases of the qualitative research process. The detailed results are presented in Section 6.

3.3.1 Orientation

During the orientation phase, the research problem is selected and the relevant scientific literature is reviewed.

Based on the findings regarding the impact of IP modularity on software platform ecosystems discussed in Section 5, the research problem is formulated as follows (named research question in Section 3.2.1):

---

[3] This section builds on the Master thesis of Schreiner Schreiner (2012) that essentially based on the findings of Waltl *et al.* (2012). The primary focus of the work was the extension of qualitative findings on the implications of IP modularity in software platform ecosystems (see Section 5) and the preparation of further quantitative analysis of the implications of IP modularity for the attractiveness of a software platform for providers of complementary software products.

Research problem: *How does an IP modular platform architecture influence the attractiveness of a software platform from a complementor's perspective?*

A qualitative pre-study, conducted as described in Section 3.2, in combination with a thorough review of the literature on modularity (see Section 2.2) and platform attractiveness (see Section 6.1, Figure 46), yields a model describing the factors influencing the attractiveness of a software platform:

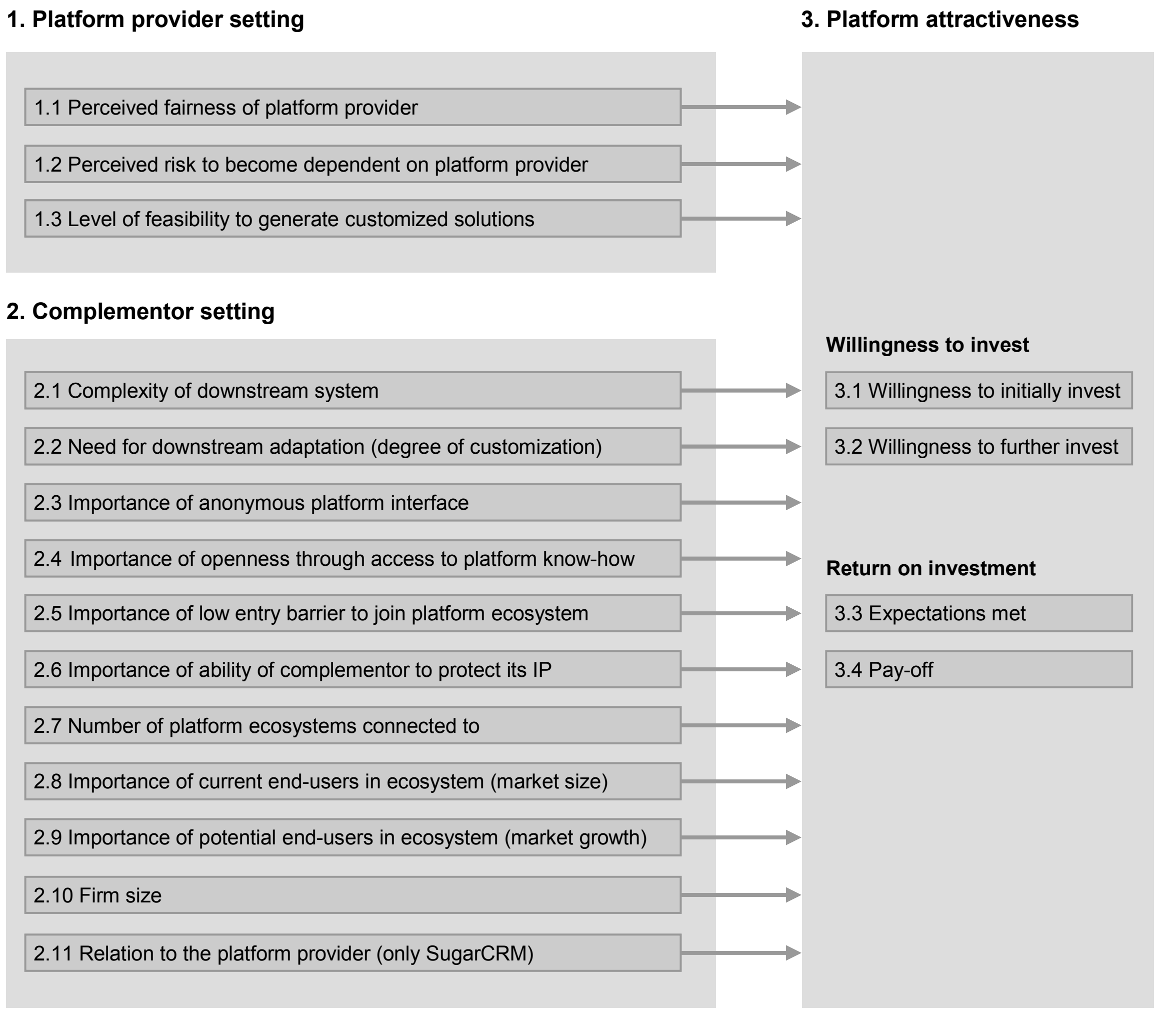

1. Platform provider setting
1.1 Perceived fairness of platform provider
1.2 Perceived risk to become dependent on platform provider
1.3 Level of feasibility to generate customized solutions
2. Complementor setting
2.1 Complexity of downstream system
2.2 Need for downstream adaptation (degree of customization)
2.3 Importance of anonymous platform interface
2.4 Importance of openness through access to platform know-how
2.5 Importance of low entry barrier to join platform ecosystem
2.6 Importance of ability of complementor to protect its IP
2.7 Number of platform ecosystems connected to
2.8 Importance of current end-users in ecosystem (market size)
2.9 Importance of potential end-users in ecosystem (market growth)
2.10 Firm size
2.11 Relation to the platform provider (only SugarCRM)
3. Platform attractiveness
Willingness to invest
3.1 Willingness to initially invest
3.2 Willingness to further invest
Return on investment
3.3 Expectations met
3.4 Pay-off

Figure 18 – Platform attractiveness model

In accordance with the results of the literature review and the qualitative pre-study, three research hypotheses are formulated (please refer to Section 6.2 for further details on the hypotheses formulation):

Hypothesis 1$_A$: *We expect the coefficients of variable 2.1 Complexity of downstream system to be larger for SugarCRM than for Salesforce.com.*

Hypothesis 1$_B$: *We expect the coefficients of variable 2.2 Need for downstream adaptation to be larger for SugarCRM than for Salesforce.com.*

Hypothesis 2: *We expect the coefficient of variable 2.3 Importance of anonymous platform interface to be larger for SugarCRM than for Salesforce.com.*

The platform attractiveness model and the formulated hypotheses provide the orientation for crafting the quantitative study, as described in the following sections.

3.3.2 Study design and execution

The study design and execution phase consists of four steps: *operationalization, research design, sampling* and *method selection.* These steps will be explained briefly in this section. For further details, please refer to the work of Schreiner (2012).

To test the research hypotheses, the underlying research model must be *operationalized*. In operationalization, the proposed relationships are translated into entities that can be tested (Flick, 2011, p. 49). These entities, or variables, are divided into two basic groups: *dependent variables* and *explanatory* variables.

Explanatory variables, also named *independent variables,* most likely influence the *dependent variables* (Creswell, 2003, p. 94). To test our hypotheses on platform attractiveness, a single indicator is generated from the set of variables using the Cronbach's alpha coefficient (Cronbach, 1951).

During the *sampling* phase, the population and the related sample are specified. For our research, the impact of IP modularity on software platform attractiveness, the population is defined as follows:

Population: *Companies that develop complementary goods for CRM software platforms.*

We apply a non-random process to draw a clustered sample from this population, as we aim to compare the two Customer Relationship Management (CRM) software ecosystems Salesforce.com and SugarCRM (see Section 6).

For the Salesforce.com ecosystem, we address the entire population because all complementors are listed on the ecosystem website. For SugarCRM, not all complementors are known to the platform provider because of the possibility of anonymous co-creation (see Section 6.2).

During the method selection phase, the sample generation is specified. For our research, we use an online questionnaire, as it is the only method of addressing our global target group using reasonable effort. Critics claim that with online questionnaires, only internet-affine participants respond and that there is little incentive to respond. Because our target group consists of managers in the software industry, it is obvious that the former criticism does not apply. To mitigate the second point, we utilized a raffle on original Oktoberfest Beer Steins. Feedback from our survey participants revealed that this incentive was effective in increasing survey participation. The survey was implemented using the online tool Unipark[4].

Based on the developed platform attractiveness model provided in Figure 18, the concrete measurement items and the survey questions are generated based on existing literature on the operationalization of theoretical constructs (Zaheer *et al.*, 1998; Steensma and Corley, 2000; Sako and Helper, 1998).

The questions used with Salesforce.com partners and with SugarCRM partners differ, as necessitated by the peculiarities of each ecosystem, and were finalized after a pretest given to seven Salesforce.com complementors and five SugarCRM complementors. For the final survey, the Salesforce.com complementors were contacted via email based on the contact details provided on the appexchange.salesforce.com platform. SugarCRM's ecosystem manager posted a link on the developer portal sugarforge.com to be used to reach SugarCRM complementors.

In total, we received 126 answers (87 for Salesforce.com, 39 for SugarCRM) between June 4, 2012 and Aug 16, 2012.

[4] See www.unipark.info

3.3.3 Analysis

Opposed to qualitative research, the analysis of quantitative survey data is well structured and defined. This analysis essentially consists of two steps. First, the sample is described by various parameters and, second, the defined hypotheses are tested using statistical tests.

The *sample description* aims to provide an overview of types of survey respondents, as shown in Figure 19.

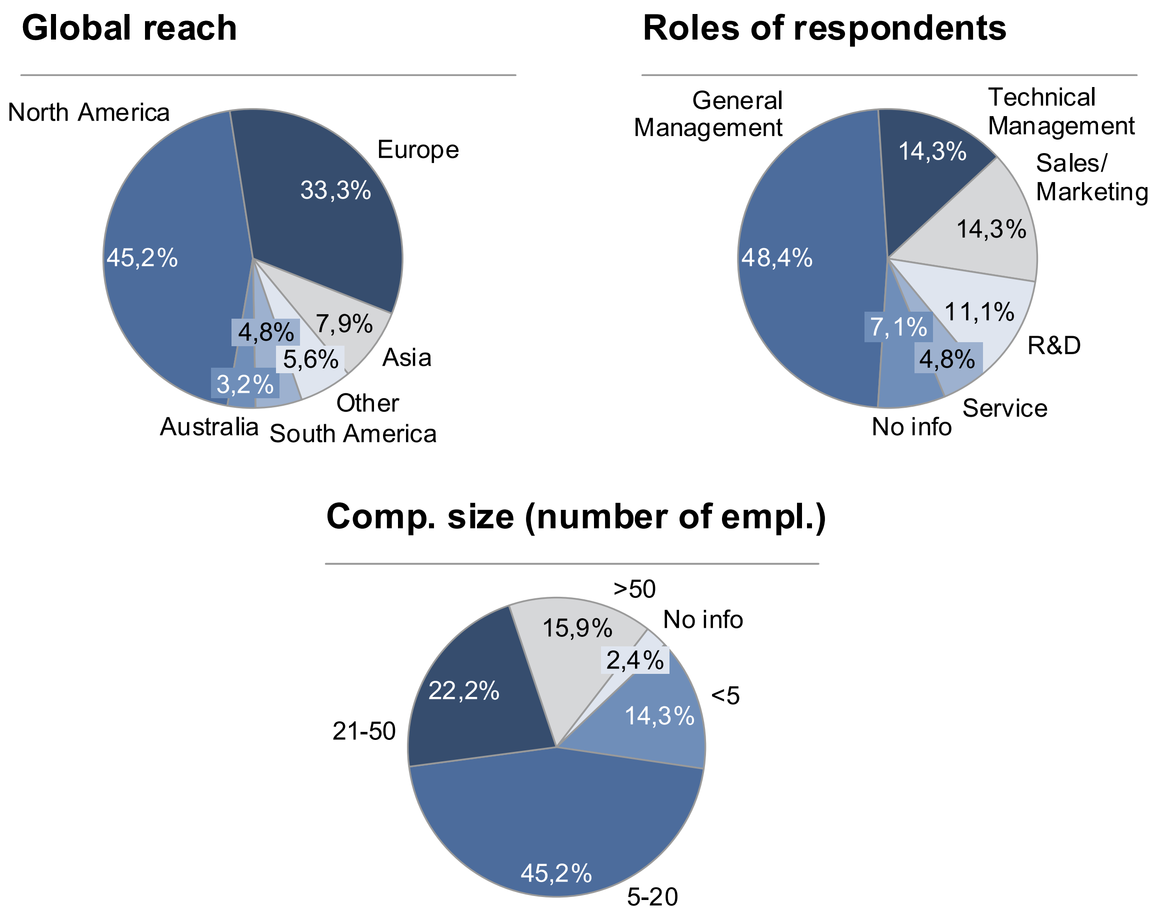


Figure 19 – Sample description

Following the general description of survey participant characteristics, statistics regarding the results for the dependent and independent variables are provided. The overall aim is to provide an adequate overview of the sampled dataset and produce initial findings. All detailed results are presented in Section 6.2.

Through a basic understanding of the dataset, the *hypotheses-tests* are prepared. For this dissertation, we test the hypotheses using a linear regression model. All detailed results are presented in Section 6.

The overall findings are then aligned with the expected findings obtained from the initial qualitative research, as shown in Figure 8, allowing us to formulate the final conclusions.

3.4 Conclusion

This section provides the methodological foundation of the research conducted for this dissertation. We apply a hybrid (qualitative and quantitative) approach to extent current evidence on IP modularity in the software industry domain.

As the clear focus of this dissertation is on qualitative findings, the qualitative research process is described with a high level of granularity. With regard to the quantitative research methodology, we provide the basics of the data gathering and research process based on the work of Schreiner (2012). The primary focus of this dissertation is the analysis phase, as presented in Section 6.2.

4 IP modularity in software products

In this section, we analyze research topic one (see Section 3.2.1), the impact of IP modular design on software products, using the case study approach as outlined in Section 3.2. The empirical analysis is based on the preselected engineering software (Case 2) for outgoing IP modularity and the data management software (Case 4) for incoming IP modularity (see Figure 13). This section aims to answer two basic research questions (see Section 3.2.1):

- Why are software products modularized with regards to IP considerations?
- How do the intended effects of IP modular product design relate to the real effects?

To introduce the topic, we first review the challenges in software product design and provide the basics of the software product business models. Then, in Sections 4.2 and 4.3, we present the results of the within-case analysis. Following that, in Section 4.4, a cross-case analysis allows us to identify the effects of IP modularity for the researched software products. In this section, we also review how consistent our findings are with the existing literature on IP modularity, software product design and business models.

4.1 Software products – design and business models

To understand the impact of IP modular design on the software products, we must understand what a software product is. In general, a software product is a piece of software that is developed once and sold to many customers. This model differentiates software product companies from IT service companies, which also develop software solutions but only for the use of one specific customer. Kittlaus and Clough (2009); Fricker (2012, p. 55) define a software product as follows:

> *"A software product is a product whose primary component is software. Software is an information good that manifests human know-how in bits and bytes. This characteristic makes a software product special in comparison to other goods." (Kittlaus and Clough, 2009; Fricker, 2012, p. 55)*

This description provides the principal difference between software products and other products. Software products are purely virtual goods. Unlike with physical products, the complexity is not limited by physical rules. The value of a software product is purely generated in the design phase, as the cost of reproduction is negligible. The designers of software systems strive to generate value for users with the provision of solutions to real world problems implemented in software technology.

The existing literature on software system design differentiates between business requirements, functional requirements and non-functional requirements (Wiegers, 2003) as shown in Figure 20.

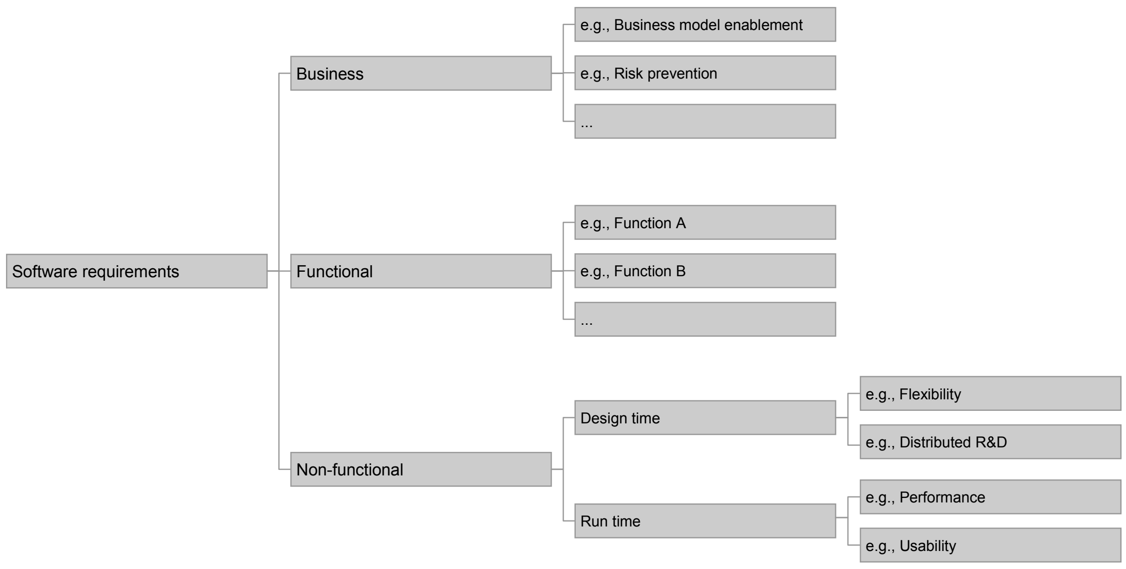


Figure 20 – Software requirements

The business requirements are the benefits that a new system generates to its stakeholders such as sponsors, buyers and users. Therefore, these business requirements are strongly influenced by the software provider's business model. Moreover, the business requirements materially influence other functional and non-functional requirements (Wiegers, 2003). The functional requirements describe the set of functions that a software product must provide to solve real-world problems. The non-functional requirements describe the behavior of a software system. The non-functional requirements (also referred to as the quality requirements) are divided into two basic categories: run time requirements and design time requirements. The run time requirements describe how a software system behaves during its application. This behavior can be related to, for example, the performance or the usability. The design time requirements describe the software system's characteristics during the design process[5] (Chung and do Prado Leite, 2009, p. 369).

The existing literature provides indications that these non-functional requirements and the (modular) design of the software system are closely linked and act as rationales for a certain type of design (Wiegers, 2003; Chung and do Prado Leite, 2009, p. 369). Because we introduce IP as an additional rationale for a system's modular design, this logic would also require the IP to influence the requirement specifications of software systems. To our knowledge, there is currently no empirical evidence that IP influences software requirements.

Business requirements are driven by a business model that describes how a company creates and appropriates value (Weill *et al.*, 2005, p. 5; Popp, 2011, p. 26). Osterwalder (2004, p. 14) describes a business model as follows:

> *"... a business model is a representation of how a company buys and sells goods and services and earns money." (Osterwalder, 2004, p. 14)*

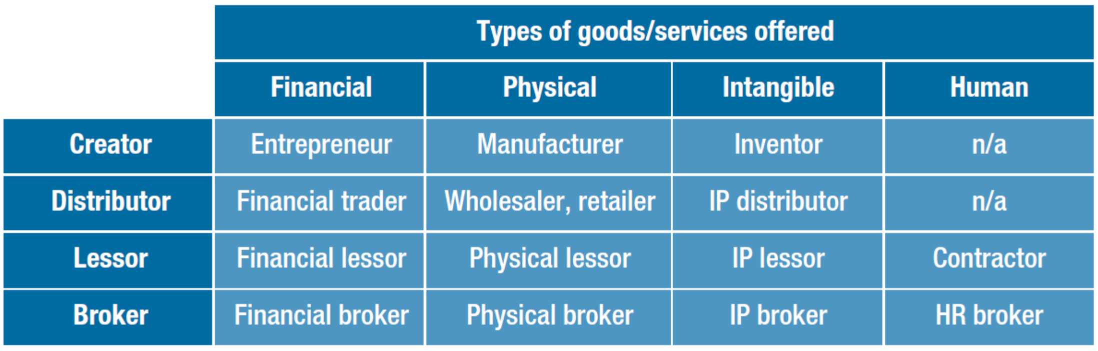
A business model can be described in a two dimensional topology (Weill *et al.*, 2005, p. 7; Popp, 2011, p. 26):

The first dimension differentiates the type of right that is being sold. This leads four basic business models: *creator*, *distributor*, *lessor* and the *broker.* While the creator builds goods from basic material and components, the distributor buys goods and sells them. Finally, the lessor sells the allowance to use a good (but not the good itself) and the broker connects sellers and the buyers (Weill *et al.*, 2005, pp. 8–9; Popp, 2011, p. 26) .

The second dimension distinguishes between the types of assets that are involved which can be: *financial*, *physical*, *intangible* or *human services*.

By combining types of goods and services and basic business model types, Weill *et al.* (2005, p. 10) and Popp (2011, p. 27) identify 14 specific business model types:

[5] Because software products are frequently adapted, the design process is an ongoing process throughout the lifetime of a software product.



	Types of goods/services offered			
	Financial	Physical	Intangible	Human
Creator	Entrepreneur	Manufacturer	Inventor	n/a
Distributor	Financial trader	Wholesaler, retailer	IP distributor	n/a
Lessor	Financial lessor	Physical lessor	IP lessor	Contractor
Broker	Financial broker	Physical broker	IP broker	HR broker

Figure 21 – Business model types (Popp, 2011, p. 27)

Most software product companies[6] focus on the intangible column and implement a hybrid business model where the IP lessor business model type funds the inventor business model type (Popp, 2011, p. 27). The successful implementation of this hybrid business model approach depends on the license under which a software product company allows others to use the created IP. This license describes what the user of a software product is entitled to do with it (Lindman *et al.*, 2011, p. 31). We differentiate between *proprietary*, *free-* and *open source software licenses* (FOSS) (Carver, 2005, p. 453).

In the first case of a traditional proprietary license, the firm charges fees for the use of its software as an IP lessor (Hecker, 1999, p. 48). The conditions are described in the license terms such as the usage conditions or the number of allowed installations. These licenses are individual contracts between the buyer and the seller and are therefore not standardized. It is also typical that the product is provided in a way that hides the source code to protect the IP of the lessor. Thus, a proprietary software license is the easiest way to prevent unwanted leakage of IP (Riehle, 2012, p. 10).

FOSS licenses can be differentiated between *permissive* and *restrictive* (also named copyleft) licenses. Permissive licenses do not require a software company or an individual to reveal the source code for derivative work. The most commonly known permissive licenses are the Massachusetts Institute of Technology (MIT), Berkeley Software Distribution (BSD) and Apache licenses (Lindman *et al.*, 2011, p. 32). Restrictive licenses such as the GNU General Public License (GPL) require the software company or the individual to publish the final product's source code and to license all

[6] Here, we refer to the predominant business of software companies to sell products – Software as a Product (SaaP). However, software companies do also sell consulting services and sell software as web services – Software as a Service (SaaS) Popp (2011).

derivative work under the same license (Lindman *et al.*, 2011, p. 33)[7]. Figure 22 provides and overview of the described software licensing possibilities:

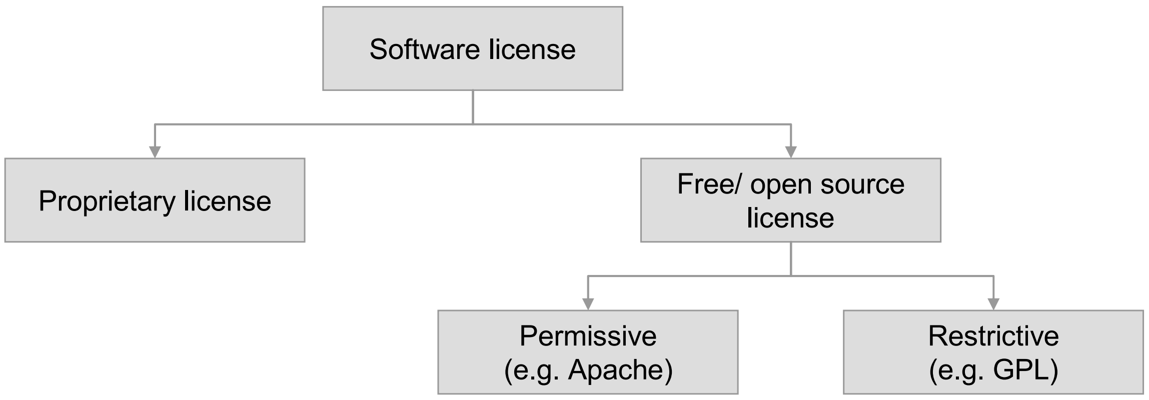


Figure 22 – IP lessor compatible software licenses

If an OSS project is implemented as community open source software based on a restrictive license with code coming from a large number of contributors, the potential for distributed value creation is maximized. The downside is that in this case, a product-based business model is difficult to implement, and the original owner of the code may even lose control over its further development if he does not have the complete copyright for the source code. To overcome this difficulties many companies with a community open source approach generate revenue from OSS related services (Hecker, 1999, p. 49; Bonaccorsi *et al.*, 2006, p. 1085) which corresponds with the contractor business model shown in Figure 21.

In addition, the consideration of the software licenses is of utmost importance when third-party IP is included in a software product. In such cases, the software product company also acts as the IP distributor (see Figure 22). The license conditions of these third-party components may lead to a holdup risk and thus pose a serious threat to the software company's business model in terms of value appropriation. In this section, we also aim to identify whether IP modularity can mitigate this holdup risk (Henkel and Baldwin, 2010) while validating their holdup proposition (see Section 3.2.1):

[7] The GPL is the most restrictive license. Please refer to www.opensource.org for less restrictive FOSS licenses.

Proposition 7: [Holdup Risk] *Incoming IP modularity is advantageous when the focal firm faces the risk of holdup from suppliers of incoming IP. The module boundaries should go between the firm's own IP and the incoming IP.*

Summing up, in this section, we provided a basic software product definition and an overview of the design of software products, software requirements and the related business models. The upcoming sections provide the results from two case studies. First, we research outgoing IP modularity based on an engineering software case (Case 2 as presented in Section 3.2.1). In the second case, we investigate incoming IP modularity based on a data management software case (Case 4 as presented in Section 3.2.1).

4.2 Outgoing IP modularity in software products

To answer the basic research questions on outgoing IP modularity, an example of an engineering software product was selected (Case 2, see Section 3.2.1).

In this case, we follow the modularization decisions of a highly complex engineering software product developed by an international technology company with over 1 billion USD in annual revenue. As outlined in Section 3.2.1, information specific to both the product and individuals must be kept confidential.

The software product at hand is used to configure and program different types of hardware components. Figure 23 shows the basic structure of the engineering software.

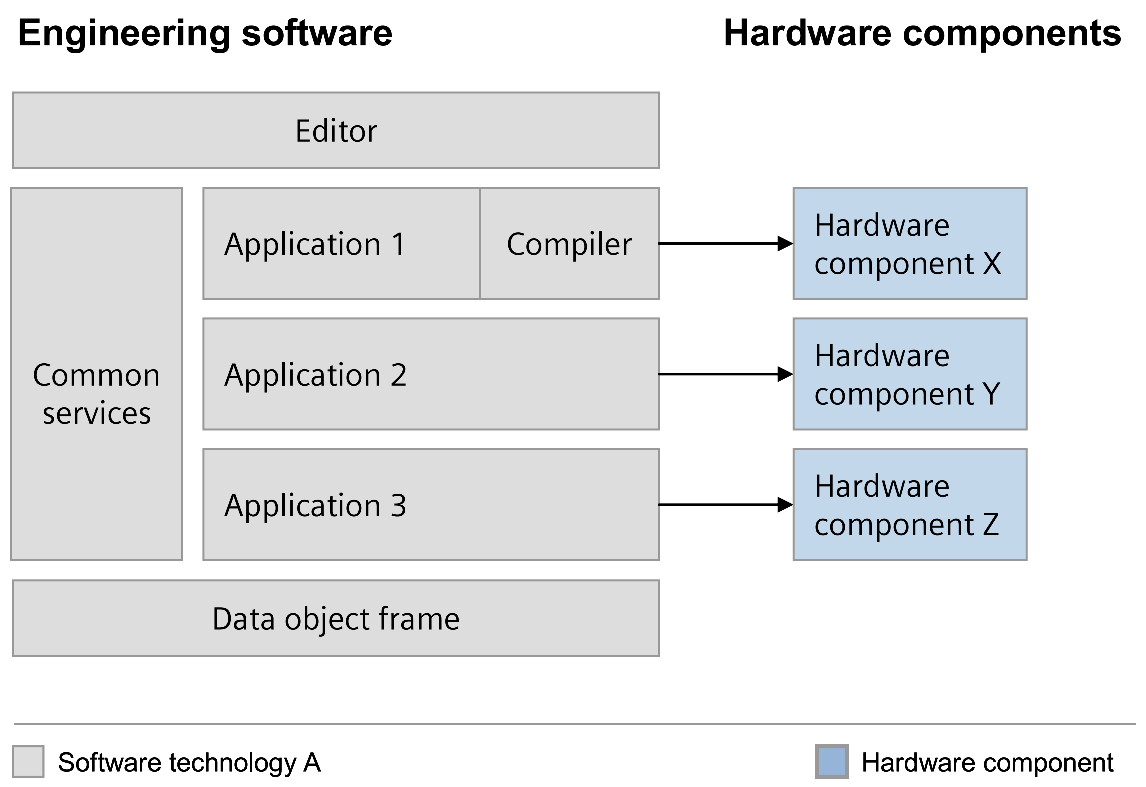


Figure 23 – Schematic structure of the engineering software (Case 2)

The engineering software product consists of several architectural components: the developer workbench, common services, the data object frame and the applications. Each of the applications is designed to configure a specific type of hardware component. For our case, we focus on *Application 1*. This application consists of a core engineering functionality and a compiler module that is required to transfer the configurations to the hardware components.

The revenue model behind this case is based on hardware component sales. The engineering functionality and the software/hardware integration act as key differentiators towards the competitors. The compiler is the most crucial component because it comprises know-how from many years of experience and directly influences the performance of the corresponding hardware components. The compiler is the core IP that under no means is to be disclosed to secure the business model.

The case is a new software development project that succeeds a successful existing engineering software product. In the beginning of this new software development project, a set of architectural studies were conducted to identify the best suited implementation technology. Finally, software technology A was chosen because it offered several benefits, as one of the senior software architects describes:

"The decision for technology A offered the possibility for simpler and faster code implementation, it has better features for testing and error detection and, finally, it was also favored by our developers. Furthermore, well-educated developers can be hired directly from universities." (Software architect, Person G)

During the implementation phase, it turned out that an increase in product performance was required. Therefore, extensive architectural tests were conducted. As additional finding of these tests was that technology A does not provide sufficient protection against reverse engineering. While this protection is not critical for large parts of the system, it does matter for the core IP in the compiler. Based on this realization, the decision was made to re-modularize the compiler into a general part and the compiler core, as shown in Figure 24.

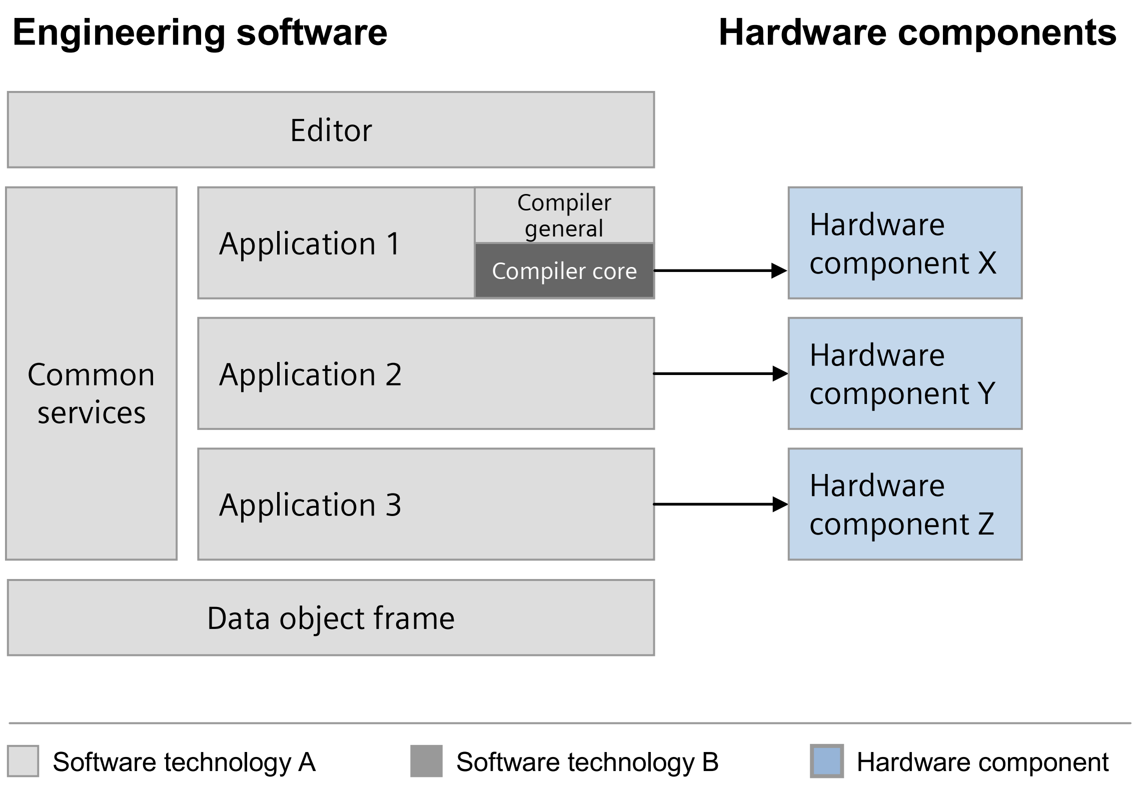


Figure 24 – IP modular engineering software (Case 2)

The general parts of the compiler have been maintained in technology A, while the compiler's core IP was re-implemented in technology B[8], creating additional effort and risk regarding completion. The additional effort and risk were taken into account to protect the compiler core IP and, subsequently, the business model, thus answering our first research question. The analysis of an internal document reveals that a total of 13 out of 23 components were subject to the re-modularization.

We based our findings on interviews with seven key stakeholders, as shown in Table 3. The diverse roles of the respondents ensure that all relevant perspectives on outgoing IP modularity in Case 2 are captured. Our interviewees were three R&D managers, two software architects, one developer and the responsible general manager. In total, we analyzed 71 pages of interview transcripts from approximately seven interview hours and two internal documents, as presented in Table 4. We derive our results from a structured analysis process as outlined in Section 3.2.3.

In the first step of our within-case analysis, the transcribed interview data were coded based on the interview guidelines introduced in Figure 14. The interview guideline itself is based on the core research framework for the effects of IP modularity as shown in Figure 10. Hence, it is ensured that the initial coding scheme, interview questionnaire and research framework are all aligned towards the goal of unraveling the effects of IP modularity. Because we approached the data without any assumptions, additional code categories emerged during the coding process. Therefore, the coding scheme was adapted using an iterative approach and yielding new hierarchical structures, which were then used to unfold the effects in a manner favorable to subsequent interpretation (see Appendix A for the full coding scheme).

In the course of this iterative process, two major elements of the original research framework and their mapping to the interview questions were altered because we noticed an overlap of answers across several questions[9]. First, the category *Intended effects* was renamed as *IP-related effects* because this was the prevailing construct being measured by the actual interview questionnaire (see Figure 14). Second, the category *Other effects* was renamed as *Non-IP-related effects* for the same reason. Figure 25 displays how these adaptations alter the research framework.

[8] Sections of the compiler were already available in technology B because the earlier versions of the engineering software products were implemented in technology B. Consequently, not all parts of the compiler core were newly implemented.

[9] To ensure the openness of the interviewees, the interviews did not strictly follow the guidelines. For the research framework at hand, this easing of the interview structure means having a certain degree of overlap within the answers given and the initial questions.

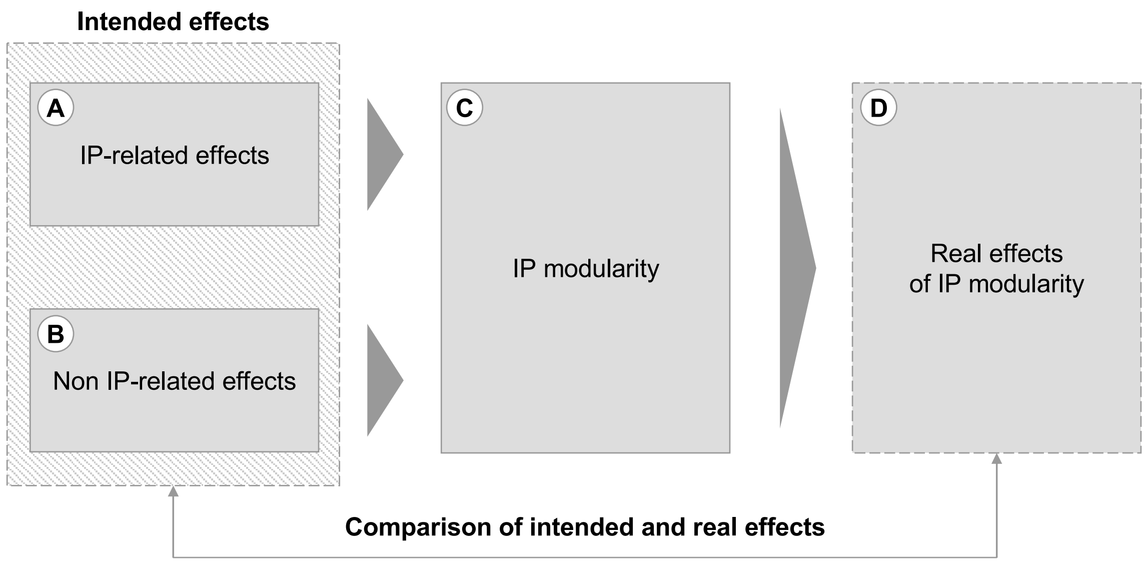


Figure 25 – Adapted research framework

In the following sections, the intended effects and the real effects are described and compared to derive cause and effect propositions. In each of the following sections, we will describe the effects that we identified based on the coding of data and then assess the importance of the effects across different roles with the use of role-ordered checklist matrices (see Section 3.2.3).

We first present the intended effects of IP modularity in the engineering software case. Figure 26 shows all IP-related and non-IP-related effects that were identified by our respondents when answering questions 9-13 (see Appendix A).

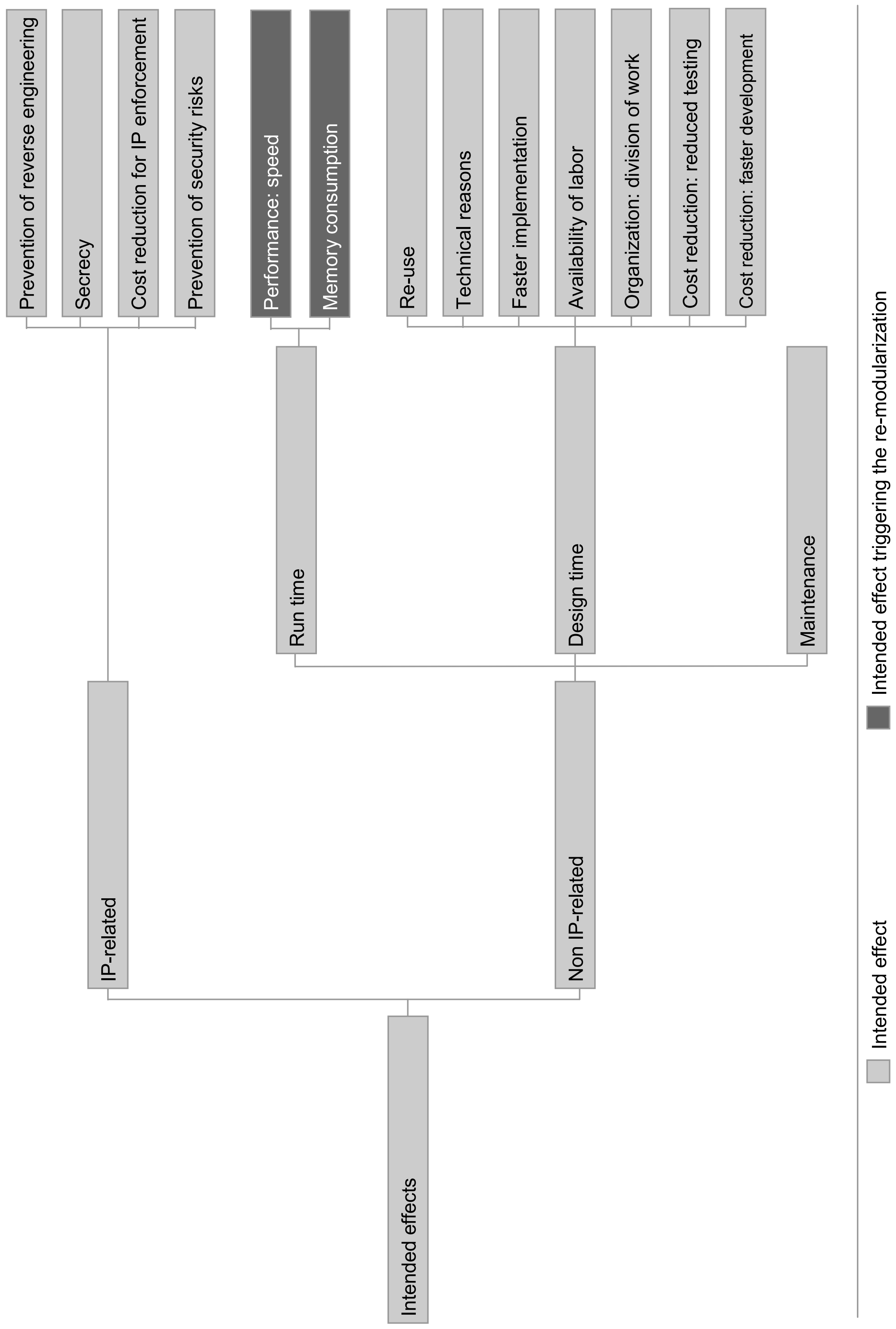

Intended effects
IP-related
Non IP-related
Prevention of reverse engineering
Secrecy
Cost reduction for IP enforcement
Prevention of security risks
Run time
Design time
Maintenance
Performance: speed
Memory consumption
Re-use
Technical reasons
Faster implementation
Availability of labor
Organization: division of work
Cost reduction: reduced testing
Cost reduction: faster development
Intended effect
Intended effect triggering the re-modularization

Figure 26 – Intended effects (Case 2)

For the IP-related effects, we discovered four main effects. As previously described in the basic case description, the *prevention of reverse engineering* was one of the main IP-related effects of this specific modularization. Related to that effect is the *secrecy* of own data structures and data formats from the customers and other third-parties. Another goal of this IP modular architecture is that the secured differentiating IP of the compiler would not be subject to costly *legal IP enforcement* actions. Finally, the *security risk* of third-parties manipulating the system is reduced by the IP modular system.

The non IP-related effects are divided into *run time*, *design time* and *maintenance*. The run time effects *performance: speed* and *memory consumption* were the initial triggers for the re-modularization that finally yielded an IP modular architecture. In the context of IP modularity, it is also of particular interest that in this case, the IP modular design also resulted in a better division of work, which in general may also present a tradeoff with IP modularity marked as distributed R&D in Figure 5. In this case, the module boundaries are also technology boundaries, delineating where different teams work on different parts of the product.

The list of intended effects is the basis for further analysis. With the role-ordered checklist matrix, each row presents one effect. The columns of the matrix represent the interviewees and, hence, separate interviews, which are all based on identical interview guidelines to guarantee consistency. Because we explicitly asked for the importance of the intended effects in question ten (see Figure 14), the importance is indicated with *High*, *Med* and *Low* when the interview data provided enough evidence. Thus, not all intended effects were ranked or put into the perspective of relative importance. Where no ranking could be indicated that the indicated effect had some level of importance, the effects were marked with an *X* as shown in Figure 27.

	Role of interviewee						
	R&D manager	General manager	Software architect	Developer	R&D manager	Software architect	R&D manager
Intended effect	Person A	Person B	Person I	Person H	Person F	Person G	Person E
IP-related							
Prevention of reverse engineering	High	High	Low (Job persp.) High (Corp. persp.)		High	High	High
Secrecy				High			
Cost reduction for IP enforcement						Low	
Prevention of security risks					Low		
Non IP-related							
Run time							
Performance: speed	Med x	Med x	x	Med	Low	Med x	
Memory consumption			High x	x			
Design time							
Re-use	Low x	Low x	x		x	x	x
Technical reasons				Low			
Faster implementation						Med	
Availability of labor						Med	
Organizazion - division of work							Low
Cost reduction: reduced testing						x	
Cost reduction: faster development						x	
Maintenance						x	

Legend:		
High	Rated as highly important effect in several statements	
Med	Rated as important effect in one or several statements	
Low	Rated as less important effect in one or several statements	
x	Effect not included in the importance ranking	

Figure 27 – Intended effects checklist matrix (Case 2)

All of the participants (regardless of their background or role) rate the importance of the IP-related effects as high, whilst the non IP-related effects are frequently regarded as low or medium priority. This ordering of priorities occurs because the IP-related effects pose the greatest threat to the companys' business model as the selected quotes show:

> *"The business strategy is that the core know-how is in the compiler. It is the heart of our offering that has to be protected. [...] Probably IP protection is even more important than performance." (R&D manager, Person F)*

> *"The core IP was re-implemented to prevent de-compiling. So, protection of core IP was the main reason to re-arrange the module boundaries between technology A and technology B." (General Manager, Person B)*

> *"Nevertheless, I'd rate IP protection highest – finally it's about our long-term business." (Developer, Person H)*

For the non IP-related effects, it appears that the run time aspects are considered to be more important than the design time aspects.

> *"Run time performance is the vital criteria for the software to be accepted by our customer base." (Software architect, Person G)*

Interestingly, the IP-related effects came on the table much earlier triggered by an identified performance issue:

> *"For technical reasons, we needed to adapt the initial design. [...] Once we had the new model on paper, we also decided to take the IP topic seriously." (Developer, Person H)*

> *"In this case, IP was not the initial driver for that re-modularization. The main driver was a massive technical problem that brought us close to a total collapse of the project. So we had to act. Thereby, we said: Let's tackle the IP issue in the same effort." (Software architect, Person I)*

Thus, although IP is considered to be highly important, it was not considered in the first modularization phase that was driven by the software architects and developers. The initial trigger was a run time performance and memory consumption problem. Finally the criterion for the product's modular structure was IP protection. This process is manifested by the "dual ranking" by Person I of the IP-related effects. For the personal job, IP protection is not considered to be significantly important – on the contrary, it makes the job even more complicated. Because the software architect was one of the main designers of the initial modularization, IP was not considered as a criterion.

However, the same software architect also states that from a company perspective, IP is most likely the most important criterion.

> *"From a company perspective, ranking of what would probably be IP protection, re-use of code, prevention of source code leakage and performance. From my personal perspective, it's the other way round." (Software architect, Person I)*

Answering the first research question regarding why this software product is made IP modular, we can conclude that the protection of the outgoing IP is the main criterion for the final modularization. Interestingly, the IP protection lies outside of the traditional scope of functional and non-functional requirements engineering (Wiegers, 2003). Furthermore, IP requirements were not associated with business requirements in the engineering process, in spite of their close linkage to the business model.

The problem resulting from this misconception is that IP-related requirements would have started to arise during the engineering process or even after the product launch (ex-post). This delayed attention may lead to time- and cost-intensive re-modularization (LaMantia *et al.*, 2008, p. 8). These expenditures could be reduced (ex-ante) if the IP-related requirements were to become perceived as an integral part of the non-functional requirements.

When asked about the performance indicators of the engineering software, five of the seven interviewees felt that IP protection is a performance indicator of the focal software. This result makes the question "why is IP protection not considered to be a non-functional requirement?" even more relevant. This result reveals that the involved project stakeholders are, in fact, aware of the importance and yet did not take action at an early stage.

This problem may be rooted in a misconception: people are aware of IP-related issues but perceive them to be irrelevant within the initial requirements engineering phase. A change of perception may mitigate the risks of additional time- and cost-expenditures later in the process.

To answer the second research question on the relationship between the intended and the actual effects of IP modularity in software products, we first identify the actual effects and then contrast them with the intended effects. In other words, we compare whether the elements in A and B were also named in D of the research framework shown in Figure 25.

Figure 28 displays the actual effects that were identified from the coding of question 14 (see Appendix A).

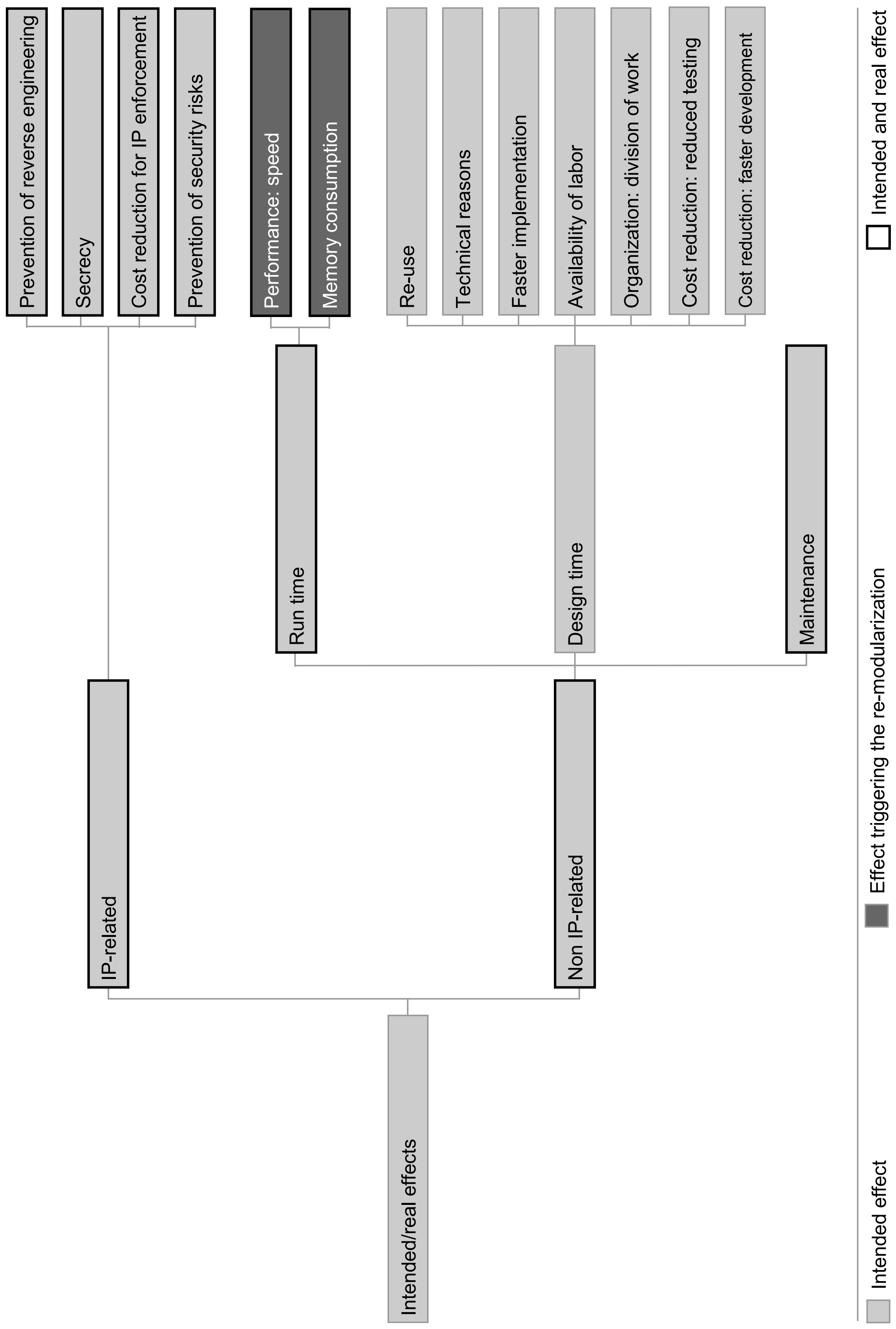

Intended/real effects
IP-related
Non IP-related
Prevention of reverse engineering
Secrecy
Cost reduction for IP enforcement
Prevention of security risks
Run time
Design time
Maintenance
Performance: speed
Memory consumption
Re-use
Technical reasons
Faster implementation
Availability of labor
Organization: division of work
Cost reduction: reduced testing
Cost reduction: faster development
Intended effect
Effect triggering the re-modularization
Intended and real effect

Figure 28 – Comparison of intended and real effects (Case 2)

The analysis reveals that the actual effects are a subset of the intended effects. In the engineering software case there were no unintended effects (ex-post) that arose due to IP modularity. On the side of the IP-related effects, the intended effects materialize as stated by three interviewees:

> *"So far (approximately 6 months after product launch) we have not seen compatible machines in the market." (Software architect, Person A)*

> *"For us, it is important that we have sufficient time to innovate further and use lead time to protect our leading edge technology - the shift of our core IP to technology B gives us this lead time." (General manager, Person B)*

> *"I'd say: goal achieved. I do not know any competitor that has used or violated algorithms that we protected (with the IP modular design)." (R&D manager, Person E)*

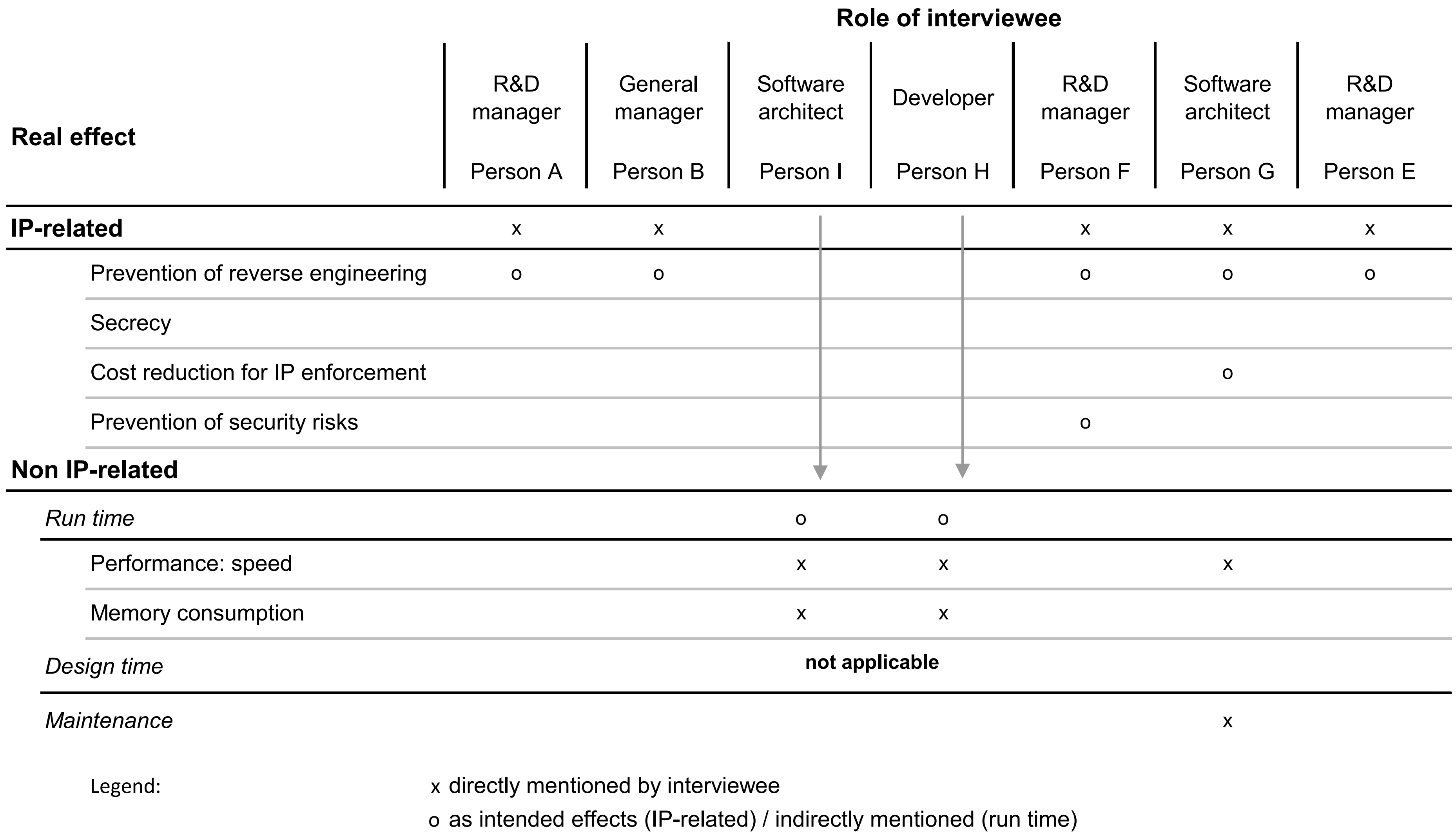
In addition, the initial triggers for the re-modularization – run time performance and memory consumption – could be solved with the re-modularization towards an IP modular structure:

> *"With this specific modularization, we increased the performance at factor ten. Without it, we would not have the engineering software on the market." (Software architecture, Person I)*

The design time effects were not mentioned throughout the interviews, which emphasizes that these effects are only observable during the development process and not during the actual run time phase. As the software product is released, the employees' focus is directed towards run time and IP-related effects[10].

The checklist matrix (see Figure 29) for the real effects shows that all of the interviewees except Person I and H mention the prevention of reverse engineering:

[10] An additional explanation could be that our interviews took place closely after the first product launch where the run time issues are more present than design issues in upcoming versions.



	Role of interviewee						
Real effect	R&D manager Person A	General manager Person B	Software architect Person I	Developer Person H	R&D manager Person F	Software architect Person G	R&D manager Person E
IP-related	x	x			x	x	x
Prevention of reverse engineering	o	o			o	o	o
Secrecy							
Cost reduction for IP enforcement						o	
Prevention of security risks					o		
Non IP-related							
Run time			o	o			
Performance: speed			x	x		x	
Memory consumption			x	x			
Design time			**not applicable**				
Maintenance						x	

Legend:
x directly mentioned by interviewee
o as intended effects (IP-related) / indirectly mentioned (run time)

Figure 29 – Checklist matrix of real effects (Case 2)

Despite explicitly asked for the real effect from company perspective this mention may be attributed to the specific roles of persons I and H. As already shown in the previous section, person I does not see IP protection as having a high impact on the achievement of the software architect's job-related goals. Also for Person H, the developer, the performance-related effects are of higher importance than IP protection, which is fully consistent with the goals that he pursues on a daily basis.

4.3 Incoming IP modularity in software products

To complete our picture of the effects of IP modularity in software products, we extend our findings with a case on incoming IP modularity (Case 4, see 3.2.1). In this case study, we analyze a large data management software product of a technology company with over 1 billion in annual revenue. As already outlined in Section 3.2.1, we must provide confidentiality with regards to product and person specific information.

The software product under investigation is a highly complex application that is implemented in several software technologies. To keep the software product up to date and competitive, constant development effort is required. R&D management therefore focuses its endeavors towards the generation of new functionality to differentiate the product from the competition. The use of well-tested and freely available open source components allows the developers to accomplish this goal. So, from release to release, the developers included more and more open source code in the different application modules as displayed in Figure 30 a).

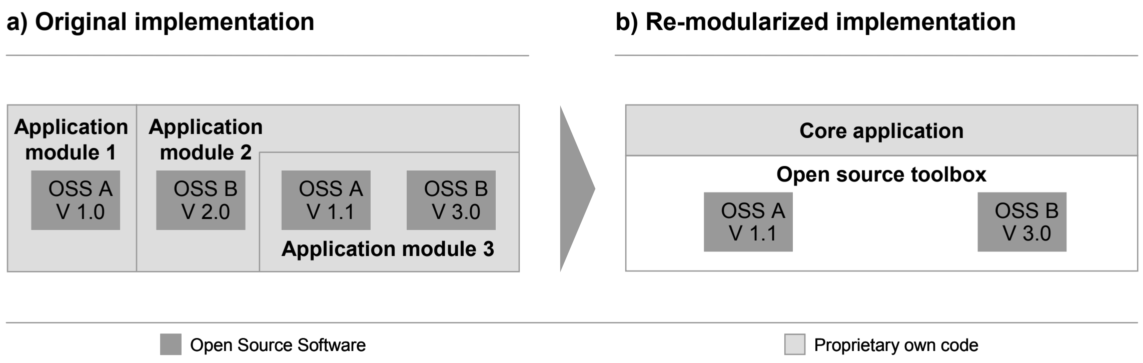


Figure 30 – Data management software (Case 4)

While the unmanaged use of open source components has helped developers to increase their productivity, it has also yielded a high level of fragmentation with respect to the proprietary source code and has lowered the degree of control over the various versions of the incoming IP. In this matter, the leading software architect states:

> *"We found versions where the (Ass. open source code) code was renamed and then was checked into our proprietary versions." (Software architect, Person AE)*

This mix of proprietary and third-party IP caused several problems. First, there was no control when open source code was used. Second, the open source code was copied into the own code tree and could not be easily detected. Third, there were different versions of one open source component in different application modules. Because sometimes the license conditions change across different versions of the open source software, these multiple versions impose significant legal risks.

To solve these problems, the architectural team re-modularized the software. They generated a special module named *open source toolbox* as shown in Figure 30 b). This toolbox is the only module that contains the open source software, as the leading architect further elaborates:

> *"We want to make sure that the open source of third-party products retains its identity and integrity. So if they were a JAR[11] file as packaged by the provider, they stay the exact same JAR file as when it is shipped. And the way we do that is we make sure they stay in the toolbox as a JAR file and that we don't take their source code and embed their source code in our source code and throw them together." (Software architect, Person AE)*

With the creation of the open source toolbox, the source code of the data management software product became IP modular in that the code of the proprietary core application no longer contained third-party IP. The entire architecture changed from an IP incompatible to an IP homogeneous status (as shown in Figure 6). Therefore, just as in Case 2, IP modularity was only established after a re-modularization effort.

[11] A Java Archive (JAR) file is a container for several Java classes (which can be seen as synonymous to modules in our context).

Remarkably, as in Case 2, the original modularization did not take into account IP as a criterion for modularization.

We base our findings on four interviews with the three key stakeholders as shown in Table 3. In total, we analyzed 67 pages of interview transcripts from five interview hours and the full description of the internal process for the integration of third-party IP (see Appendix C).

With all of the key stakeholders being involved and the core process description being available to us, we ensure that our dataset sufficiently represents reality so that we can draw well-grounded conclusions.

The case results are based on the clearly defined analysis process that has been outlined in detail in Section 3.2.3. As in the previous section, the interview coding started with an initial coding scheme that was derived from the interview questionnaire (see Figure 14). This process yielded hierarchical structures that were used for subsequent interpretation (see Appendix B). The analysis is structured based on the adapted research framework from Case 2 (see Figure 25) because it allowed us to better structure the answers for questions nine through 13. From this coding, the following intended effects for incoming IP modularity can be identified:

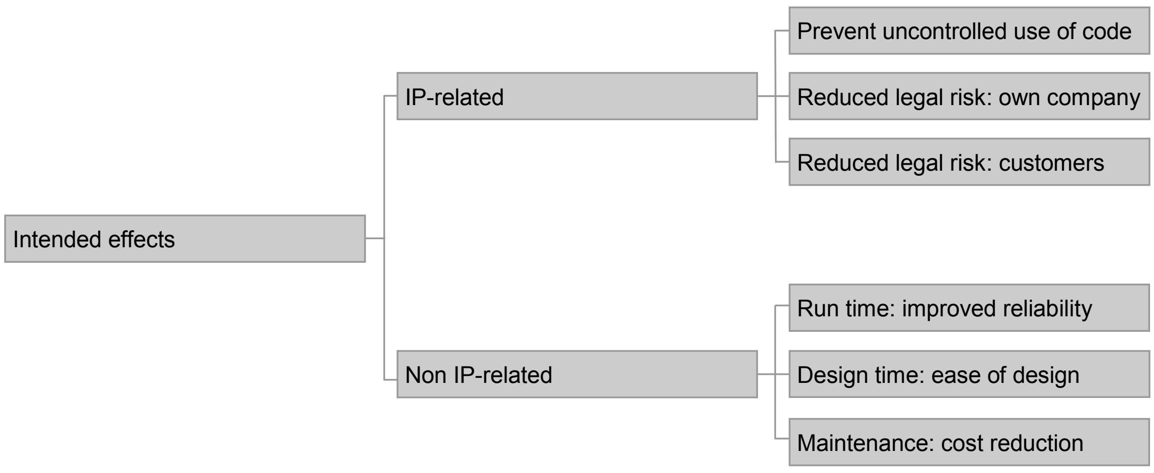

Intended effects
IP-related
Prevent uncontrolled use of code
Reduced legal risk: own company
Reduced legal risk: customers
Non IP-related
Run time: improved reliability
Design time: ease of design
Maintenance: cost reduction

Figure 31 – Intended effects (Case 4)

The analysis yielded a division between the IP-related and non IP-related effects – just as in Case 2. Similarly, the non IP-related effects can be divided into run time, design time and maintenance effects:

"It's really a situation where you have really difficult bugs that sneak into the product that are really difficult to find and so just by being conscientious we

kind of improve the reliability of the platform[12] *by having kind of more controlled behavior. I think it's about improved software quality, really." (Staff function manager, Person AF)*

For the IP-related effects, we discovered that the IP modular design prevents the uncontrolled use of code as mentioned earlier by the leading architect. The IP modular design prevents the legal risk of violating the license conditions of the incoming IP for the company and its customers. The checklist matrix in Figure 32 for the intended effects provides a more detailed picture of how the different stakeholders rate the intended effects:

Intended effect	R&D manager Person N	Software architect Person AE	Staff function manager Person AF
IP-related			
Prevent uncontrolled use of code		x	
Reduced legal risk: own company	x	x	x
Reduced legal risk: customers	x		x
Non IP-related			
Run time: improved reliability			x
Design time: ease of design		x	
Maintenenace: cost reduction	x		x

Legend: x Effect named one or several times

Figure 32 – Intended effects checklist matrix (Case 4)

All of the interviewees stated that risk reduction for the own company was one of the main intended effects. Hence, own company risk reduction appears to be a common theme and the main driver IP modularity in this case, as the responsible R&D manager best describes:

"What was happening was when there are different parts of the system using the same component, then we start diverging on the versions and we ran into issues

[12] In this case, the term platform refers to the core product platform and is not to be confused with a platform in a software ecosystem

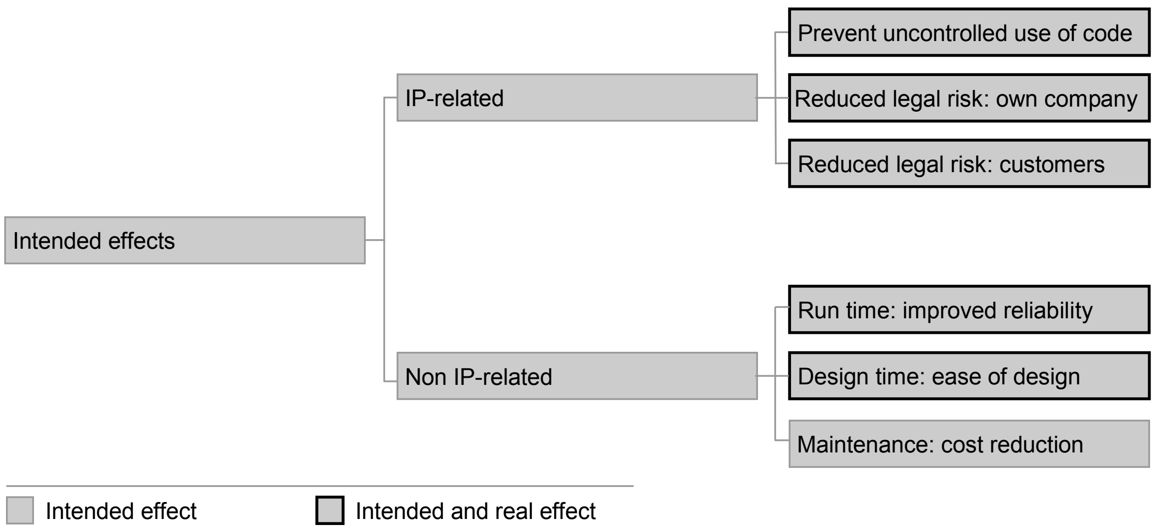
with different versions of these components which have different licensing associated with them from an IP viewpoint." (R&D manager, Person N)

IP-related customer risks also play a role; not as in Case 2 for security risks, but from a legal perspective:

"Our software becomes a mission critical application to them. [...] They are worried that if we run into any IP issues it would impact them because they have a deployment of our data management software." (R&D manager, Person N)

In spite of the consensus that IP issues are important from both a company and a customer perspective, the importance of IP was not taken into account when the data management software was initially developed. Instead, a re-modularization was required, and the toolbox for third-party IP was introduced. Currently, all incoming third-party IP requires an evaluation through a specifically designed process before it can be included in the toolbox (see Appendix C). Interestingly, within this process, the business/legal evaluation and the technical evaluation are on the same level. Therefore, business and technical aspects are formally treated with equal concern. Finally, the legal evaluation can prevent the use of the integration of the third-party IP in the toolbox.

As defined in the research framework the intended effects and the real effects are opposed. At the time of the analysis, the toolbox and the process were newly introduced, which limits our findings on the real effects to those that were observable at that time:



Intended effects
IP-related
Non IP-related
Prevent uncontrolled use of code
Reduced legal risk: own company
Reduced legal risk: customers
Run time: improved reliability
Design time: ease of design
Maintenance: cost reduction
Intended effect
Intended and real effect

Figure 33 – Comparison of intended and real effects (Case 4)

The incoming IP modularity also eases the software design. The developers are free to use the third-party IP from the approved toolbox components and are therefore much faster in development as the leading architect explains:

> *"That's one of the big benefits, that it becomes easier, much easier, to teach new developers. This is where we put third-party and this is where you put your own code. Two different places - two different processes. And then some processes to keep the wall between them intact. So that's really important." (Software architect, Person AE)*

After the implementation of the toolbox and the process, the proprietary part of the data management software became IP homogeneous. Having the security of the legal approval for the third-party components means that the legal risks are eliminated.

4.4 The effects of outgoing and incoming IP modularity in software products

The last two sections presented the within-case analysis results for outgoing and incoming IP modularity. We now move forward to the cross-case analysis, striving to unravel similarities and thus provide further external validity to our study.

Comparing the within-case analysis results, we identified one common pattern: in spite of the close linkage to the focal companies' business model, IP-related requirements were not captured in the initial requirements analysis. In both cases, large sections of the product were already realized. It was only when the products were already built that the IP-related issues drew the attention of the development team. Even more, in Case 2, the perception of IP-related risks only arose due to non IP-related run time issues (performance and memory consumption).

Based on our analysis, we find that an ex-ante inclusion of the IP-related requirements within the non-functional requirements not only protects a software companies' business model but also can prevent additional efforts to re-modularize a software product ex-ante to make it IP modular. Figure 34 presents the extended software requirements model as result of our analysis.

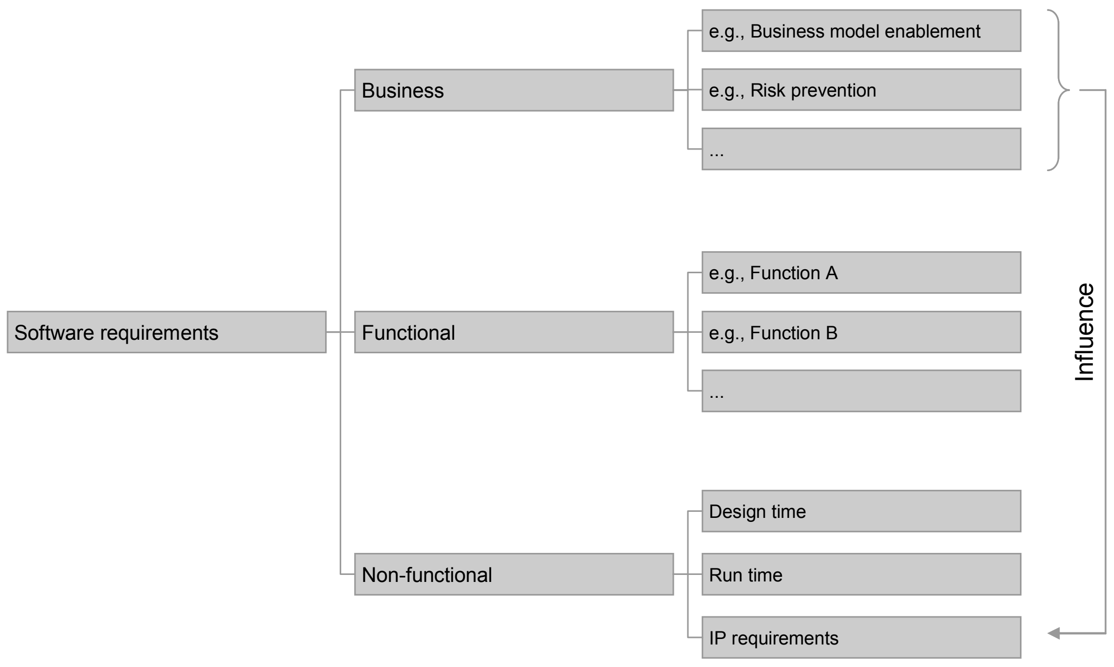


Figure 34 – Extended software requirements model

In both cases, the business requirements drive the need for IP modular design. For the outgoing IP modularity case on the engineering software, the safeguarding of the existing IP lessor business model was the main driver for the re-modularization. Based on that finding, the following proposition can be formulated:

Proposition A – IP requirements*: The earlier that IP requirements are considered in the software design process, the lower the overall cost to reach the final modular structure of a software system.*

For the incoming IP modularity case on the data management software, the prevention of IP-related risks was the main driver for the re-modularization. The comparison of the effects analysis reveals a fine-grained picture of the IP-related effects of IP modularity for the incoming and outgoing case:

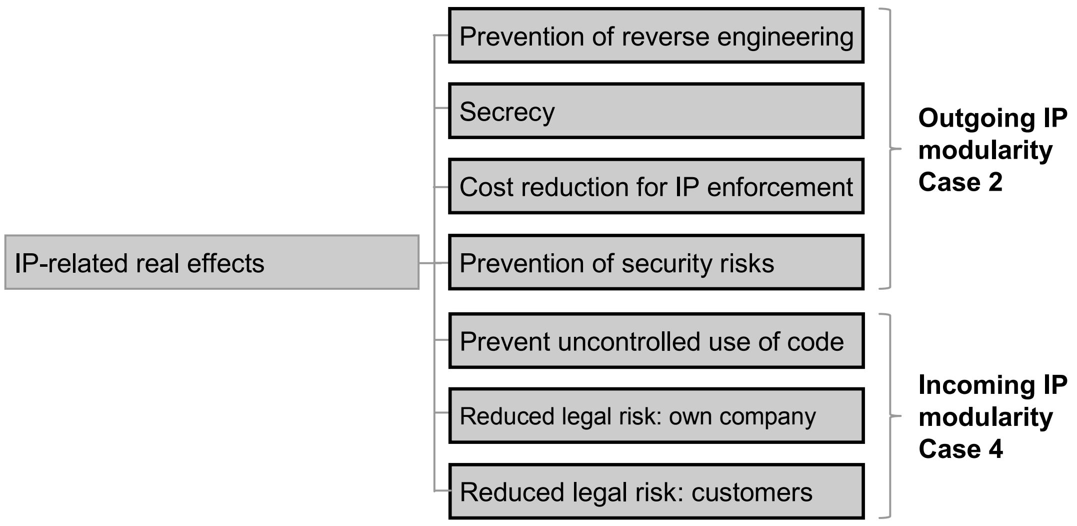

IP-related real effects
Prevention of reverse engineering
Secrecy
Cost reduction for IP enforcement
Prevention of security risks
Prevent uncontrolled use of code
Reduced legal risk: own company
Reduced legal risk: customers
Outgoing IP modularity Case 2
Incoming IP modularity Case 4

Figure 35 – Real effects of outgoing and incoming IP modularity (Case 2 and 4)

The cross-case comparison shows that the reduction of IP-related legal risks is a realized effect of incoming and outgoing IP modularity in both cases. The analysis of the incoming IP modularity directly reveals the reduction of legal risks as a primary rationale. For the outgoing IP modularity, the legal risks materialize from the legal IP enforcement efforts and the customer claims in the case of security risks.

In addition to the holdup risk from the supplier side (see holdup risk proposition in Section 4.1), there are also significant legal risks from the customer side and the competitor side. In Case 2, the IP modular design eliminates the possible cost of enforcing differentiating IP against competitors. Also the risk of being sued by a customer for security risks is reduced with the IP modular design.

Hence, IP modular design is not only favorable for incoming but also for differentiating IP, which in this case is identical to outgoing IP, with regards to legal risks. Figure 36 illustrates this situation.

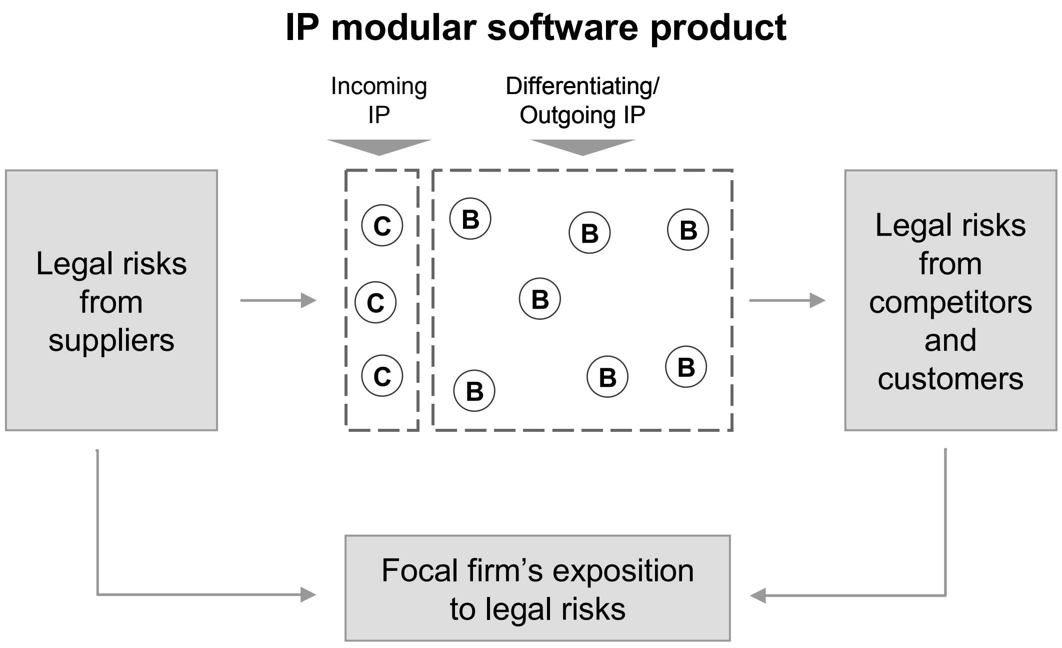

IP modular software product
Incoming IP
Differentiating/ Outgoing IP
Legal risks from suppliers
C
C
C
B
B
B
B
B
B
B
Legal risks from competitors and customers
Focal firm's exposition to legal risks

Figure 36 – Holdup risk from incoming and outgoing IP modularity

Hence, the holdup risk proposition initially defined by Henkel and Baldwin (2010) can be confirmed. Based on the cross-case analysis, a general proposition on legal risks can be formulated:

Proposition B – Legal risks: *IP modularity in a software product can mitigate the focal firm's exposure to legal risks from the supplier, competitor and customer sides.*

In summary, the cross-case analysis reveals the common effects of outgoing and incoming IP modularity. The empirical data show that IP as the criterion for modularization is directly related to the business requirements but was in the researched cased not considered in the initial design process as a non-functional requirement. Explicitly including IP requirements in the software design process offers the potential to save cost from re-modularization ex-post.

The analysis also shows that IP modular software product design can reduce the exposure to legal risks. IP modular software design introduces product design as a valid lever to manage the legal risks of a focal company.

4.5 Conclusion

This chapter describes two cases of software products where the modular structure is related to IP optimization. In the first case of engineering software the IP modular product structure prevents the reverse engineeering of the application’s core IP. In the second case of data management software the IP modular product structure prevents the uncontrolled diffusion of third-party IP into the core modules of the product.

In both cases IP optimization was not a part of the original software requirement specification, resulting in a later re-modularization which could have been prevented by considering IP as initial non-functional requirement.

Finally the findings provide indications that IP modular design is an important lever to reduce a firm’s exposure to legal risks.

5 IP modularity in software platform ecosystems

In this section, we explore SugarCRM and SAP NetWeaver PI[13] as concrete realizations of IP modularity in software platforms. Software platform providers face the conflicting goals of openness toward providers of complementary goods for the purpose of distributed value creation, and of maintaining exclusivity of essential modules to appropriate value (Henkel and Baldwin, 2010).

Hence, the goal of IP modularity is to optimize the system with respect to the firm's business model and particularly to reconcile distributed value creation and value appropriation. While relinquishing control typically increases adoption and outside contributions, it makes it harder for the platform owner to realize profits and maintain differentiation (Boudreau, 2010). To analyze whether and how the concept of IP modularity can be applied in software platforms, we study two different software platform products that are particularly interesting in this respect: SugarCRM and SAP NetWeaver PI. The two platforms represent polar cases as they are based on entirely different IP strategies and related business models (see Figure 12). SAP NetWeaver PI is a market-leading software platform with a restrictive approach in terms of IP provision to third-parties, whereas SugarCRM runs its platform as open core[14] and allows its complementors to access large parts of the internally developed IP directly[15].

This section aims to understand the strategic rationales behind making a software platform IP modular to answer the specific research questions defined in Section 3.2.1:

- Why are software platforms modularized with regard to IP considerations?
- How does IP modular platform design influence the cooperation between platform provider and complementors?

In addition, we juxtapose our findings to the propositions derived by Henkel and Baldwin (2010) (see Section 3.2.1). Analyzing the reasons behind and the effects of

[13] The full product name is SAP NetWeaver Process Integration.
[14] See http://alampitt.typepad.com/lampitt_or_leave_it/2008/08/open-core-licen.html.
[15] The analysis of the case of SugarCRM in this section is partly based on an earlier publication on the topic: Waltl *et al.* (2012)
The reprint is with permission of © Springer-Verlag Berlin Heidelberg.

SugarCRM Inc.'s adoption of an IP modular architecture, we find them fully in line with theoretical predictions.

Based on the analysis of both cases, SugarCRM and SAP NetWeaver PI, we introduce IP optimization as a new non-functional requirement for the design of software platforms (in line with the findings on IP modularity in software products in Section 4). Specifically, we argue that the delineation of modules must be decided in conjunction with their "IP status" – e.g., if they are proprietary and binary-only, sources licensed to select recipients, or under the GPL (Henkel and Baldwin, 2010).

We also provide insight on how the identified effects resulting from an IP modular platform design influence both the platform and product attractiveness, as well as the competitive position toward other platforms.

In the following sections, we provide a short introduction into the current literature on software platform ecosystems and present the results of the within-case analysis and the commonalities across the two polar platform cases.

5.1 Software platform ecosystems

In the following section, we provide a brief introduction to the concept of and the dynamics behind platform ecosystems in order to build a foundation for the upcoming case analysis.

Technological platforms have extensively been researched in the last few years (Gawer and Cusumano, 2002; West, 2003; Gawer and Henderson, 2007; Baldwin and Woodard, 2009; Boudreau, 2010; Cusumano, 2010).

Our understanding of "platform" draws on the work by Gawer and Cusumano (2002), who define an industry platform as a product that provides a core technology that can be reused in different product variations, that are likely to come from different companies. In addition, Baldwin and Woodard (2009) define a "platform architecture" as a modularization that partitions a system into a set of components, with a stable design, and a complementary set of components, which are allowed — indeed encouraged — to vary (see Figure 37).

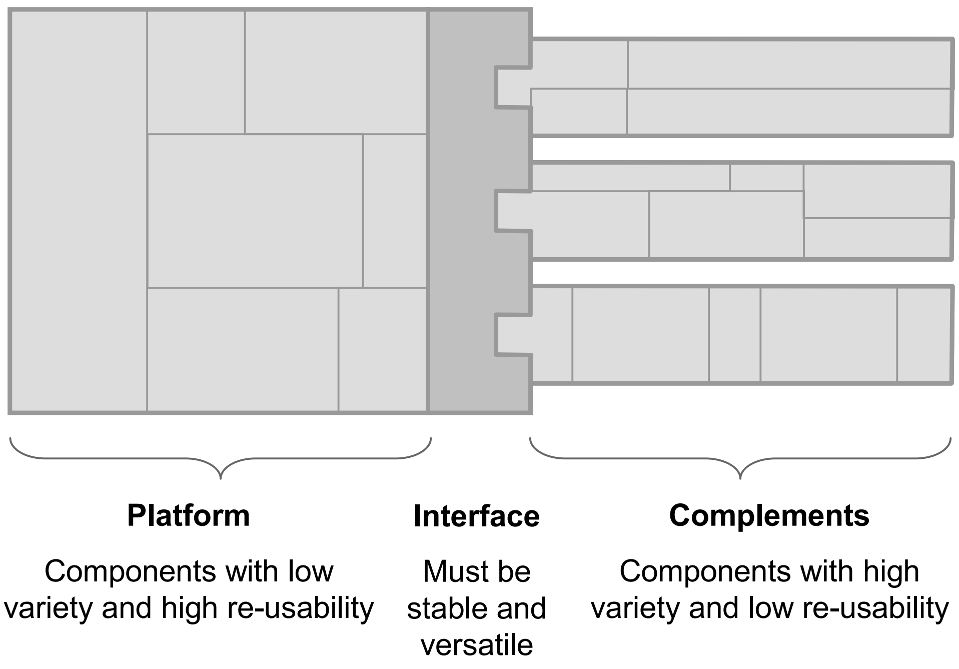

Platform
Components with low variety and high re-usability
Interface
Must be stable and versatile
Complements
Components with high variety and low re-usability

Figure 37 – Platform architecture (based on Baldwin and Woodard, 2009)

In many cases, not only the company that built the platform but also other companies generate complementary components[16] that foster innovation in functionality (Boudreau and Lakhani, 2009). End-users then profit from both the platform core functionality and the additional functionality provided by the complementors.

Eisenmann *et al.* (2009) provide a model for platform-mediated networks to describe the interaction between the providers of industry platforms (Gawer, 2009), complementors and end-users:

[16] We name these companies *complementors* as first defined by Brandenburger and Nalebuff (1996)

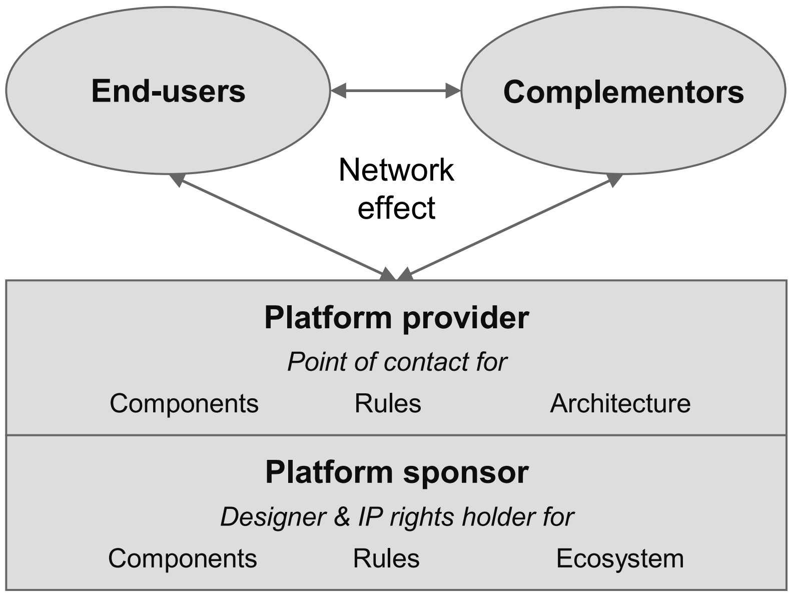


Figure 38 – Platform-mediated network (based on Eisenmann *et al.*, 2009)

Furthermore, they differentiate between *platform sponsor* and *platform provider*. In our cases on SugarCRM and SAP NetWeaver PI, sponsor and provider are the same party, which is why we use the latter term for both.

In platform-mediated networks network externalities exist between platform provider, end-users and complementors (Katz and Shapiro, 1985). Platform providers must get both sides of the market on board (Rochet and Tirole, 2003). A platform is more attractive for end-users the more complementary products exist which extend the core platform functionality. Similarly, a platform-mediated network is more attractive for possible complementors when the market of possible end-users is larger (see Section 6.1). Additionally, for platform providers, it is obviously beneficial to nourish the ecosystem of complementors and end-users to increase the sales of the core platform.

For the context of this dissertation, we use the model of a platform-mediated network as the basic definition of a software platform ecosystem. Such an ecosystem is created when a platform provider releases an external API for the development of complements. Independent software vendors (ISVs) or system integrators (SIs) can then generate platform complements – applications or apps – and sell them to end-users. In line with this description, Jansen and Cusumano (2012, p. 46) formally define a software ecosystem as:

"A software ecosystem is a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships

among them. These relationships are frequently underpinned by a common technological platform or market and operate through the exchange of information, resources and artifacts." (Jansen and Cusumano, 2012, p. 46)

Opening up an industry platform to a wide set of potential complementors implies that the platform provider has to cede a certain degree of control to catalyze the development of complements, as Boudreau (2010) shows for the case of handheld computing systems. There is a broad consensus in the literature that the management of IP is a key lever for platform providers to manage the cooperation with ISVs and SIs. Cusumano (2010) notes the connection of platform modularity and openness (through accessibility of the interfaces and IP) as a major lever for platform leadership (see Section 6.1). These interfaces are typically implemented as APIs as a means to control the openness of proprietary platforms, and the complementors fully depend on these interfaces. Our research on the IP restrictive SAP NetWeaver PI platform aims to illuminate further this dependency and link it to IP optimization and platform architecture (see Section 2.2).

The challenge lies in preventing complementors from imitating features essential to the platform. Hence, it is paramount to find the right degree of openness, which can, to a large extent, be influenced through the provision of IP and the modular structure of the platform (Cusumano, 2010).

The decision regarding which parts of a system are open and which are closed depends on the business model, which can be open to varying degrees: e.g., totally open source, hybrid, or based on a proprietary approach (Bonaccorsi *et al.*, 2006; Lindman *et al.*, 2011). In the platform design process, the architects need to balance these requirements with other requirements, such as technical considerations or R&D process requirements (Favaro and Pfleeger, 2011).

A similar tension between openness and control for platforms exists for commercially developed OSS. If an OSS project is organized on a public platform with code coming from a large number of contributors, then the potential for distributed value creation is maximized. Downsides are that in such a case a product-based business model is difficult or impossible, and the original owner of the code may even lose control over its further development.

Various approaches have been proposed and implemented in practice to harness the power of community-based OSS development while still running a product-based business model. Hecker (1999) and Raymond (2001) suggest various ways to profit indirectly from OSS by selling complements. West (2003), Bonaccorsi *et al.* (2006) and Lindman *et al.* (2011) empirically study "hybrid" business models that either combine open source and proprietary elements or make use of dual licensing. Riehle (2012) comprehensively presents the properties of this hybrid approach, referred to as *single vendor commercial open source business model.*

Our research relates to this literature as SugarCRM Inc. licenses a community edition of its software under an open source license while selling the commercial edition under proprietary license terms. However, their approach goes beyond simple dual licensing by creating two distinct versions out of the same IP modular code tree.

To our knowledge, this study is the first to link research on hybrid OSS business models with that on modular product architectures.

5.2 IP modularity in an open source software platform ecosystem

SugarCRM Inc. was founded in 2004 and provides an open source CRM software platform product and commercial editions[17] with extended functionality. This software platform is the object of our study. IP modularity is of the highest strategic relevance as the CTO and co-founder explains[18]:

> *"I would say it's one the most fundamental aspects of our entire business model strategy that we purposely keep the modularity in such a way that we can easily create different [open source and proprietary] editions. What we sell is based on IP modularity." (Clint Oram, CTO and co-founder)*

The business model built around SugarCRM differs from that of other open source companies in the way that IP and source code ownership are managed. The company maintains all source code for its products in one proprietary code tree and licenses parts of it for the open source community edition under the GNU Affero General Public

[17] At the time of this research four commercial editions exist that add further functionality. For our study we do not differentiate between them. However, a detailed look across the versions in further research would add an additional dimension to our findings.
[18] The basic concept of IP modularity was explained prior to this statement as part of our research introduction, which is why Mr. Oram was able to directly apply it to describe SugarCRM's business model.

License (AGPL)[19]. The commercial product editions are sold to customers under proprietary license terms[20]. Thus, SugarCRM combines an open source approach with a proprietary software product business model (see Section 4.1 for details on software business models). SugarCRM also opens up the platform to enable complementors to create additional extension modules for end-users and builds an entire software ecosystem.

Our research draws on twelve interviews with SugarCRM Inc. executives in technical, business and legal roles, as well as executives of complementors (see Table 3). In total, we collected more than ten hours of interview recordings that were transformed to 136 pages of interview transcripts. The interviews with SugarCRM executives were mostly in person and took place at the headquarters in Cupertino, CA, and the firm's office in Munich, Germany. The interviews with the ISVs were conducted via telephone. To triangulate our data, we complement the interviews with information from extensive field notes, analyst reports (see Table 4) and data from the company's open source community website[21].

Our research shows that SugarCRM's platform architecture and its business model are inseparably linked. The business model is enabled by a product architecture that separates the IP elements for the community edition from the elements required for the commercial editions. Figure 39 depicts the basic architecture in a schematic representation for selected platform functionality and extension modules.

[19] See www.gnu.org/licenses

[20] Sugar CRM is implemented in PHP technology, which does not allow to keep source code secret for it is interpreted at run time and not compiled. Following this technical conditions SugarCRM operates on a combination of OSS and open code software.

[21] See www.sugarforge.org

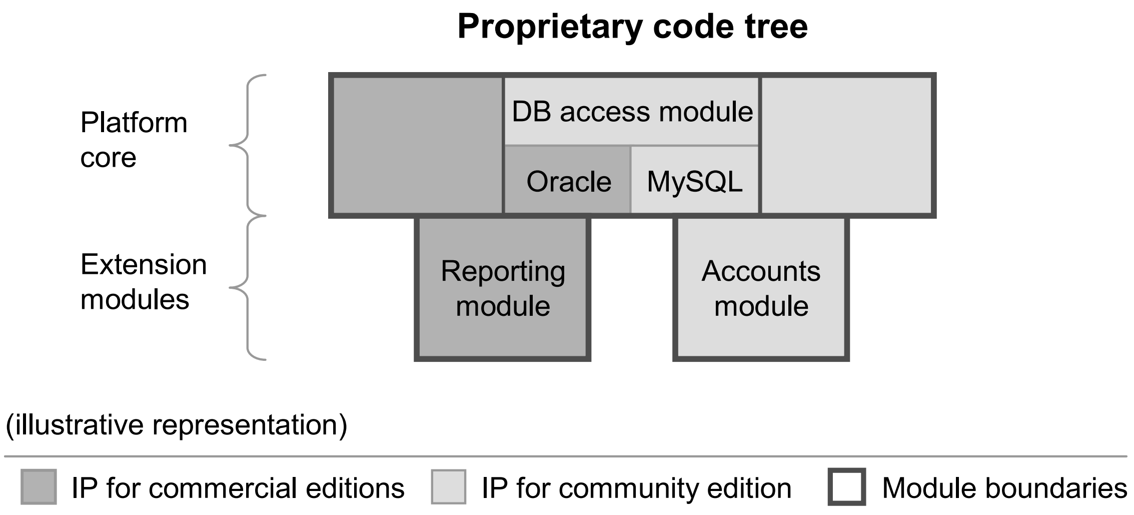


Figure 39 – Schematic architecture overview (Case 11) [22]

Nick Halsey, VP (Vice President) of marketing, explains the link between SugarCRM's product architecture and the company's business model as follows:

> *"Our business model would not be possible without an IP modular architecture. If we solely had an open source product and sold consulting services around it, we would probably do very well, but we wouldn't have the same kind of explosive growth[23] we're experiencing with our commercial product." (Nick Halsey, VP of marketing)*

The goal of SugarCRM is to reach clear IP modularity between the open platform core and the extension modules. This is the case when certain business functionality that is to be included into one of the commercial editions can be implemented in one or several extension modules.

However, there may be situations where this is not possible because some commercial edition functionalities simply cannot be implemented in one extension module and require modifications of the platform core.

If somehow possible, such IP for commercial editions is then encapsulated in separate modules within the platform. In the worst case, the code for the commercial

---

[22] Source: Waltl *et al.* (2012)
© Springer-Verlag Berlin Heidelberg – reprinted with permission
[23] To give an example: In North America the billings for the first quarter in 2011 showed a 63% growth over the same quarter in 2010 [17]

editions cannot be split into different modules (called components in the SugarCRM context) within the platform. Lila Tretikov, CIO and VP of engineering states:

> *"We try to keep the platform IP modular as well, but there are some historical things that are there from before. An example of this would be our general database component that has code to connect to MySQL that goes into our community edition and code to connect to Oracle that goes into our commercial editions." (Lila Tretikov, CIO and VP of engineering)*

To overcome this problem and reach IP homogeneity in the released editions, the code is tagged in the proprietary code tree when it is only to be included in one of the proprietary editions. As shown in Figure 40, a special build process removes the IP for the commercial editions from the code that is released under the AGPL license. Thus, IP homogeneity within the community and the commercial editions can be achieved with only one proprietary code tree that has to be managed. Alternatively, two parallel code trees would have to be managed that only differ in the parts for the commercial editions. This would require additional maintenance effort and increase the risk of incompatibility between the versions.

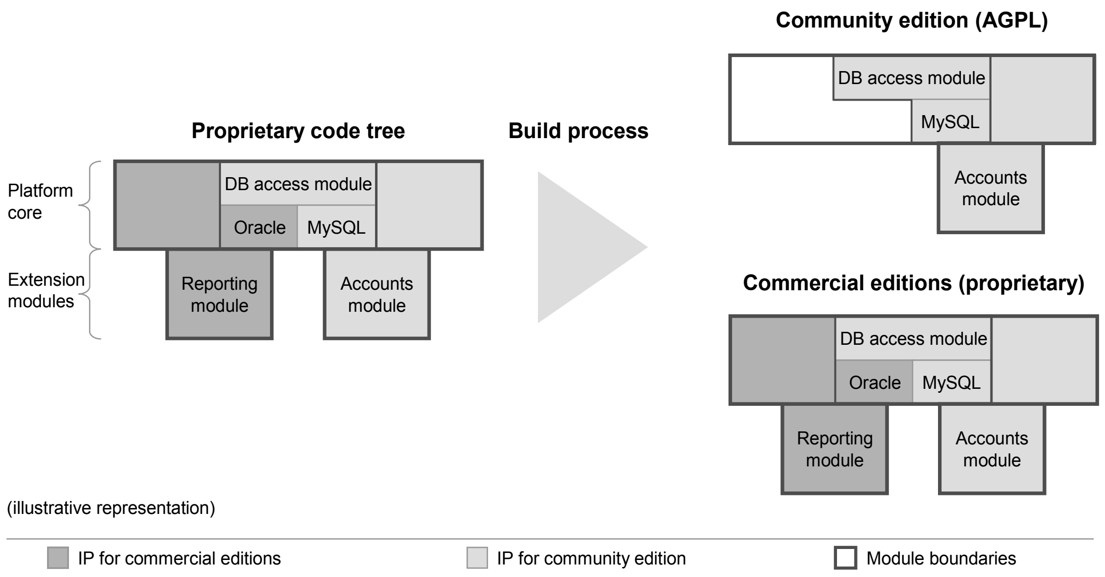


Figure 40 – Build process to separate IP (Case 11) [24]

[24] Source: Waltl *et al.* (2012)
© Springer-Verlag Berlin Heidelberg – reprinted with permission

Based on these findings, we can extend the concept of IP modularity to include the notion of hierarchy levels. In our case, we identified three levels of IP modularity. On the first and highest level, the relevant elements are the platform and the extension modules on an architectural level. The second level is within the platform itself in the way that its sub-modules are IP homogeneous. On the third level, the modules must split on a source code level to reach IP homogeneity in the end products.

Table 6 provides a detailed description of the identified hierarchy levels of IP modularity and their implications for platform strategy, governance mechanisms, and implementation. According to CTO Clint Oram, the overall aim is to keep the IP separation at the platform architecture and module level because the implementation of IP modularity on the source code level implies higher technical complexity and implementation cost.

	Platform architecture level	**Module level**	**Source code level**
Characteristics	▪ Modularity between between architectural elements	▪ Modularity between modules within one architectural element	▪ Modularity between source code sections within one module
Strategic rationale	▪ Split open platform from proprietary extension modules ▪ Fundamentally enabling "open core" business model	▪ Separation of proprietary and open modules	▪ Separation of proprietary and open code segments
Goverance	▪ Archtectural design committee defines general design rules	▪ Product management guides engineers	▪ Product management guides engineers
Implementation	▪ Platform API ▪ Module builder toolset	▪ Assignment of IP status to modules	▪ Assignment of IP status to code segments
Cost of implementation	Low	Low	High

Table 6 – Hierarchy levels of IP modularity [25]

In the case of SugarCRM, the need for IP modularity on the source code level is given for three different reasons. First, the SugarCRM platform is a constantly evolving product, and the focus towards IP modularity has increased over time. Thus, for historical reasons, there remains code that has not been designed with the principle of IP modularity in mind, as Nick Halsey explains:

[25] Source: Waltl *et al.* (2012)
© Springer-Verlag Berlin Heidelberg – reprinted with permission

> *"Now we are 100 percent committed to IP modularity. In the early days our main focus was to bring our product to the market." (Nick Halsey, VP of marketing)*

Second, there are modules that cannot be split due to technical or architectural reasons, for example, the DB access module as shown in Figure 40. Here, the need for technical optimality justifies the additional cost of implementing IP modularity on the source code level.

Third, the further down IP modularity is implemented in this layer model, the more granular the balance between openness and value appropriation can be managed, as CEO, Larry Augustin, explains:

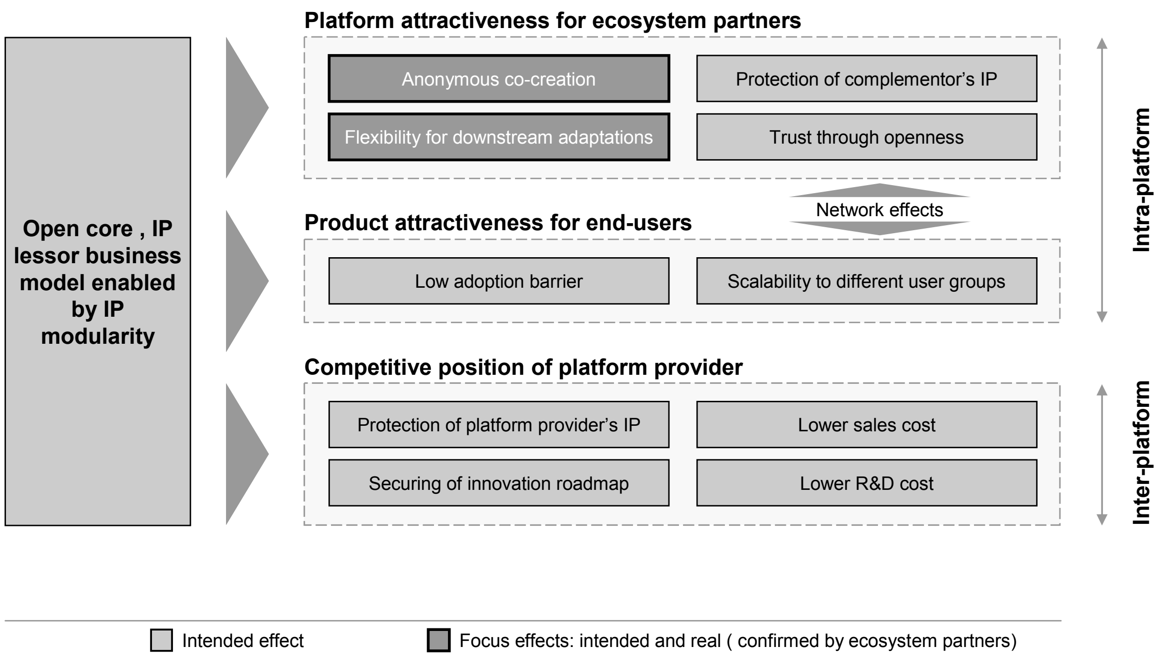
> *"With our IP modular architecture, we have more flexibility and can draw that line in a more granular place. For example, parts of our open platform can be proprietary. [...] I think we have more flexibility to choose what is free versus what is not free." (Larry Augustin, CEO)*

This highly granular possibility to control the IP even on the source code level for commercial versions is a core enabler for increased openness because modules that only partly consist of proprietary elements do not have to be kept proprietary as a whole. On the other hand, these mechanisms to separate IP on the source code level can lead to higher cost, at least initially, as Nick Halsey accounts:

> *"If you take just a little bit of extra time up front to determine the right IP modular design that fits your business needs, it might mean higher cost up front, but in the end, you will wind up saving time, saving money and having increased productivity over time." (Nick Halsey, VP of marketing)*

When IP modular platform design is fundamentally linked with a company's business model, it has to be decided to which level IP modularity is implemented to be beneficial from an overall strategic perspective. This may – and in the present case does – outweigh higher cost in R&D and/or technical drawbacks.

In addition to enabling SugarCRM's business model, the IP modular platform design entails an entire set of additional effects on company performance and the entire platform ecosystem that can be clustered into inter-platform and intra-platform effects. As in the previous chapter, we base our findings on the extended research framework of intended and real effects (see Figure 25). Figure 41 shows the combined results of the intended and real-effect analysis. A full overview of the underlying coding scheme is provided in Appendix D.



Platform attractiveness for ecosystem partners
Anonymous co-creation
Protection of complementor's IP
Flexibility for downstream adaptations
Trust through openness
Open core , IP lessor business model enabled by IP modularity
Network effects
Product attractiveness for end-users
Low adoption barrier
Scalability to different user groups
Competitive position of platform provider
Protection of platform provider's IP
Lower sales cost
Securing of innovation roadmap
Lower R&D cost
Intra-platform
Inter-platform
Intended effect
Focus effects: intended and real (confirmed by ecosystem partners)

Figure 41 – Intra- and inter-platform effects (Case 11) [26]

The intra-platform effects can be divided further into effects that increase platform attractiveness for complementors and those that increase product attractiveness for end-users. Not surprisingly and in line with existing research, our data indicate a network effect between those two (see Figure 38). Our interviews show that executives devote particular attention to the platform attractiveness for ecosystem partners, as Nick Halsey testifies:

> *"By taking the IP modular architecture approach, we have made it easier for our partner ecosystem to develop add-ons and extensions to our product that*

[26] Own figure based on: Waltl *et al.* (2012)
© Springer-Verlag Berlin Heidelberg – reprinted with permission

they can build businesses around. As a result, that means we have a much larger ecosystem with better solutions that are easier to implement and upgrade." (Nick Halsey, VP of marketing)

Furthermore, for anonymous co-creation (see proposition 2 in Section 3.2.1), the specific platform architecture enables the highly open strategy, which eliminates the need for complementors to interact directly with the platform provider. The CEO, Larry Augustin, illustrates why this fosters innovation in the entire ecosystem:

"There are many third-parties that show up and say: We have a product that works with SugarCRM, and they try to sell to our customer base. Many third-parties created those integrations using our open source tools, and they don't have to talk to us at all to develop a useful solution. We may not have supported them if they had chosen to talk to us." (Larry Augustin, CEO)

In addition, we found out that the possibility for anonymous co-creation paved the way for strong partnerships with complementors. Mirco Müller, CEO of Insignio CRM (one of SugarCRM's largest partners in Europe) describes this process as follows:

"In the beginning we did our business only based on the community edition. SugarCRM did not know us. We started to partner when we acquired our first big customers – still we did not interact much with SugarCRM. We were able to solve our problems on our own since we had access to all source code for the community and commercial editions. That changed when Sugar opened an office in Europe and we now do interact very closely, especially in marketing." (Mirco Müller, CEO of Insignio CRM)

These findings on the rationales for and the effects of IP modularity are perfectly in line with and, thus, support proposition 2 in Section 3.2.1 regarding the distribution, number and anonymity of potential value co-creators. Similarly, the following findings on customization and downstream adaptations lend support to proposition 4 in Section 3.2.1. As CTO Clint Oram explains:

“The reason why our ecosystem partners come to us is because we reduce risk for them. They have more control over the building of products because they have full visibility into the source code, and they understand exactly how the product works.” (Clint Oram, CTO and co-founder)

Additionally, the ecosystem partners appreciate the flexibility for adoption and customization on the platform core to implement end customer requirements. The core benefits are best described by Clemens von Dinklage, CEO of SugarCRM’s Gold-Partner MyCRM GmbH:

“We don’t have to ask SugarCRM when we customize the product, since we can open the engine hood ourselves and implement customer requirements directly without additional communication overhead towards the platform vendor.” (Clemens von Dinklage, CEO MyCRM)

Overall, the identified framework of intra-platform and inter-platform effects shows that IP modular platform design has a variety of strategic implications far beyond the basic business model mechanics. The checklist matrix (see Figure 42) proves that the effects on *anonymous co-creation* and the *flexibility for downstream adaptations* that are intended by the platform provider management materialize on the ecosystem partner side.

	Role of interviewee							
	General manager	Legal expert	R&D manager	Staff function manager	General manager	General manager (ISV partner)	General manager (ISV partner)	General manager (ISV partner)
Intended effect	Clint Oram	Jay Seirmaco	Lila Tretikov	Nick Halsey	Larry Augustin	Christian Knoll	Mirco Müller	Clemens v. Dincklage
Platform attractiveness for ecosystem partners								
Anonymous co-creation	x				x	x	x	x
Flexibility for downstream adaptions		x		x	x	x	x	x
Protection of complementor's IP	x							
Trust through openness	x			x	x			
Product attractiveness for end users								
Low adoption barrier					x			
Scalability to different user groups		x						
Competitive position of platform provider								
Protection of platform provider's IP	x	x	x	x				
Lower sales cost					x			
Lower R&D cost	x			x				
Securing of innovation roadmap				x				

Figure 42 – Intended effects checklist matrix (Case 11)

We found that IP modular platform architecture is the key enabler for the company's hybrid OSS business model. SugarCRM separates components that it licenses under an OSS license from those that it puts under a proprietary license in a single code tree, and in this way, it manages to combine the benefits of open source licensing with those of a proprietary product-based business model.

With a detailed analysis of SugarCRM's software platform, we identified three hierarchy levels for the application of IP modularity: the architectural level, the module level and the code level. IP modularity is easiest to implement on the architectural level, but hybrid architectural components may remain. These, in turn, can be made IP modular through a clear separation on the module level. Nevertheless, there may remain modules with a hybrid IP status that require differentiation on the source code level. The further down in this hierarchy IP modularity is implemented the higher is the related cost, but also the better is the strategic control of value appropriation in a hybrid OSS business model.

We also identified a set of secondary effects that can be clustered into effects that increase the platform attractiveness for end-users and effects that increase the competitive position towards other platform providers. Analyzing the effects on platform attractiveness for ecosystem partners, we found them fully in line with propositions derived in earlier work (Henkel and Baldwin, 2010).

Our findings on the implementation and the effects of IP modularity in one particular software ecosystem may provide insights beyond that particular case, encourage additional research, and provide insights for industry practitioners.

5.3 IP modularity in a proprietary software platform ecosystem

SAP NetWeaver PI is a core platform of SAP's enterprise software portfolio that connects SAP solutions to third-party solutions. As SAP decides to focus on a defined set of connectors to the most common third-party software systems, it opens up the platform for third-party companies to develop additional connectors to other systems. The ecosystem around NetWeaver PI is closely managed in that complementors need to certify under SAP conditions. The provision of knowledge is strictly controlled via an API, as well as access to data structures and sample code for API implementation.

Similar to SugarCRM, SAP implements an IP lessor business model (see Section 4.1). It is sold as an on-premise[27] solution that typically is embedded into the customers' IT landscape with the help of a system integrator.

Contrary to SugarCRM, NetWeaver PI is a completely proprietary system, and access to the platform is only granted to selected ecosystem partners via an external API. As this API is rich in functionality, it reveals platform internal knowledge. Accordingly, it is implemented in a separate set of modules. While this separation of internal and outgoing IP is common in API design, the SAP software architects further separated the business logic implementation in a separate module (A) and inserted an SAP internal API module (B) that is not visible for external application developers. Figure 43 illustrates how SAP NetWeaver PI implements IP modularity.

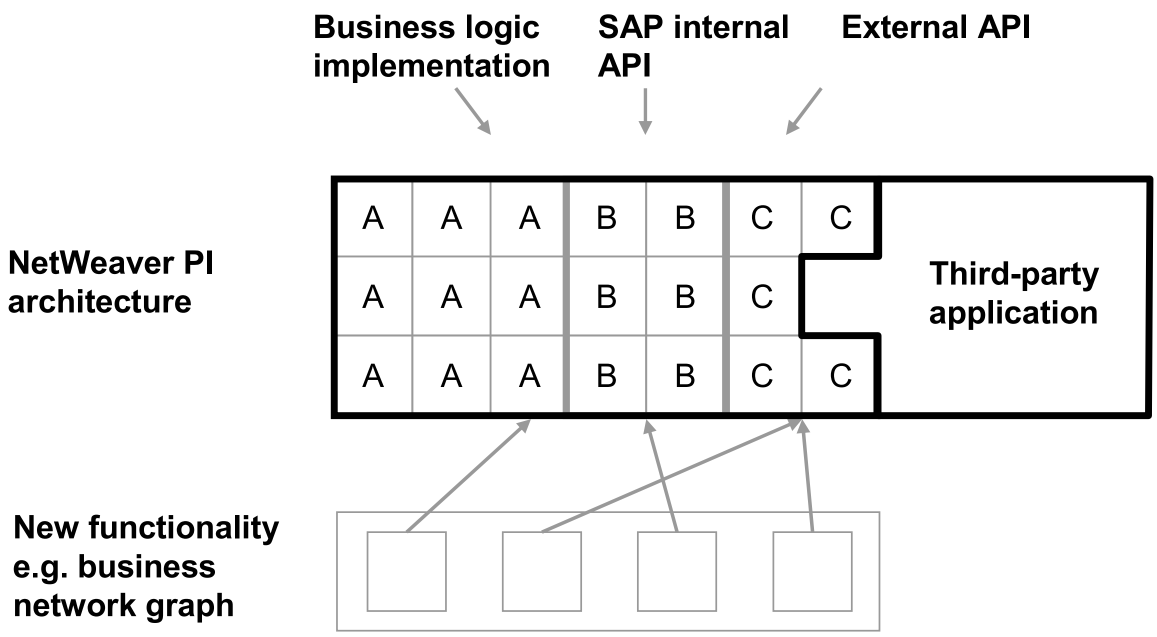

Business logic implementation
SAP internal API
External API
NetWeaver PI architecture
A A A B B C C
A A A B B C
A A A B B C C
Third-party application
New functionality e.g. business network graph

Figure 43 – IP modularity in SAP NetWeaver PI (Case 9)

From an abstract, top-level perspective, the NetWeaver PI platform consists of three modules: the implementation of the business logic that under no circumstance is to be revealed to third-parties; the SAP internal API that is subject to change and available to other SAP product units, and the external API that is to be kept stable across releases.

Whenever new functionality is implemented, it is allocated to one of the three modules. That is, its architectural integration is tightly linked to the decision about its IP

[27] On-premise software describes a model where the a customer buys a software license to run the product on its own hardware.

status. Thus, IP modularity is a core requirement for new functionality. As for SugarCRM, the importance of establishing and maintaining IP modularity of the architecture increased over time. In particular, the adaptation of the external API from version 7.0 to 7.1 caused additional effort in the entire ecosystem as Frank-Oliver Hoffmann, the leading software architect for SAP NetWeaver PI, reports:

> *"There are cases where the API is very closely linked to the implementation and it can happen that artifacts are to be exposed that are core IP. We had such cases until version 7.0. To overcome this problem we inserted the internal API module, but had to break the API compatibility from version 7.0 to 7.1. Now we are able to exactly decide what we want to expose and keep the API stable at the same time." (Frank Oliver Hoffmann, Software architect SAP)*

On the complementor side John Senor, software architect and Chief Development Officer (CDO) at Information Builders Inc., confirms the importance of an open and stable API:

> *"It is the openness and the robustness of the APIs that they provide. That are absolutely the most important points [for making a platform attractive]." (John Senor, Software architect and CDO at Information Builders Inc.)*

To ensure IP modularity, the SAP architects implemented a special build process that assigns different code artifacts to one of the three modules: *Business logic implementation*, *SAP internal API* or *External API*.

For our research on SAP NetWeaver PI, we identified six interview partners and conducted a total of eight interviews. Of these, three were with general and staff function managers on the platform provider side, whereas two were with software architects of companies providing complements. In addition, we conducted two in-depth interviews with the leading software architect for NetWeaver PI and one interview with an ecosystem manager. Overall, close to six hours of interviews resulting in 48 pages of interview transcripts built a solid empirical basis for our findings.

For each case, a set of initial interviews was conducted that aimed to understand the implementation of IP modularity in that specific case. All other interviews were based on an interview guideline derived from the basic research questions (see Section 3.2.1) and existing literature.

We coded all empirical data using the software package NVivo, using the interview guidelines for platform providers and ecosystem partners (see Figure 14 and Figure 15) as the initial coding scheme. As the first findings emerged from the data, the coding scheme was adapted in an iterative approach.

The analysis of the effects of IP modularity reveals the same structure for inter- and intra-platform effects as in the SugarCRM case described in the previous section:

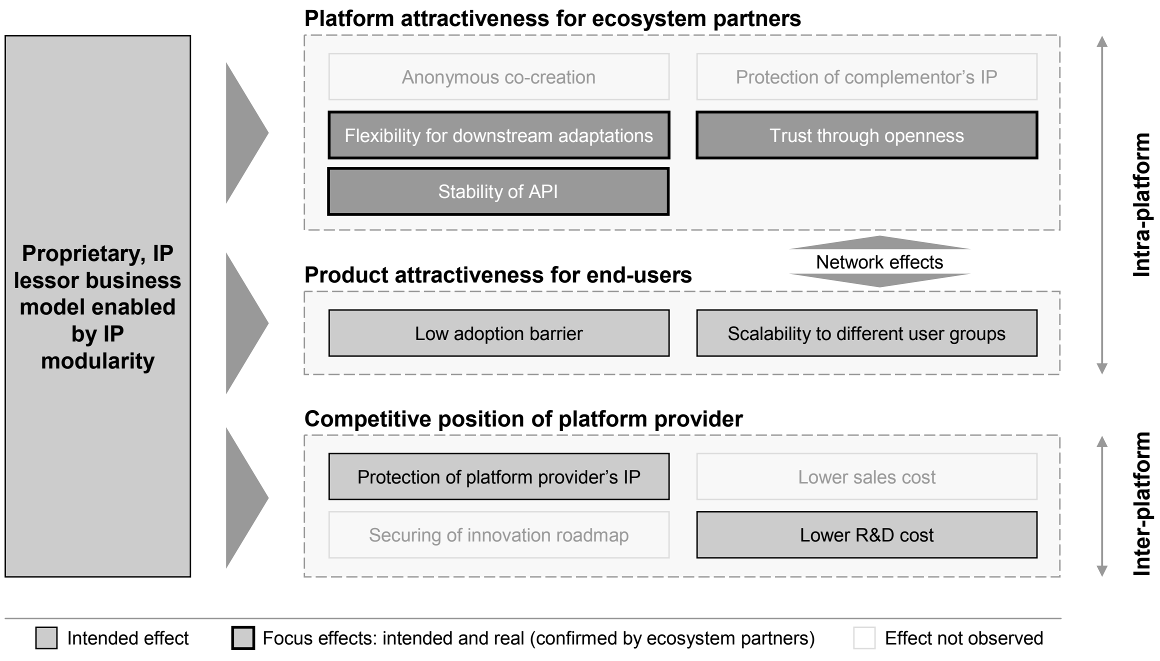

Platform attractiveness for ecosystem partners
Anonymous co-creation
Protection of complementor's IP
Flexibility for downstream adaptations
Trust through openness
Stability of API
Proprietary, IP lessor business model enabled by IP modularity
Network effects
Product attractiveness for end-users
Low adoption barrier
Scalability to different user groups
Competitive position of platform provider
Protection of platform provider's IP
Lower sales cost
Securing of innovation roadmap
Lower R&D cost
Intra-platform
Inter-platform
Intended effect
Focus effects: intended and real (confirmed by ecosystem partners)
Effect not observed

Figure 44 – Intra- and inter-platform effects (Case 9)

For the inter-platform effects, we found that SAP NetWeaver PI's modular design allows the NetWeaver product unit to protect its IP externally and particularly internally towards other SAP units. It also lowers the R&D cost because the platform core can be adapted in a flexible manner not bound to the external API.

As expected for the platform attractiveness, anonymous co-creation could not be observed due to the restrictive IP management and the close management of ecosystem entry with a partner certification program. Notably, we found that the IP modular platform architecture increases the flexibility for downstream adaptations as the responsible general manager for the SAP Netweaver PI development explains:

"Based on the subordinate business targets, we have to ensure that our product is open for adaptations and that it can also be connected to other middleware systems." (Achim Kraiss, General manager for the SAP NetweaverPI development)

The most important effects on platform attractiveness for ecosystem partners is the increased *Stability of the API* and *Flexibility for downstream adaptations* as the checklist matrix shows[28]:

	Role of interviewee			
	General manager Achim Kraiss	Software architect Frank Oliver Hoffmann	Software architect (ISV Partner) John Senor	Software architect (ISV Partner) Robert Schimansky
Platform attractiveness for ecosystem partners				
Anonymous co-creation				
Protection of complementor's IP				
Flexibility for downstream adaptions	x	x	x	x
Trust through openness		x		x
Stability of API	x	x	x	x
Product attractiveness for end users				
Low adoption barrier		x		
Scalability to different user groups		x		
Competitive position of platform provider				
Protection of platform provider's IP	x	x		
Lower sales cost				
Lower R&D cost		x		
Securing of innovation roadmap				

Figure 45 – Intra- and inter-platform effects checklist matrix (Case 9)

For the SAP NetWeaver unit, this stability of the API is a core lever to attract and retain complementors contributing to the NetWeaver PI ecosystem as the responsible general manager explains:

[28] With respect to the specific questions we found that the SAP general manager and the SAP software architect could provide the highest level of detail as they were directly involved in decisions on IP modular platform design. So to keep the results as straight forward as possible we did not include the data from the interviews with the SAP staff function manager and the SAP ecosystem manager in the result display.

“We really have to invest additional effort upfront in platform design to keep the APIs as stable as possible.” (Achim Kraiss, General manager for the SAP NetweaverPI development)

The view on the ecosystem partner side proves that the API stability, enabled by IP modular platform architecture, is a key factor of platform attractiveness as already described by John Senor from the complementor Information Builders Inc. earlier in this section.

With the analysis of the SAP NetWeaver PI case, we show that IP modular platform design can also be beneficial for providers of proprietary software platforms with a restrictive management of IP.

The levers to make such a platform more attractive for ISVs are naturally different from more open platforms, but the stability of the API, enabled by IP modular platform architecture, appears to be an equal lever to increase the trust in the platform provider and increase platform attractiveness.

5.4 Effects of IP modularity in open and proprietary software ecosystems

The cases of SugarCRM and SAP NetWeaver PI illustrate the importance of making IP requirements a key consideration when designing platform architectures. Detailed analysis of our empirical data reveals a number of intra- and inter-platform strategic benefits that result from an IP modular platform design. These benefits relate to (a) platform development, (b) platform attractiveness for complementors and (c) platform attractiveness for end customers.

(a) Both SugarCRM and SAP internally benefit in platform development. SugarCRM can reduce its R&D cost compared to a fully proprietary approach, as large parts of the platform are co-developed by the user community. On the other hand, selected functionality is placed in the proprietary part of SugarCRM’s code tree, increasing the value of the firm’s commercial offerings. In the SAP case, additional flexibility in the development of the core functionality is realized, as the internal API is easier to adapt and does not limit the implementation of additional functionality.

(b) In both cases, the IP modular platform design makes the platform more attractive for complementors. For SugarCRM, adopting the *Community Edition* entails only minimal upfront transaction cost; third-parties can develop complements without any relationship to the platform provider.

In terms of ecosystem access, the closed approach of SAP implies an entry barrier as the ecosystem partners have to certify their products to prove that they are compatible with SAP NetWeaver PI. With this upfront investment for joining the ecosystem, complementors incur a certain economic investment risk that requires SAP to provide risk-mitigating counter-measures to keep the platform attractive for ISVs. Specifically, the IP modular design introduced in version 7.1 enables SAP to keep the API stable, which in turn secures the complementors' investments.

(c) SugarCRM's dual licensing model is a key marketing tool to make the product attractive for end customers. They can download the Community Edition and use it for free. As soon as they require additional functionality that is in one of the professional editions, they obtain the additional modules but do not have to switch to another product. End customers benefit from reduced transaction cost during the first time installation, as well as the migration to professional product versions. Additionally, SAP customers (typically large corporations with a complex IT blueprint) benefit from SAP's IP modular design. They do not require product updates of third-party SAP NetWeaver PI connectors caused by API changes. Furthermore, this also applies to custom-made connectors for legacy systems.

Summarizing, despite rather different levels of platform openness, the need to separate core IP from code to be made accessible to third-parties drives the requirement of an IP modular architecture in both cases.

In line with our findings on IP modularity in software products, we identify IP requirements as an additional instance of non-functional requirements situated on the same level as quality requirements. Because non-functional requirements play a critical role for the architectural design (Wiegers, 2003; Chung and do Prado Leite, 2009), they are reflected in the system architecture, which upon fulfillment of the IP requirements results in being IP modular.

In line with existing literature (Wiegers, 2003), our research shows that business requirements influence the non-functional requirements. In particular, the management of Intellectual Property Rights (IPRs) is directly influenced by the business model. In the case of SugarCRM, the business model requires the encapsulation of core functionality that is excluded from the open source version. In the SAP NetWeaver PI case, the exposure of core functionality via the API is to be prevented. The platform architecture is then defined based on the non-functional IP modularity requirement.

From our analysis of two software platforms, we identify IP modularity to be a key criterion to map a software business model to specific platform architecture. We see a direct link between IP requirements and the technical modularization and architecture that has to be actively managed – confirming our model on IP related requirements shown in Figure 34.

Although based on entirely different business models and different levels of openness, both cases showed that the need to encapsulate platform core IP and to open-up certain other parts of the system eventually led to an IP modular design. In the initial generation of the platform architecture, IP requirements played only a subordinate role. As business requirements gained importance, re-modularization of the initial platform architecture was inevitable. The consideration of IP related requirements in early stages of the platform design could have prevented the additional effort needed to re-modularize the platform in an IP modular fashion.

5.5 Conclusion

By analyzing two widely used software platforms in the enterprise software market, we have carved out the influence of IP related requirements on the modular structure of a platform's architecture. In both cases, IP requirements were more clearly recognized over time as being vital to the respective platform business model. They gained importance accordingly, and eventually gave rise to re-modularizations producing IP modular architectures.

IP modular designs are beneficial in various ways. In the cases we studied, they allow to prevent exposure of core IP and, simultaneously, to practice selective openness. For SAP Netweaver PI, the platform gains attractiveness through stable APIs. For SugarCRM, the open source Community Edition simplifies adoption and invites external contributions.

These benefits can be obtained at almost no additional cost when IP related requirements are considered in the initial platform design, but they entail considerable cost when introduced later through re-modularizations. Thus, software product managers should be aware of that fact and how the firm's strategy and its business model influence the IP requirements of software platforms. IT architects then need to embrace the IP requirements and weigh them against other non-functional and functional requirements. If they succeed, the resulting architecture will balance openness and protection and will be aligned with the software vendor’s business model.

Our research results support software engineers and IT architects in their endeavor of successfully mapping the business model to the platform architecture. Executives, architects and ecosystem managers are provided with a model that links strategic decisions to the architectural requirements. In addition, management field scholars and information systems researchers can build on our research to further investigate the field.

6 The impact of IP modularity on platform attractiveness

This section focuses on the effects of IP modularity on the attractiveness of a software platform for complementors or ISVs. We now switch perspectives from the strategic intentions of a platform provider to the implications for the complementors. We thus investigate whether the strategic intentions that led managers of the platform provider to implement an IP modular platform architecture result in making their platforms more attractive for complementors. Our research on platform attractiveness builds on our findings with respect to SugarCRM and SAP NetWeaver PI, which are discussed in Sections 5.2 and 5.3. We explore platform attractiveness factors by comparing the SugarCRM ecosystem with that of Salesforce.com, which functions in the same industry but is completely different in terms of platform openness and ecosystem management.

Because both ecosystems are populated by a sufficiently large number of complementors, we may apply a hybrid research approach, such as that described in Section 3.1 above, to answer our basic research question:

- How does IP modularity influence the attractiveness of a software platform from a complementor's perspective?

With this research, we extend the results of the Master's thesis of Schreiner (2012), which I initiated and supervised. This thesis is based on our findings on the SugarCRM case presented in Section 5.3 and Waltl *et al.* (2012). As part of the Master's thesis, a platform attractiveness model was developed, a qualitative pre-study of the SugarCRM and the Salesforce.com ecosystems was conducted, and a survey on platform attractiveness was initiated. For the research in this section, we revise and simplify the platform attractiveness model and test the hypotheses on an extended dataset of N=126 with an Ordinary Least Squares (OLS) regression model.

6.1 Platform attractiveness for ecosystem partners

"Platform leaders need to decide on the degree of modularity for their product architectures and the degree of openness of the interfaces to the platform. In particular they must balance openness with how much information about the platform and its interfaces to disclose to potential complementors, who may use this information to become or assist competitors." (Cusumano, 2010, p. 45)

As a vendor opens up for collaboration, complementors require access to platform-specific knowledge. Research shows that providing knowledge and waiving IPRs are key levers for ecosystem growth (Gawer and Henderson, 2007, p. 3; Boudreau, 2010). Unfortunately for the platform provider, waiving IPRs may diminish the ability to appropriate returns (West, 2003).

In addition, Jansen *et al.* (2012, p. 1509) identify platform openness as a key factor generating a vibrant ecosystem by showing that the decision between open and closed is neither black nor white but instead is multifaceted; further research is required to fully understand the implications of this decision for the management of software platform ecosystems. Current research in this nascent field focuses on understanding the interplay of open-platform architecture and platform attractiveness primarily through qualitative methods; however, large-scale quantitative studies are non-existent (Anvaari and Jansen, 2010; Kude *et al.*, 2012, p. 262). To our knowledge, this study[29] offers the first quantitative analysis that links IP modular platform architecture to platform attractiveness for complementors. We base our analysis on a detailed model in which *Platform attractiveness* depends on the *Platform provider setting* and the *Complementor setting*:

[29] Including the work of Schreiner (2012).

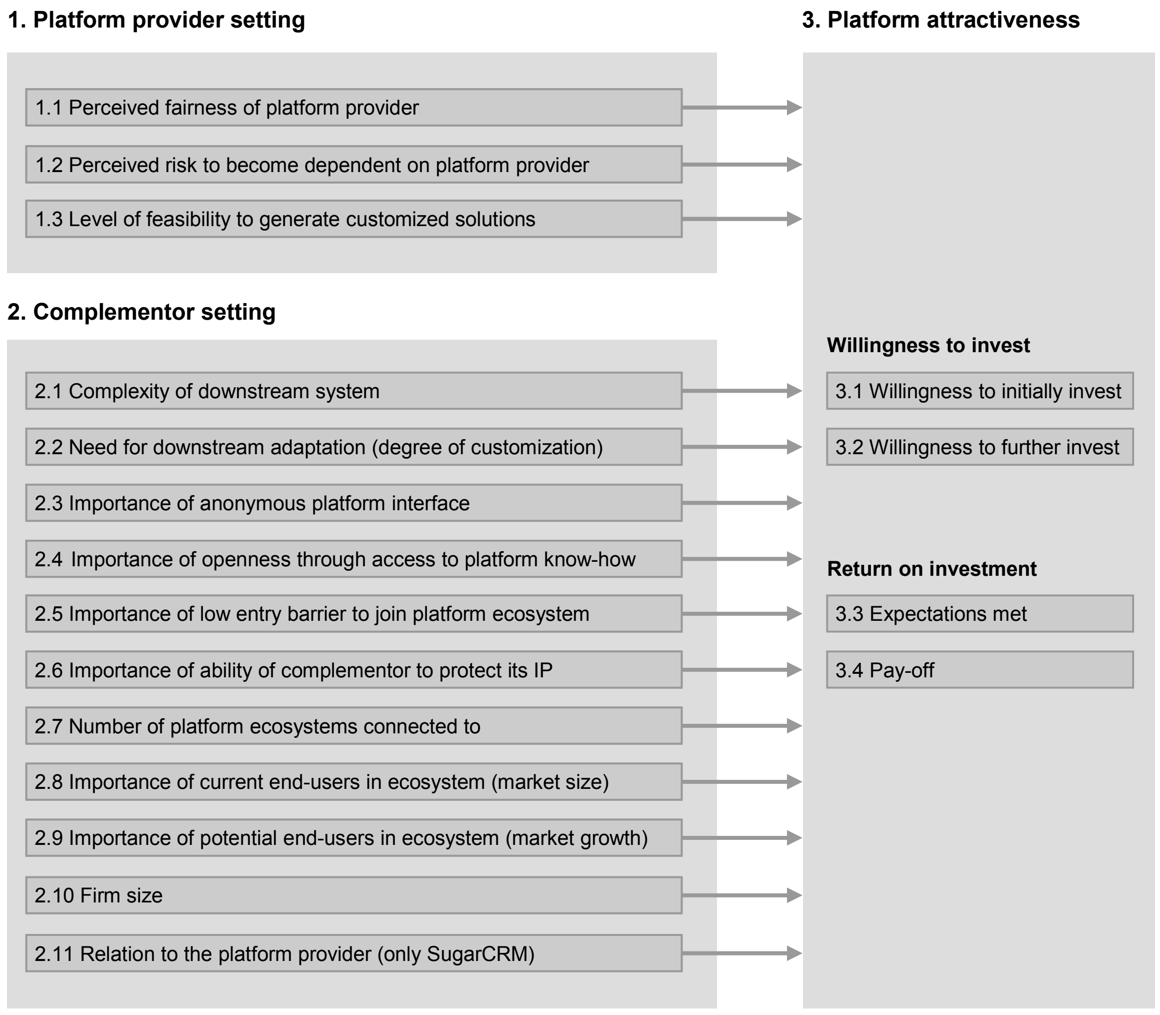

1. Platform provider setting
1.1 Perceived fairness of platform provider
1.2 Perceived risk to become dependent on platform provider
1.3 Level of feasibility to generate customized solutions
2. Complementor setting
2.1 Complexity of downstream system
2.2 Need for downstream adaptation (degree of customization)
2.3 Importance of anonymous platform interface
2.4 Importance of openness through access to platform know-how
2.5 Importance of low entry barrier to join platform ecosystem
2.6 Importance of ability of complementor to protect its IP
2.7 Number of platform ecosystems connected to
2.8 Importance of current end-users in ecosystem (market size)
2.9 Importance of potential end-users in ecosystem (market growth)
2.10 Firm size
2.11 Relation to the platform provider (only SugarCRM)
3. Platform attractiveness
Willingness to invest
3.1 Willingness to initially invest
3.2 Willingness to further invest
Return on investment
3.3 Expectations met
3.4 Pay-off

Figure 46 – Platform attractiveness model (identical with Figure 18)

For the *Platform provider setting*, the variables in the model are *Perceived fairness of platform provider*, *Perceived risk of becoming dependent on platform provider* and *Level of feasibility of generating customized solutions*.

The variable *Perceived fairness of platform provider* is essential for complementors to join a software platform ecosystem. Platform providers and complementors are in a cooperative and competitive relationship simultaneously; platform providers offer the basic technology for complementors to utilize but may also include functionality as part of their package that may make complementors` offerings obsolete. Research into the semiconductor and enterprise software industry has shown that complementors fear being squeezed out by the innovations of the platform provider (Gawer and Henderson, 2007, pp. 21–22; Kude *et al.*, 2012, p. 262) . To overcome this problem, the platform provider must signal to its complementors that they will be treated fairly.

Platform providers must also ensure that for the complementors, the *Perceived risk of becoming dependent on the platform provider* is not too high. The platform provider therefore must show that it is able to further develop the platform and thus support the complementors as they grow (Perrons, 2009, p. 1301). In addition, our own findings with respect to the SugarCRM and SAP NetWeaver PI software platforms show that platform providers must keep their API stable to prevent re-engineering effort on the complementor side (see Sections 5.2 and 5.3).

Software platforms are typically developed to satisfy diverging user needs, e.g., in specific industry verticals, that the platform provider cannot satisfy (Parker and van Alstyne, 2009, p. 32). Therefore, it is vital for complementors to be able to meet these varying user needs. Therefore, a greater *Level of feasibility of generating customized solutions* (see *Proposition 4* in Section 3.2.1) should entail a more attractive platform for a complementor.

For the *Complementor setting,* a total of eleven variables have been identified to influence the attractiveness of a software platform: *Complexity of downstream system, Need for downstream adaptation, Importance of anonymous platform interface, Importance of openness through access to platform know-how, Importance of low entry barrier to join platform ecosystem, Importance of ability of complementor to protect its IP, Number of platform ecosystems connected to, Importance of current end-users in ecosystem (market size), Importance of potential end-users in ecosystem (market growth), Firm size, Relation to the platform provider (only SugarCRM).*

Consistent with *Proposition 4* on outgoing IP modularity, the IP modular platform design may be more attractive for complementors that develop products facing a greater *Complexity of the downstream system*.

In addition, based on *Proposition 4* for outgoing IP modularity, we identify the *Need for downstream adaptation* as variable that may increase the attractiveness of an open platform.

The variable *Importance of anonymous platform interface* refers to the initial transaction costs (Williamson, 1979) to join a platform ecosystem as a complementor. Our research confirms that the ability to enter the SugarCRM ecosystem anonymously with no upfront investment is a key driver inducing complementors to join the ecosystem (see Section 5.2).

The variable *Importance of openness through access to platform know-how* relates to the complementors' evaluation of the know-how that is made available by the platform provider. In extreme cases, such as the SugarCRM platform, the platform provider fully opens up the platform core with the aim of attracting complementors (see Section 5.2).

For complementors of a software platform, the variable *Importance of low entry barrier to join platform ecosystem* should impact the attractiveness of a specific platform because a low entry barrier reduces the complementors' economic risk. Conversely, platforms with a high entry barrier and thus high switching costs are likely to be less attractive to complementors (Farrell and Klemperer, 2007, p. 1972).

A platform provider must ensure that complementors are able to secure their IP (Huang *et al.*, 2009). Therefore, the level that complementors rate their *Ability to protect their IP* should influence the attractiveness of a software platform.

The variable *Number of platform ecosystems a complementor is connected to* plays an important role in platform attractiveness. With this variable, we measure whether a complementor focuses on only one platform or offers its products in several software ecosystems. The qualitative pre-study in the two ecosystems revealed that complementors that find a platform attractive enough do not consider entering additional platform ecosystems (Schreiner, 2012, p. 51). By contrast, complementors that do not see a certain platform ecosystem as attractive may tend to diversify their offerings to other ecosystems.

The *Importance of current end-users in ecosystem* influences the attractiveness of a software platform because of the network effects in the ecosystem (Eisenmann *et al.*, 2006, p. 96); complementors validate the market size as a determinant of whether to enter a certain software platform ecosystem. In addition, the *Importance of potential end-users in ecosystem* influences the attractiveness of a specific software platform, as Schreiner (2012, p. 51) identified in her qualitative pre-study.

With the variable *Firm size*, we control whether the companies differ in size between the SugarCRM and the Salesforce.com ecosystem.

Finally for the complementor setting, the variable *Relation to the platform provider* controls for the transaction costs' entering an ecosystem for complementors by measuring whether they have an anonymous or certified relationship with the platform provider. This variable only applies to the SugarCRM ecosystem, as only SugarCRM offers anonymous co-creation as well as several levels of partner certifications.

The variable of interest in this model, *Platform attractiveness*, cannot be observed directly. To overcome this problem, a set of observable variables (see Figure 46) have been defined based on the findings of the qualitative pre-study of the SugarCRM and Salesforce.com ecosystems.

Utilizing these observable variables, *Platform attractiveness* is calculated. The relationship between the observable variables and *Platform attractiveness* is tested in advance of hypotheses testing using Cronbach`s alpha coefficient (see Section 3.3.2).

For our model on platform attractiveness, the following are the observable variables: *Willingness to initially invest*, *Willingness to further invest*, *Expectations met* and *Pay-off*.

For a full overview of the operationalization of the simplified platform attractiveness model and a basic description of the survey results for each variable, please refer to Appendix F – Appendix H.

We use the described platform attractiveness model to verify the qualitative findings on the impact of SugarCRM's IP modular architecture on its attractiveness for complementors.

To formulate the first set of hypotheses we revisit *Proposition 4* for outgoing IP modularity as described by Henkel and Baldwin (2010) (see also Section 3.2.1):

Proposition 4: [Customization] *The greater and more varied the need for downstream adaptations, the more advantageous is outgoing IP modularization. The module boundaries should separate the IP that serves as the basis for modification from the IP supporting the proprietary "core" modules.*

Our findings on the impact of SugarCRM's IP modular platform architecture on the attractiveness of their platform are fully in line with *Proposition 4* (see Section 5.2 and Waltl *et al.*, 2012):

> *"The reason why our ecosystem partners come to us is because we reduce risk for them. They have more control over the building of products because they have full visibility into the source code, and they understand exactly how the product works." (Clint Oram, CTO and co-founder)*

Transferred to our platform attractiveness model, we expect that a more complex downstream system (variable 2.1) of a complementor and a higher need for downstream adaptation (variable 2.2) for a complementor imply a higher attractiveness of an open software platform. Therefore, the first set of hypotheses may be formulated as follows:

Hypothesis 1$_A$: *We expect the coefficients of variable 2.1 Complexity of downstream system to be larger for SugarCRM than for Salesforce.com.*

Hypothesis 1$_B$: *We expect the coefficients of variable 2.2 Need for downstream adaptation to be larger for SugarCRM than for Salesforce.com.*

The formulation of the second hypothesis relates to *Proposition 2* for outgoing IP modularity as described by Henkel and Baldwin (2010) (see also Section 3.2.1):

Proposition 2: [Distributed Co-creators] *The more distributed, numerous, and anonymous the co-creators of value, the more advantageous is outgoing IP modularity.*

The aspect of the anonymity of co-creators has been fully confirmed by our findings on the SugarCRM ecosystem, as Mirco Müller, CEO of Insignio CRM (one of SugarCRM's largest partners in Europe) describes (see Section 5.2 and Waltl *et al.*, 2012):

> *"In the beginning, we did our business only based on the community edition. SugarCRM did not know us. We started to partner when we acquired our first big customers – still, we did not interact much with SugarCRM. We were able to solve our problems on our own, as we had access to all source code for the community and commercial editions. That changed when Sugar opened an office in Europe, and we now do interact very closely, especially in marketing." (Mirco Müller, CEO of Insignio CRM)*

Based on this observation, we formulate the second hypothesis as follows:

Hypothesis 2: *We expect the coefficient of variable 2.3 Importance of anonymous platform interface to be larger for SugarCRM than for Salesforce.com.*

The platform attractiveness model and the hypotheses represent the basis for the quantitative analysis that will be described in the following section.

6.2 The impact of IP modularity on platform attractiveness – analysis results

In this section, we present the analysis results of our survey on the attractiveness of software platforms for complementors following the defined research process described in Section 3.3 and shown in Figure 17. At the outset, the sample is described and the results of the hypotheses testing are outlined, which is followed by a discussion of the findings in connection with the prior literature and our own qualitative analysis.

As previously discussed in Section 3.1 (see Figure 8), we apply a hybrid research approach in this dissertation with a clear focus on qualitative methods due to the nascent status of the current research on IP modularity in software platform ecosystems. Therefore, we present basic tests of the hypotheses outlined in the previous chapter. Additional analysis will be subject to further research.

We derive our findings from a survey based on the two CRM software ecosystems of SugarCRM and Salesforce.com. The ecosystems differ significantly in their approach to openness. SugarCRM licenses large parts of its platform core as OSS (see Section 5.2), whereas Salesforce.com provides an entirely proprietary software platform in the format of a Software as a Service (SaaS) offering[30] featuring an extensive and well-documented interface. From our findings on SugarCRM, we know that its openness is enabled by its IP modular platform architecture. As outlined in the previous section, this openness should influence the attractiveness of the platform compared to a completely proprietary system. In addition, Salesforce.com experiences IP modularity to a certain degree in that the proprietary API may be accessed to generate complementary applications, whereas the platform core is undisclosed. We observe Salesforce.com to be less IP modular than SugarCRM, which applies an extensive IP modular approach to open the platform core.

SugarCRM and Salesforce.com are considered to be leaders in the CRM market for small- and mid-sized businesses (William Band, 2010). However, the companies differ significantly in size and in the features of their ecosystems, as shown in Table 7.

[30] In the SaaS delivery model, the software runs on the hardware of the software provider. The customer accesses the software over the internet.

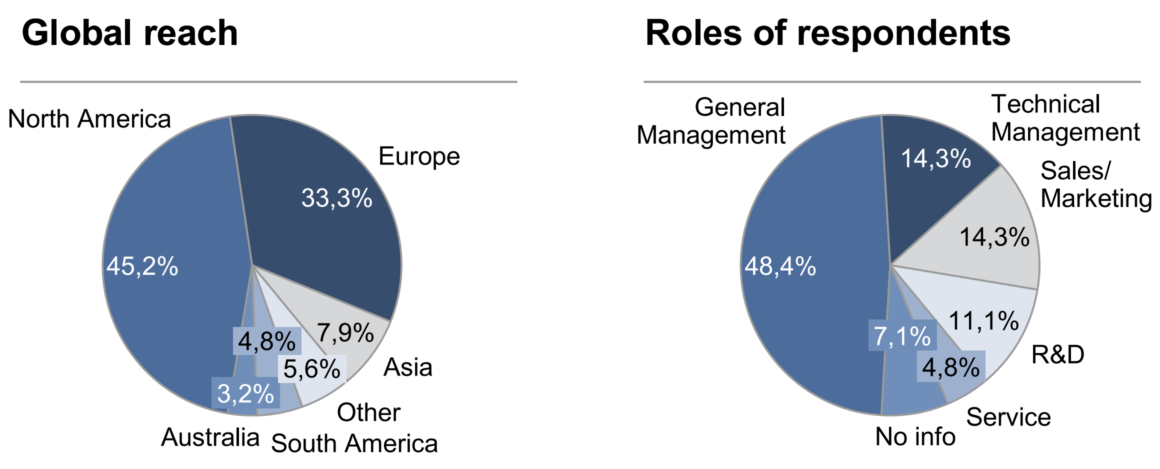
	Salesforce.com		SugarCRM	
Company		Source		Source
Launch	1999	1	2004	2
Number of employees	8,000	3	>250	2
Customers	> 100,000	3	> 7,000	4
Customer growth (CAGR 2010-2011)	19.2 %	1	7.7 %	2
Target customer	Enterprise, Mid, Small	3	Mid, Small	5
Ecosystem				
Application exchange platforms	appexchange.salesforce.com		sugarexchange.com	
Number of applications (2010)	743	6	85	6
Number of complementors (2010)	500	6	73	6

1 www.salesforce.com/company
2 www.sugarcrm.com/company-overview
3 Salesforce.com annual report 2012
4 www.sugarcrm.com/newspress/sugarcrm-expands-revenues-and-new-customer-numbers-pacific-rim
5 (William Band, 2010)
6 (Burkard *et al.* , 2011)

Table 7 – Ecosystem comparison

The survey was conducted using an online questionnaire that was open from June 4, 2012 until August 16, 2012 (see Section 3.3.2). In total, we retrieved answers from 126 ISVs (87 for Salesforce.com and 39 for SugarCRM). Based on prior measurements on both ecosystems (Burkard *et al.*, 2011), the response rate would amount to approximately 18% for Salesforce.com and 53% for SugarCRM. However, this response rate calculation may only be interpreted as an indication for two reasons. First, the information we have from prior research is on the status of 2010; and second, in the SugarCRM case, vendors are not required to provide upfront certification and anonymous co-creation is permitted, making it impossible to draw conclusions about the overall number of applications and complementors. Our research shows that SugarCRM complementors first start anonymously and then cooperate with SugarCRM (see section 5.2). The detailed ecosystem description of Burkard *et al.* (2011) also revealed that most vendors focus on the provision of one application.

The dataset retrieved from the online survey provides considerable variation with respect to global reach, the role of respondents and company size, as shown in Figure 47.



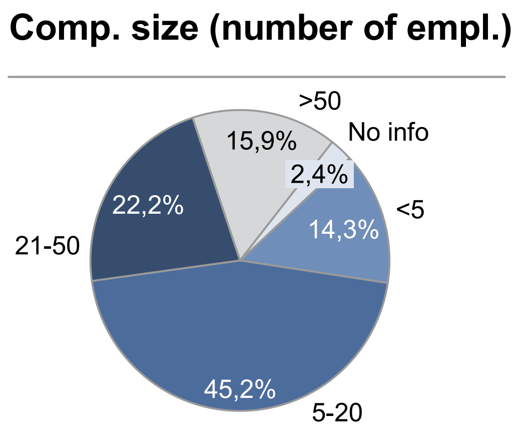


Figure 47 – Sample description (identical with Figure 19)

For additional descriptive statistics about each variable in the dataset, please refer to Appendix F – Appendix H.

Prior to hypotheses testing, the dataset must be prepared. As described in the previous section, *Platform attractiveness* cannot be measured directly. Therefore, we build a factor for each data point *i* by summing up the values of each *Platform attractiveness* variable (see Figure 46) and dividing this sum by the number of variables *N*:

$$Platform\,attractiveness_i = \frac{\sum_{j=1}^{N} Platform\,attractiveness\,variable_j}{N}$$

Figure 48 – Platform attractiveness calculation

As described in Section 3.3.3, the *Platform attractiveness* factor is validated with Cronbach's alpha test to assess internal consistency, as shown in Table 8 a).

	Full sample	Salesforce.com	SugarCRM
a) Platform attractiveness (4 items)	0.67	0.71	0.49
b) Willingness to invest (2 items)	0.72	0.75	0.62
c) Return on investment (2 items)	0.67	0.72	0.35

Table 8 – Cronbach's alpha tests

Higher values for alpha indicate a higher reliability of the construct. Values larger than 0.7 are regarded as satisfactory, but lower thresholds are used in the literature as well (Bland and Altman, 1997, p. 572; Santos J. Reynaldo A., 1999). Table 8, row a) shows that the full sample narrowly misses the 0.7 threshold. For the Salesforce.com subsample, the *Platform attractiveness* factor may be regarded as satisfactory, whereas the alpha value of 0.49 for the smaller SugarCRM subsample is questionable. A possible reason for this may be that the entry barrier in the SugarCRM ecosystem may be zero due to the possibility of anonymous co-creation and thus that the measureable variables *Expectations met* and *Pay-off* are of less importance for the attractiveness of the SugarCRM platform.

Because of the poor result for the SugarCRM subsample, we conducted an exploratory factor analysis for all platform attractiveness variables to search for additional factors that might be generated from the platform attractiveness variable[31]:

[31] For further analysis details please refer to Appendix I.

Factor analysis unrotated

Factor	Eigenvalue	Difference	Proportion	Cumulative
Factor 1	2.04	0.96	0.51	0.51
Factor 2	1.08	0.53	0.26	0.77
Factor 3	0.54	0.20	0.13	0.91
Factor 4	0.33	.	0.08	1.00

Factor loadings (pattern matrix) and unique variances

Variable	Factor 1	Factor 2	Uniqueness
3.1 Willingness to initially invest	0.66	-0.65	0.15
3.2 Willingness to further invest	0.81	-0.35	0.22
3.3 Expectations met	0.67	0.57	0.22
3.4 Pay-off	0.70	0.46	0.30

Factor analysis rotated

Factor	Variance	Difference	Proportion	Cumulative
Factor 1	1.57	0.02	0.39	0.39
Factor 2	1.55	.	0.39	0.78

Rotated factor loadings (pattern matrix) and unique variances

Variable	Factor 1	Factor 2	Uniqueness
3.1 Willingness to initially invest	0.92	0.00	0.15
3.2 Willingness to further invest	0.82	0.32	0.22
3.3 Expectations met	0.08	0.88	0.22
3.4 Pay-off	0.18	0.82	0.30

Table 9 – Exploratory factor analysis

The unrotated analysis shows that *Factor 1* and *Factor 2* have eigenvalues larger than one. According to Kaiser's rule, these factors should be retained for further research (Kaiser, 1960). However, recent research questions the strict applicability of Kaiser's rule (Lance and Vandenberg, 2009, p. 83; Ruscio and Roche, 2012, p. 290). As our research is exploratory, we retain both *Factor 1* and *Factor 2* to ensure that our hypotheses testing shows the entire picture.

In the unrotated case of the factor analysis, the factor loadings do not differ significantly, preventing us from identifying variables with which to build a new factor. The orthogonal rotation of the analysis, however, enables us to identify that the variables *Willingness to initially invest* and *Willingness to further invest* load heavily on *Factor 1*. Similarly, we observe that the variables for *Expectations met* and *Pay-off* load on *Factor 2*.

Using the calculation outlined in Figure 48 for the identified pairs of variables, we are able to build the additional factors *Willingness to invest* and *Return on investment.*

The first factor, *Willingness to invest*, shows significantly better values in the internal consistency check with Cronbach's alpha for the entire sample and the subsamples (see Table 8, row b)). The second factor, *Return on investment*, shows an acceptable alpha value only for the Salesforce.com subsample, as shown in see Table 8, row c)[32].

We begin the interpretation of the obtained dataset with a mean comparison for each variable in the platform attractiveness model and apply a Mann-Whitney-U-Test (Wilcoxon, 1945; Mann and Whitney, 1947) to identify significant differences across the subsamples for Salesforce.com (SFDC) and SugarCRM (SCRM):

		Full sample		SFDC		SCRM		Mann-Whitney
	Variable	**Mean**	**S.D.**	**Mean**	**S.D.**	**Mean**	**S.D.**	**test**
1.	**Platform provider setting**							
1.1	Perceived fairness of platform provider	3.50	1.21	3.37	1.18	3.795	1.218	0.0386*
1.2	Perceived risk to become dependent on platform provider	3.31	1.12	3.52	1.09	2.846	1.065	0.0024**
1.3	Level of feasibility to generate customized solutions	4.62	0.63	4.62	0.63	4.615	0.633	0.9254
2.	**Comlementor setting**							
2.1	Complexity of downstream system	3.48	1.26	3.59	1.19	3.231	1.404	0.1884
2.2	Need for downstream adaptation	3.30	1.35	3.00	1.37	3.974	1.038	0.0002**
2.3	Importance of anonymous platform interface	2.95	1.19	2.86	1.19	3.154	1.159	0.1366
2.4	Importance of openness through access to platform know-how	4.00	1.10	3.77	1.14	4.513	0.79	0.0003**
2.5	Importance of low entry barrier to join platform ecosystem	3.69	1.16	3.81	1.13	3.436	1.209	0.1194
2.6	Ability of complementors to protect its IP	4.33	0.89	4.48	0.76	3.974	1.063	0.0045**
2.7	Number of platform ecosystems connected to	1.94	1.56	1.94	1.68	1.949	1.276	0.2606
2.8	Importance of current end-users in ecosystem (market size)	4.10	0.98	4.22	0.93	3.821	1.023	0.0349*
2.9	Importance of potential end-users in ecosystem (market growth)	4.32	0.90	4.38	0.83	4.179	1.048	0.4112*
2.10	Firm size	3.41	0.93	3.39	0.99	3.436	0.788	0.6495
2.11	Relation to the platform provider (only SCRM)	2.90	0.47			2.667	0.806	
3.	**Platform attractiveness**							
	Platform attractiveness	3.98	0.67	3.92	0.72	4.115	0.512	0.1773
	Willingness to invest	3.93	0.84	3.82	0.85	4.154	0.779	0.0342*
	Return on investment	4.03	0.80	4.01	0.89	4.077	0.545	0.6864

note: ** $p<0.01$, * $p<0.05$

Table 10 – Descriptive statistics

[32] For further details on the factor analysis, please refer to Appendix I.

In the above table, we observe that SugarCRM is perceived as showing higher fairness to its complementors at the 5% significance level. The perceived fairness of a software platform is dependent on the behavior of the platform provider and the terms and conditions under which a complementor may participate in the ecosystem. The moderately higher mean value in the SugarCRM subsample may be an indicator that the open core model (see Section 5.2) increases perceived fairness.

The variable *Perceived risk of becoming dependent on platform provider* shows the opposite pattern at 5% significance level. In line with our qualitative findings on SugarCRM and also on SAP NetWeaver PI (see Sections 5.2 and 5.3), this finding could be interpreted in a way that Salesforce.com's closed platform strategy increases the risk of complementors becoming dependent. An explanation could be that in a closed platform, the provider controls access to the platform core via the API, whereas in an open platform, the API does not limit access to the platform core. The complementors in the closed platform case depend on the platform provider to keep the API broad in functionality and stable across releases. This characteristic could be one of the reasons why Salesforce.com's complementors, on average, perceive the risk of becoming dependent as higher.

For the variable *Level of Feasibility of generating customized solutions,* both platforms appear to provide enough freedom for the developers of complementary applications.

For the variables in the *Complementor setting,* we observe that the *Complexity of the complementary applications (downstream system)* does not significantly differ between the Salesforce.com and the SugarCRM ecosystem.

The *Need for downstream adaptation* is on average much higher for the complementors in the SugarCRM ecosystem, with a mean of 3.974 compared to 3.00 for the complementors of Salesforce.com. Self-selection provides a plausible explanation for the finding that the complementors that generate more customized solutions prefer SugarCRM as platform for their application.

As expected, we find the average *Importance of an anonymous platform interface* to be higher in the SugarCRM subsample. However, the Mann-Whitney-U-Test does not show significant differences between the two subsamples.

In line with our expectations for the variable *Importance of openness,* we observe the mean in the SugarCRM subsample to be significantly larger than the mean in the Salesforce.com subsample. Not only is the difference significant at the 5% level, the absolute value of the mean for SugarCRM is the highest of all complementor setting variables. This finding may also be interpreted as a self-selection effect toward an open platform.

Surprisingly the variable *Importance of low entry barrier to join a platform ecosystem* shows a higher mean value for the Salesforce.com subsample but is not significantly different compared to the SugarCRM subsample. Based on our prior qualitative findings on the SugarCRM case, we would have expected a different outcome. An explanation could be that in the SugarCRM ecosystem, there is no entry barrier, whereas in the Salesforce.com ecosystem, the entry barrier is a critical driver of the complementor's profitability. Nonetheless, additional research is required to fully understand this surprising result.

As expected, we observe that the means for the variable *Importance of ability of complementors to protect its IP* are high in both ecosystems at the 5% significance level. Nonetheless, it is interesting why the mean value is even higher in the Salesforce.com ecosystem, with the difference being significant at the 5% level.

For the rest of the complementor setting variables, the subsamples only differ considerably in the *Importance of current end-users in the ecosystem* – the market size. Here the mean value for the Salesforce.com subsample is moderately higher than that of SugarCRM. An explanation could be the size and the maturity of the Salesforce.com ecosystem as an established market of end-users for the complementors. The mean of the variable *Firm size* does not differ largely in size and is not significantly different between the subsamples. We interpret this finding as a sign that both Salesforce.com and SugarCRM are able to attract firms of comparable size.

The mean values for the calculated platform attractiveness variables are between 3.82 and 4.154. This finding shows that both platforms are relatively attractive for its complementors, which is obvious. The comparison of the subsamples shows a highly interesting pattern, especially for managers of platform provider companies. The mean of all platform attractiveness variables is moderately higher in the SugarCRM subsample. This result could be an indication that the complementors in the SugarCRM subsample on average have a higher willingness to invest in their complements. This result could support software ecosystem managers in their decision regarding how open a specific ecosystem should be and extend existing understanding about platform openness (Boudreau, 2010; West, 2003). However, for two reasons, extensive additional research is required to solidify this result. First, the sample size of the SugarCRM subsample with N=39 may be a factor that reduces the accuracy of our findings, and second, the difference in the mean values is only moderate.

Before testing our hypotheses, we perform a correlation analysis for all variables (measured and calculated) in the platform attractiveness model (see Appendix J). The analysis shows a strong significant correlation between the three platform attractiveness factors, which is obvious. In addition, an examination of the maximal Variance Inflation Factor (VIF) of the analyzed regression models (see Table 11 and Table 12) does not indicate a problem with multicollinearity[33].

For the hypotheses tests, we performed an OLS regression on the full sample as well as the subsamples for Salesforce.com and SugarCRM with all independent model variables (see Figure 46) for *Platform attractiveness*, *Willingness to invest* and *Return on investment* as dependent variables. The results of this first step revealed inacceptable model significance results for the regression analysis with *Willingness to invest* as the dependent variable. Therefore, the regression results on *Willingness to invest* had to be excluded for the hypotheses tests.

In a second step, we excluded the variables *Importance of current end-users in ecosystem (market size), Importance of potential end-users in ecosystem (market growth)* and *Firm size*, which have not been significant in the first round, and repeated the analysis. Table 11 and Table 12 show the regression results including the F-Test results for the excluded variables.

[33] We consider a regression model to be acceptable when the maximal VIF value is less than ten. However, it must be mentioned that this condition does not exclude the possibility of multicollinearity, as discussed in recent research by Echambadi *et al.* (2006) and Echambadi and Hess (2007).

Platform attractiveness		**Full sample**		**SFDC**		**SCRM**	
	Variables	Coeff.	Std. err.	Coeff.	Std. err.	Coeff.	Std. err.
1.1	Perceived fairness of platform provider	0.191***	0.048	0.246***	0.062	0.047	0.074
1.2	Perceived risk to become dependent on platform provider	0.057	0.052	0.117*	0.068	0.034	0.080
1.3	Level of feasibility to generate customized solutions	0.177**	0.088	0.170	0.118	0.254*	0.136
2.1	Complexity of downstream system	0.118***	0.046	0.127**	0.063	0.089	0.085
2.2	Need for downstream adaptation (degree of customization)	0.082*	0.043	0.051	0.054	0.072	0.120
2.3	Importance of anonymous platform interface	-0.011	0.047	-0.007	0.061	0.074	0.085
2.4	Importance of openness through access to platform know-how	-0.026	0.051	-0.066	0.062	0.117	0.124
2.5	Importance of low entry barrier to join platform ecosystem	0.016	0.049	0.016	0.064	0.016	0.080
2.6	Ability of complementors to protect its IP	-0.073	0.061	0.044	0.092	-0.177*	0.090
2.7	Number of platform ecosystems connected to	-0.073**	0.034	-0.077*	0.041	-0.003	0.068
2.11	Relation to the platform provider	0.066	0.125			0.193	0.122
Number of observations		124			85		39
F statistic		4.99			4.210		2.240
Prob > F		0.000			0.000		0.043
R-squared		0.329			0.363		0.477
Adjusted R-squared		0.2631			0.277		0.264
Root mean squared error		0.5731			0.614		0.440
VIF (Max)		1.31			1.250		3.060
F-Test for excluded variables		0.2771			0.4573		0.5673

note: *** p<0.01, ** p<0.05, * p<0.1

Table 11 – OLS regression: Platform attractiveness

The regression models for *Platform attractiveness* as the dependent variable are all significant at the 5% level, and the corresponding F-Tests for the dropped variables are insignificant.

Return on investment		**Full sample**		**SFDC**		**SCRM**	
Variables		Coeff.	Std. err.	Coeff.	Std. err.	Coeff.	Std. err.
1.1	Perceived fairness of platform provider	0.264***	0.054	0.356***	0.073	0.052	0.066
1.2	Perceived risk to become dependent on platform provider	0.054	0.060	0.082	0.080	0.073	0.071
1.3	Level of feasibility to generate customized solutions	0.346***	0.101	0.381***	0.138	0.390***	0.122
2.1	Complexity of downstream system	0.048	0.052	0.055	0.073	-0.056	0.077
2.2	Need for downstream adaptation (degree of customization)	0.112**	0.049	0.067	0.063	0.297***	0.108
2.3	Importance of anonymous platform interface	-0.033	0.054	-0.029	0.072	0.132*	0.076
2.4	Importance of openness through access to platform know-how	-0.045	0.058	-0.059	0.073	0.037	0.111
2.5	Importance of low entry barrier to join platform ecosystem	0.034	0.056	0.009	0.075	0.013	0.071
2.6	Ability of complementors to protect its IP	-0.029	0.070	0.043	0.107	-0.160**	0.081
2.7	Number of platform ecosystems connected to	-0.082**	0.039	-0.091*	0.048	0.004	0.061
2.11	Relation to the platform provider	0.134	0.143			0.255**	0.110
Number of observations			124		85		39
F statistic			6.390		5.650		4.130
Prob > F			0.000		0.000		0.001
R-squared			0.386		0.433		0.627
Adjusted R-squared			0.325		0.356		0.475
Root mean squared error			0.657		0.718		0.394
VIF (Max)			1.310		1.250		3.060
F-Test for excluded variables			0.284		0.5216		0.375

note: *** p<0.01, ** p<0.05, * p<0.1

Table 12 – OLS regression: Return on investment

Table 12 indicates that the regression models for *Return on investment* as dependent variables are all significant at the 1% level and that the corresponding F-Tests for the dropped variables are insignificant.

Given the characteristics of the conducted regression analyses, we may conclude that the models for *Platform attractiveness* and for *Return on investment* as dependent variables may be applied for further interpretation.

The results for the full sample shown in Table 11 and Table 12 provide valuable insights for the existing research on the attractiveness of software platforms. In the two regression models, *Platform attractiveness* and the expected *Return on investment, respectively,* are significantly positively influenced by the variables: *Perceived fairness of platform provider* and *Level of feasibility of generating customized solutions*. Software ecosystem managers in particular may find these findings helpful to focus their effort and prioritize their resources. Interestingly, *Fairness to complementors* appears to be only relevant for complementors of the proprietary Salesforce.com ecosystem and appears to be less important for SugarCRM complementors, which have access to the full platform code. Our findings may be an indication that active and fair ecosystem management are of higher relevance for providers of proprietary, closed source platforms.

The conducted analysis adds to the existing quantitative research on platform attractiveness (Boudreau, 2010). The results on the variable *Perceived fairness of platform provider* confirm prior qualitative findings from Gawer and Henderson (2007, pp. 21–22) and Kude *et al.* (2012, p. 262) on the importance of platform provider fair-play for the complementors.

The formulation of the research hypotheses requires a comparison between the regression models for the Salesforce.com and SugarCRM subsample, as we wish to carve out the difference between an open platform (SugarCRM) and a closed proprietary platform (Salesforce.com). Table 13 displays the research hypotheses and the relevant regression results for further interpretation.

		Platform attractiveness				**Return on investment**			
		SFDC		**SCRM**		**SFDC**		**SCRM**	
Variables	**Hypo-theses**	Coeff.	Std. err.	Coeff.	Std. err.	Coeff.	Std. err.	Coeff.	Std. err.
2.1 Complexity of downstream system	$H\ 1_A$	0.127**	0.063	0.089	0.085	0.055	0.073	-0.056	0.077
2.2 Need for downstream adaptation	$H\ 1_B$	0.051	0.054	0.072	0.120	0.067	0.063	0.297***	0.108
2.3 Importance of anonymous platform interface	H 2	-0.007	0.061	0.074	0.085	-0.029	0.072	0.132*	0.076

note: *** $p<0.01$, ** $p<0.05$, * $p<0.1$

Table 13 – Hypotheses tests

Hypothesis 1_A is not supported by our results because the coefficient of the variable *Complexity of downstream system* is not larger in the regression analysis for the SugarCRM subsample.

For *Hypothesis 1_B*, both coefficients show greater values for the SugarCRM subsample. For the regression with *Return on investment* as the dependent variable, the coefficient is considerably larger than the coefficient in the Salesforce.com subsample with significance at the 1% level. The coefficients differ between the regression analyses of the subsamples at the 5% level (p=0.012; see Appendix K). These findings may be interpreted as an indication that *Hypothesis 1_B* could be supported. However, these results must be viewed in light of the limited dataset for the SugarCRM subsample, and further quantitative research is required to validate these findings.

For *Hypothesis 2,* the coefficients for the variable *Importance of anonymous platform interface* are moderately larger for SugarCRM in both regression models. As expected, this coefficient is close to zero for the Salesforce.com subsample. For the regression analysis with *Willingness to Invest* as the dependent variable, the coefficients differ at the 5% significance level (p=0.026, see Appendix K). This finding may be interpreted as indication that *Hypothesis 2* could be supported. However, we find a significant effect for only one construct measuring platform attractiveness (*Return on Investment).* Therefore, additional quantitative research is required to validate this finding.

A limitation of the above analysis is that it is based on the relatively small SugarCRM ecosystem, resulting in a small subsample size of N=39. Further research would ideally compare another open code, IP modular software platform with a closed, proprietary platform to shed additional light on our results.

To summarize our findings, our tests on *Hypothesis 2* provide indications that *Proposition 2* as formulated by Henkel and Baldwin (2010) (see Section 3.2.1) could correctly describe when outgoing IP modularity is advantageous. For *Proposition 4,* the conflicting results for the tests on *Hypothesis 1_A* and *Hypothesis 1_B* open up the field for further research.

6.3 Conclusion

In this section, we analyzed the consequences of an IP modular platform architecture on how complementors find and evaluate the attractiveness of a software platform. We base our findings on a survey on two rather different software platform ecosystems. The first is SugarCRM, with a widely open core enabled by its IP modular platform architecture; the second is Salesforce.com, with a completely proprietary approach and only the interface specifications being "open" (but still IP protected).

We tested the hypotheses for the developed platform attractiveness model based on a quantitative study in both ecosystems. An additional factor analysis for the dependent variable revealed *Return on investment* as a separate factor and an additional dependent variable to measure platform attractiveness.

The hypotheses tests provide indications that complementors with a higher need to customize their solutions evaluate the open-core SugarCRM platform as more attractive. Based on our prior qualitative findings, we would also have expected that complementors with more complex applications evaluate the SugarCRM platform as more attractive, but could not observe such an effect. In line with our qualitative findings, we find indications that *Proposition 4 [Customization]* from Henkel and Baldwin (2010) could correctly describe when outgoing IP modularity is advantageous. Further research is required to fully understand the conflicting results and the mechanisms of complement customization, complexity and related platform attractiveness.

For *Proposition 2 [Distributed Co-creators],* our findings provide indications that it correctly describes when outgoing IP modularity is advantageous, particularly for the regression model with *Return on investment* as dependent variable.

Because our quantitative analysis is based on the relatively small SugarCRM ecosystem resulting in a small subsample size of N=39, the results may only provide indications if our hypotheses can be supported. Further research with larger samples sizes would be required to clearly accept or reject our hypotheses.

Finally, our findings indicate that *Fairness to complementors* appears to be highly important for *Platform attractiveness* and perceived *Return on invest* in proprietary, closed software platform ecosystems. We could not observe such an effect for the open SugarCRM platform. These findings are especially relevant for ecosystem managers of proprietary, closed platforms, emphasizing the importance of a clear and fair rule set for complementors to make such a platform attractive.

7 Conclusion

This dissertation is the first empirical study to investigate the impact of IP modularity in the software domain. Based on a hybrid research approach (see Section 3), the study generates insights on the effects of IP modular software architecture from three different perspectives. First, there is the perspective of software product developers (Section 4). Second, there is the perspective of software platform providers (Section 5). And third, our research takes the perspective of complementors that contribute to a software platform ecosystem (Section 6).

Our findings on the impact of IP modular software product design are based on two case studies. For outgoing IP modularity, we analyzed an engineering software product. We found that the main reason for the IP modular architecture is the protection of the IP that differentiates the product from those of competitors. For incoming IP modularity, we derived our findings from a data management product case. We found that the IP modular product structure prevents the uncontrolled diffusion of third-party IP into the core modules of the product. Based on a cross-case comparison, we identified that an early inclusion of IP considerations in the requirements engineering process would have prevented a time- and cost-intensive re-modularization. In addition, the IP modular software product architecture reduces a firm's exposure to legal risks.

For the research on the perspective of a software platform provider, our results are based on the analysis of an open and a closed software platform. For the open SugarCRM platform, its IP modular architecture is the fundamental enabler of its commercial open-source business model. For the case of SugarCRM, a variety of effects related to IP modularity could be observed. In particular, SugarCRM's IP modular architecture aims to increase platform attractiveness through the possibility of anonymous co-creation and extensive possibilities to generate customized solutions – including modifications of the platform core. In the case of the closed, proprietary NetWeaver PI platform, SAP aims to increase attractiveness for its complementors through API stability, which is enabled by its IP modular platform architecture. In both cases, IP requirements were more clearly recognized over time as being vital to the respective platform business model. They gained importance accordingly and eventually gave rise to re-modularizations producing IP modular architectures.

Following the hybrid research approach, the analysis of the platform complementor's perspective is based on a quantitative survey in two software platform ecosystems in the

CRM software industry. First, SugarCRM, with the open platform core approach enabled by its IP modular platform architecture; and second, Salesforce.com with a closed, proprietary approach.

The developed hypotheses are based on qualitative findings on platform attractiveness and *Propositions 2 [Distributed Co-creators]* and *Proposition* 4 *[Customization]* for outgoing IP modularity described by Henkel and Baldwin (2010). Our results indicate that for SugarCRM complementors, a high need to customize their solutions positively influences the attractiveness of the open SugarCRM platform. We also see indications that the SugarCRM complementors that rate an anonymous platform interface as important rate the attractiveness of the SugarCRM platform more highly. These results provide indications that the quantitative analysis confirms our qualitative findings and prior research by Henkel and Baldwin (2010). However, our analysis on the relatively small SugarCRM ecosystem is based on a sample size of N=39; therefore, further research is required to confirm our findings.

Our analysis also revealed that fairness to complementors appears to be highly important in the proprietary, closed Salesforce.com platform ecosystem, whereas we could not observe such an effect for the open SugarCRM platform. As the complementors of both ecosystems SugarCRM and Salesforce.com rate their platform attractive, our results could be interpreted in a way that a clear and fair rule set for complementors could substitute for the lack of openness for platform attractiveness.

With our empirical findings, we link existing research on IP modularity, software platforms, multi-sided markets, software ecosystems, software business models and software requirements engineering. However, our research represents only the starting point for further investigations on the impact of IP modularity on software products and software platform ecosystems. In particular, our quantitative research on the platform attractiveness for complementors may connect the upcoming field of software ecosystem research with the concept of IP modularity. Additional scientific effort in this direction offers the potential to prove this connection and to solidify our findings.

Appendices

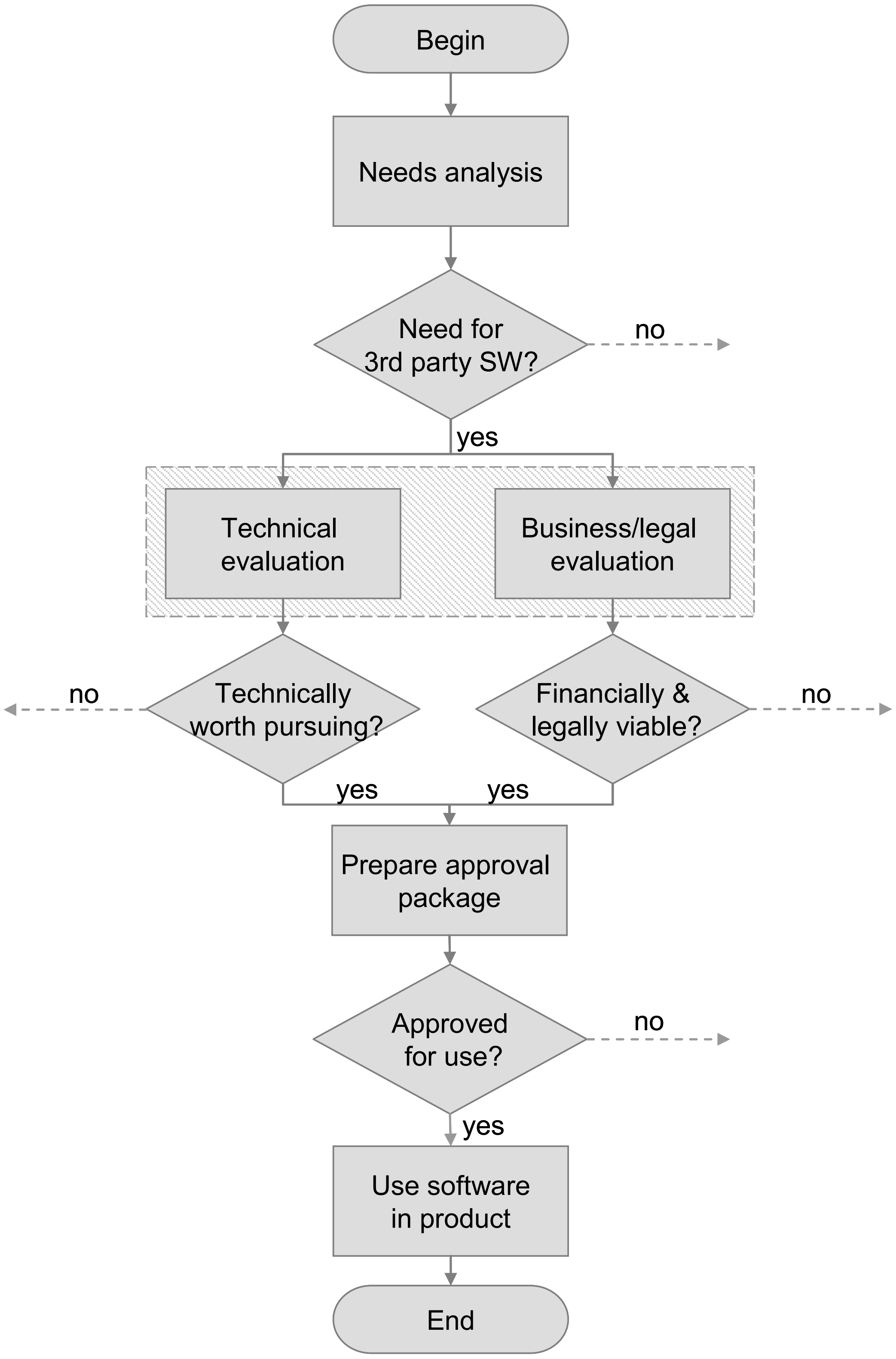
Appendix A – Final coding scheme (Case 2)

Research framework	Root node	Sub node 1	Sub node 2	Sub node 3
A - Intended effects				
	Q09 - Intended effects			
		IP-related		
				Prevention of reverse engineering
				Secrecy
				Cost reduction for IP enforcement
				Prevention of security risks
		Other		
			Run time	
				Performance: speed
				Memory consumption
			Design time	
				Re-use
				Technical reasons
				Faster implementation
				Availability of labor
				Organization: division of work
	Q10 - Effects ranking			
		IP-related		
				Prevention of reverse engineering
				Secrecy
				Cost reduction for IP enforcement
				Prevention of security risks
		Other		
			Run time	
				Performance: speed
				Memory consumption
			Design time	
				Re-use
				Technical reasons
				Faster implementation
				Availability of labor
				Organization: division of work
	Q11 - Strategic advantages			
B - Other				
	Q12 - Non-IP modularization			
		Run time		
				Performance: Speed
				Memory consumption
		Design time		
				Re-use
				Cost reduction: faster development
				Cost reduction: reduced testing
		Maintenance		
	Q13 - Interrelations IP- and other drivers			
D - Real				
	Q14 - PF performance indicators			
		IP-protection		
		Requirements fulfillment		
			Functional	
			Non-functional	
				Performance: speed
				Memory consumption
				Stability
	Q15 - Impact on performance			
	Q16 - Impact on risk mitigation			

Appendix B – Final coding scheme (Case 4)

Research framework	Root node	Sub node 1	Sub node 2
A - Intended effects			
	Q09 - Intended effects		
		IP-related	
			Prevent uncontrolled use of code
			Reduced legal risk: own company
			Reduced legal risk: customers
		Other	
			Run time: improved reliability
			Design time: ease of design
			Maintenance: cost reduction
	Q11 - Strategic advantages		
B - Other effects			
	Q12 - Non-IP modularization drivers		
		IP-related	
			Prevent uncontrolled use of code
			Reduced legal risk: own company
			Reduced legal risk: customers
		Other	
			Run time: improved reliability
			Design time: ease of design
			Maintenance: cost reduction
	Q13 - Interrelations IP- and other drivers		
D - Real effects			
	Q14 - Platform performance		
	Q15 - Modularization impact on		
	Q16 - Modularization		
		IP-related	
			Prevent uncontrolled use of code
			Reduced legal risk: own company
		Other	
			Design time: ease of design

Appendix C – Approval process for third-party software (Case 4)



Appendix D – Final coding scheme (Case 11)

Head node		Sub node 1	Sub node 2	Sub node 3
A Intended effects				
	Q 09	Intended effects		
			AA Business model	
			AA Optimized design	
			Competitive position of platform provider	
				Lower R&D cost through agile development
				Lower sales cost
				Protection of platform provider's IP
				Secure innovation roadmap
			Platform attractiveness for ecosystem partners	
				Anonymous Co-Creation
				Flexibility for downstream adaptations
				Trust through openness (Platform attractiveness)
				Protection on complementor's IP
			Product attractiveness for end-users	
				Low adoption barrier for customers
				Scalibility to different user groups (e.g. Enterprise customers)
			Proposition 2 - Distributed Co-Creators	
	Q 10	Intended effect ranking		
	Q 11	Strategic advantages2		
			Ecosystem growth	
B Other effects				
	Q 12	Other drivers for particular modularization		
D Real effects				
	Q 14	Definition of product performance		
	Q 15	Effect on product performance		
			Cost position & pricing	
			Maintainability	
			Market Share (Verbreitung)	
			Profit position	
	Q 16	Risk prevention		

Appendix E – Final coding scheme (Case 9)

Head node	Sub node 1		Sub node 2	Sub node 3
A Intended effects				
	Q09	Intended effects		
			Competitive position of platform provider	
				Lower R&D cost through agile development
				Lower sales cost
				Protection of platform provider's IP
				Secure innovation roadmap
			Platform attractiveness for ecosystem partners	
				Anonymous Co-Creation
				Flexibility for downstream adaptations
				Trust through openness (Platform attractiveness)
				Protection on complementor's IP
			Product attractiveness for end-users	
				Low adoption barrier for customers
				Scalibility to different user groups (e.g. Enterprise customers)
			Other	
				Traceability of idea/inventor affiliation
	Q10	Intended effect ranking		
	Q11A	Strategic advantages		
	Q11B	Effects on platform attractiveness		
B Other effects				
	Q12	Non-IP modularization drivers		
D Real effects				
	Q14	Platform performance indicators		
	Q15	Impact on success and ecosystem growth		
	Q16	Modularization impact on risk mitigation		

Appendix F – 1. Platform provider setting

```
variable v_233 "1.1 Perceived fairness of platform provider"
Operationalization: We are confident that SCRM/SFDC will always treat us fairly
Possible answers: [1 disagree - 5 agree]

Frequencies:
        1.1 |
  Perceived |
fairness of |
   platform |
   provider |      Freq    Percent        Cum.
------------+-----------------------------------
          0 |         1       0.79        0.79
          1 |        10       7.94        8.73
          2 |        11       8.73       17.46
          3 |        35      27.78       45.24
          4 |        41      32.54       77.78
          5 |        28      22.22      100.00
------------+-----------------------------------
      Total |       126     100.00


Mean:
Mean estimation                    Number of obs    =     126

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_233 |        3.5    .107349      3.287543    3.712457
--------------------------------------------------------------
```

```
variable v_234 "1.2 Perceived risk to become dependent on platform provider"
Operationalization: How would you assess the risk in the SCRM/SFDC ecosystem to become
dependent on the platform provider?
Possible answers: [1 very low level of risk - 5 very high level of risk]

Frequencies:
        1.2 |
  Perceived |
    risk to |
     become |
  dependent |
on platform |
   provider |      Freq    Percent        Cum.
------------+-----------------------------------
          1 |         8       6.35        6.35
          2 |        16      12.70       19.05
          3 |        56      44.44       63.49
          4 |        21      16.67       80.16
          5 |        25      19.84      100.00
------------+-----------------------------------
      Total |       126     100.00


Mean:
Mean estimation                    Number of obs    =     126

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_234 |   3.309524   .0998184      3.111971    3.507077
--------------------------------------------------------------
```

variable v_231 "1.3 Level of feasibility to generate customized solutions"
Operationalization: Please indicate how strongly you agree or disagree with the following statements about SCRM/SFDC: It is technically feasible to generate customized solutions
Possible answers: [1 disagree - 5 agree]

```
Frequencies:
  1.3 Level |
         of |
feasibility |
to generate |
 customized |
  solutions |      Freq    Percent        Cum.
------------+-----------------------------------
          2 |         3       2.38        2.38
          3 |         1       0.79        3.17
          4 |        37      29.37       32.54
          5 |        85      67.46      100.00
------------+-----------------------------------
      Total |       126     100.00


Mean:
Mean estimation                     Number of obs    =     126

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_231 |   4.619048   .0561824      4.507856     4.73024
--------------------------------------------------------------
```

Appendix G – 2. Complementor setting

variable: v_258 "2.1 Complexity of downstream system"

```
Operationalization: How many components does your product comprise?
Possible answers:  [1 Low number of components (e.gconnector) - High number of
components (e.gsystem integrator)5]

Frequencies:
        2.1 |
 Complexity |
         of |
 downstream |
     system |      Freq    Percent        Cum.
------------+-----------------------------------
          1 |         8       6.35        6.35
          2 |        23      18.25       24.60
          3 |        33      26.19       50.79
          4 |        25      19.84       70.63
          5 |        37      29.37      100.00
------------+-----------------------------------
      Total |       126     100.00

Mean:
Mean estimation                     Number of obs    =     126

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_258 |    3.47619   .1125261      3.253487    3.698894
--------------------------------------------------------------
```

variable: v_259 "2.2 Need for downstream adaptation"

```
Operationalization: How customized are the products in your portfolio for your
customers?
Possible answers:  [1 Low customization (standardized product) - 5 High customization
(customized solution)]

Frequencies:
   2.2 Need |
        for |
 downstream |
 adaptation |
 (degree of |
customizati |
        on) |      Freq    Percent        Cum.
------------+-----------------------------------
          1 |        18      14.40       14.40
          2 |        17      13.60       28.00
          3 |        28      22.40       50.40
          4 |        33      26.40       76.80
          5 |        29      23.20      100.00
------------+-----------------------------------
      Total |       125     100.00

Mean:
Mean estimation                     Number of obs    =     125

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_259 |      3.304     .12087      3.064764    3.543236
--------------------------------------------------------------
```

variable v_253 "2.3 Importance of anonymous platform interface"
Operationalization: Please indicate which of the following factors have an influence on the attractiveness of a software platform from your point of view: Anonymous platform interface
Possible answers: [1 not important - 5 highly important]

```
Frequencies:
        2.3 |
 Importance |
         of |
  anonymous |
   platform |
  interface |      Freq     Percent        Cum.
------------+-----------------------------------
          0 |         3        2.38        2.38
          1 |        12        9.52       11.90
          2 |        23       18.25       30.16
          3 |        52       41.27       71.43
          4 |        22       17.46       88.89
          5 |        14       11.11      100.00
------------+-----------------------------------
      Total |       126      100.00

Mean:
Mean estimation                     Number of obs    =     126

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_253 |   2.952381   .1056242      2.743338    3.161424
--------------------------------------------------------------
```

variable v_248 "2.4 Importance of openness through access to platform know-how"
Operationalization: Please indicate which of the following factors have an influence on the attractiveness of a software platform from your point of view: Openness: Access to know-how and source code of the platform provider
Possible answers: [1 not important - 5 highly important]

```
Frequencies:
        2.4 |
 Importance |
of openness |
    through |
  access to |
   platform |
   know-how |      Freq     Percent        Cum.
------------+-----------------------------------
          1 |         4        3.17        3.17
          2 |         7        5.56        8.73
          3 |        30       23.81       32.54
          4 |        29       23.02       55.56
          5 |        56       44.44      100.00
------------+-----------------------------------
      Total |       126      100.00

Mean:
Mean estimation                     Number of obs    =     126

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_248 |          4     .09759      3.806857    4.193143
--------------------------------------------------------------
```

variable v_247 "2.5 Importance of low entry barrier to join platform ecosystem"
Operationalization: Please indicate which of the following factors have an influence on the attractiveness of a software platform from your point of view: Low entry barrier
Possible answers: [1 not important - 5 highly important]

```
Frequencies:
        2.5 |
 Importance |
     of low |
      entry |
 barrier to |
       join |
   platform |
  ecosystem |      Freq    Percent        Cum.
------------+-----------------------------------
          0 |         2       1.59        1.59
          1 |         4       3.17        4.76
          2 |         9       7.14       11.90
          3 |        39      30.95       42.86
          4 |        34      26.98       69.84
          5 |        38      30.16      100.00
------------+-----------------------------------
      Total |       126     100.00


Mean:
Mean estimation                    Number of obs    =     126

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_247 |   3.690476   .1035646      3.485509    3.895443
--------------------------------------------------------------
```

variable v_252 "2.6 Importance of ability of complementors to protect its IP"
Operationalization: Please indicate which of the following factors have an influence on the attractiveness of a software platform from your point of view: Ability to protect your intellectual property
Possible answers: [1 not important - 5 highly important]

```
Frequencies:
2.6 Ability |
         of |
complemento |
      rs to |
protect its |
         IP |      Freq    Percent        Cum.
------------+-----------------------------------
          0 |         1       0.79        0.79
          2 |         1       0.79        1.59
          3 |        23      18.25       19.84
          4 |        31      24.60       44.44
          5 |        70      55.56      100.00
------------+-----------------------------------
      Total |       126     100.00

Mean:
Mean estimation                    Number of obs    =     126

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_252 |   4.325397   .0795458      4.167966    4.482828
--------------------------------------------------------------
```

variable v_206 "2.7 Number of platform ecosystems connected to"

```
Operationalization: On how many platforms (CRM and other platforms) do you offer your
products?
Possible answers:  [1-5]

Frequencies:
 2.7 Number |
of platform |
 ecosystems |
  connected |
         to |      Freq    Percent        Cum.
------------+-----------------------------------
          1 |        77      61.11       61.11
          2 |        22      17.46       78.57
          3 |         9       7.14       85.71
          4 |         4       3.17       88.89
          5 |         3       2.38       91.27
         >5 |        11       8.73      100.00
------------+-----------------------------------
      Total |       126     100.00

Mean:
Mean estimation                    Number of obs    =     126

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_206 |   1.944444   .1390697      1.669208    2.219681
--------------------------------------------------------------
```

variable v_275 "2.8 Importance of current end-users in ecosystem (market size)"

```
Operationalization: Please indicate which of the following factors have an influence on
the attractiveness of a software platform from your point of view: Number of current
end-users (market size)
Possible answers:  [1 not important - 5 highly important]

Frequencies:
        2.8 |
 Importance |
 of current |
  end-users |
         in |
  ecosystem |
    (market |
      size) |      Freq    Percent        Cum.
------------+-----------------------------------
          2 |         8       6.35        6.35
          3 |        30      23.81       30.16
          4 |        30      23.81       53.97
          5 |        58      46.03      100.00
------------+-----------------------------------
      Total |       126     100.00


Mean:
Mean estimation                    Number of obs    =     126

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_275 |   4.095238   .0868705      3.923311    4.267166
--------------------------------------------------------------
```

variable v_276 "2.9 Importance of potential end-users in ecosystem (market growth)"
Operationalization: Please indicate which of the following factors have an influence on the attractiveness of a software platform from your point of view: Number of potential end-users (market size)
Possible answers: [1 not important - 5 highly important]

```
Frequencies:
        2.9 |
 Importance |
         of |
  potential |
  end-users |
         in |
  ecosystem |
    (market |
    growth) |      Freq    Percent        Cum.
------------+-----------------------------------
          2 |         7       5.56        5.56
          3 |        16      12.70       18.25
          4 |        33      26.19       44.44
          5 |        70      55.56      100.00
------------+-----------------------------------
      Total |       126     100.00

Mean:
Mean estimation                     Number of obs    =     126

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_276 |    4.31746   .0801988      4.158737    4.476184
--------------------------------------------------------------
```

Variable: v_263 "2.10 Firm size"
Operationalization: How many employees does your company have?
Possible answers: [1 <5 - 5 >50]

```
Frequencies:
  2.10 Firm |
       size |      Freq    Percent        Cum.
------------+-----------------------------------
         <5 |        18      14.63       14.63
       5-20 |        57      46.34       60.98
      21-50 |        28      22.76       83.74
        >50 |        20      16.26      100.00
------------+-----------------------------------
      Total |       123     100.00

Mean:
Mean estimation                     Number of obs    =     123

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_263 |   3.406504   .0839176      3.240381    3.572627
--------------------------------------------------------------
```

variable v_226 "2.11 Relation to platform provider (only SugarCRM)"

```
Please specify your current contract with SCRM.
1 Anonymous, terms of use under AGPL (Affero General Public License)
2 Informal cooperation (e.goccasional shared marketing or R&D activities)
3 Partnership program (e.gpartnership certification)
4 Other

Frequencies:
2.11 Relation to the |
   platform provider |      Freq    Percent        Cum.
---------------------+-----------------------------------
           Anonymous |         7       5.56        5.56
 Partnership porgram |       118      93.65       99.21
               Other |         1       0.79      100.00
---------------------+-----------------------------------
               Total |       126     100.00

Mean:
Mean estimation                    Number of obs    =     126

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_226 |   2.896825    .041906      2.813888    2.979763
--------------------------------------------------------------
```

Appendix H – 3. Platfrom attractiveness variables

```
Variable: v_237 "3.1 Willingness to initially invest"
Operationalization: How would you assess the resources (time, money, man power)...you
invested in your product for SCRM/SFDC until now?
Possible answers:  [1 No resources - 5 A large amount of resources]

Frequencies:
        3.1 |
Willingness |
         to |
  initially |
     invest |      Freq     Percent        Cum.
------------+-----------------------------------
          2 |          7        5.56        5.56
          3 |         39       30.95       36.51
          4 |         37       29.37       65.87
          5 |         43       34.13      100.00
------------+-----------------------------------
      Total |        126      100.00

Mean:
Mean estimation                     Number of obs    =     126

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_237 |   3.920635   .0832691      3.755835    4.085435
--------------------------------------------------------------



Variable: v_238 "3.2 Willingness to further invest"
Operationalization: How would you assess the resources (time, money, man power)...you
are willing to invest in the SCRM/SFDC ecosystem in the future?
Possible answers:  [1 No resources - 5 A large amount of resources]

Frequencies:
        3.2 |
Willingness |
 to further |
     invest |      Freq     Percent        Cum.
------------+-----------------------------------
          1 |          1        0.79        0.79
          2 |          8        6.35        7.14
          3 |         28       22.22       29.37
          4 |         50       39.68       69.05
          5 |         39       30.95      100.00
------------+-----------------------------------
      Total |        126      100.00

Mean:
Mean estimation                     Number of obs    =     126

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_238 |   3.928571   .0852102       3.75993    4.097213
--------------------------------------------------------------
```

```
Variable: v_236 "3.3 Expectations met"
Operationalization: Overall, how well has SCRM/SFDC met your expectations?
Possible answers:  [1 Very poorly - 5 Very well]

Frequencies:
        3.3 |
Expectation |
      s met |      Freq    Percent        Cum.
------------+-----------------------------------
          1 |         2       1.59        1.59
          2 |        10       7.94        9.52
          3 |         9       7.14       16.67
          4 |        58      46.03       62.70
          5 |        47      37.30      100.00
------------+-----------------------------------
      Total |       126     100.00

Mean:
Mean estimation                     Number of obs    =     126

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_236 |   4.095238   .0846495      3.927706     4.26277
--------------------------------------------------------------
```

```
Variable: v_239 "3.4 Pay-off"
Operationalization: Does it pay off to be in the SCRM/SFDC ecosystem?
Possible answers:  [1 Very poorly - 5 Very well]

Frequencies:
3.4 Pay-off |      Freq    Percent        Cum.
------------+-----------------------------------
          1 |         2       1.60        1.60
          2 |         3       2.40        4.00
          3 |        30      24.00       28.00
          4 |        52      41.60       69.60
          5 |        38      30.40      100.00
------------+-----------------------------------
      Total |       125     100.00


Mean:
Mean estimation                     Number of obs    =     125

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_239 |      3.968   .0794627      3.810721    4.125279
--------------------------------------------------------------
```

```
Variable: v_660 "Platform Attractiveness"
Caluclation: (v_236+ v_237+ v_238+ v_239)/4

Frequencies:
        3.3 |
   Platform |
    toolset |      Freq    Percent        Cum.
------------+-----------------------------------
         no |        37      29.37       29.37
        yes |        89      70.63      100.00
------------+-----------------------------------
      Total |       126     100.00

Mean:
Mean estimation                     Number of obs    =     126

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_260 |   .7063492   .0407352      .6257291    .7869693
--------------------------------------------------------------


Variable: v_670 "Willingness to invest"
Caluclation: v_670 = (v_237+ v_238)/2

Frequencies:
Willingness |
  to invest |      Freq    Percent        Cum.
------------+-----------------------------------
          2 |         4       3.17        3.17
        2.5 |         7       5.56        8.73
          3 |        18      14.29       23.02
        3.5 |        21      16.67       39.68
          4 |        30      23.81       63.49
        4.5 |        17      13.49       76.98
          5 |        29      23.02      100.00
------------+-----------------------------------
      Total |       126     100.00

Mean:
Mean estimation                     Number of obs    =     126

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_670 |   3.924603     .07455       3.77706    4.072147
--------------------------------------------------------------



Variable: v_680 "Return on investment"
Caluclation: v_680 = (v_236+ v_239)/2

Frequencies:
  Return on |
 investment |      Freq    Percent        Cum.
------------+-----------------------------------
        1.5 |         2       1.60        1.60
          2 |         1       0.80        2.40
        2.5 |        10       8.00       10.40
          3 |         7       5.60       16.00
        3.5 |        10       8.00       24.00
          4 |        42      33.60       57.60
        4.5 |        30      24.00       81.60
          5 |        23      18.40      100.00
------------+-----------------------------------
      Total |       125     100.00

Mean:
Mean estimation                     Number of obs    =     125

--------------------------------------------------------------
             |       Mean   StdErr    [95% ConfInterval]
-------------+------------------------------------------------
       v_680 |      4.032   .0713338       3.89081     4.17319
--------------------------------------------------------------
```

Appendix I – Factor analysis

```
Factor analysis/correlation                        Number of obs    =      125
    Method: principal-component factors            Retained factors =        2
    Rotation: (unrotated)                          Number of params =        6

    --------------------------------------------------------------------------
         Factor  |   Eigenvalue   Difference        Proportion   Cumulative
    -------------+------------------------------------------------------------
        Factor1  |      2.04197      0.96557            0.5105       0.5105
        Factor2  |      1.07639      0.53408            0.2691       0.7796
        Factor3  |      0.54231      0.20299            0.1356       0.9152
        Factor4  |      0.33933            .            0.0848       1.0000
    --------------------------------------------------------------------------
    LR test: independent vs. saturated:  chi2(6)  =  111.19 Prob>chi2 = 0.0000

Factor loadings (pattern matrix) and unique variances

    -------------------------------------------------
        Variable |  Factor1   Factor2 |   Uniqueness
    -------------+--------------------+--------------
           v_236 |   0.6745    0.5728 |      0.2169
           v_237 |   0.6594   -0.6461 |      0.1478
           v_238 |   0.8105   -0.3484 |      0.2217
           v_239 |   0.7037    0.4577 |      0.2953
    -------------------------------------------------

Factor analysis/correlation                        Number of obs    =      125
    Method: principal-component factors            Retained factors =        2
    Rotation: orthogonal varimax (Kaiser off)      Number of params =        6

    --------------------------------------------------------------------------
         Factor  |     Variance   Difference        Proportion   Cumulative
    -------------+------------------------------------------------------------
        Factor1  |      1.56809      0.01783            0.3920       0.3920
        Factor2  |      1.55027            .            0.3876       0.7796
    --------------------------------------------------------------------------
    LR test: independent vs. saturated:  chi2(6)  =  111.19 Prob>chi2 = 0.0000

Rotated factor loadings (pattern matrix) and unique variances

    -------------------------------------------------
        Variable |  Factor1   Factor2 |   Uniqueness
    -------------+--------------------+--------------
           v_236 |   0.0801    0.8813 |      0.2169
           v_237 |   0.9232    0.0009 |      0.1478
           v_238 |   0.8225    0.3192 |      0.2217
           v_239 |   0.1816    0.8196 |      0.2953
    -------------------------------------------------

Factor rotation matrix

    --------------------------------
                 | Factor1  Factor2
    -------------+------------------
         Factor1 |  0.7136   0.7005
         Factor2 | -0.7005   0.7136
    --------------------------------
```

Appendix J – Correlation analysis

	Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	Platform attractiveness	1																
2	Willingness to invest	**0.82***	1															4. Platform Attractiveness
3	Return on investment	**0.8***	0.32*	1														
4	1.1 Perceived fairness of platform provider	0.37*	0.15	0.46*	1													
5	1.2 Perceived risk to become dependent on platform provider	0.03	0.08	-0.04	-0.32*	1												1. Platform provider setting
6	1.3 Level of feasibility to generate customized solutions	0.24*	0.03	0.37*	0.22*	-0.05	1											
7	2.1 Complexity of downstream system	0.28*	0.31*	0.14	-0.04	0.18*	-0.05	1										
8	2.2 Need for downstream adaptation (degree of customization)	0.28*	0.18*	0.27*	0.1	-0.13	0.14	0.3*	1									2. Complementor setting
9	2.3 Importance of anonymous platform interface	-0.02	-0.02	-0.02	0.1	-0.2*	0.07	-0.03	0.03	1								
10	2.4 Importance of openness through access to platform know-how	-0.01	-0.02	0	0.01	-0.12	0.17	-0.08	0.19*	0.13	1							
11	2.5 Importance of low entry barrier to join platform ecosystem	-0.04	-0.07	0.01	-0.04	-0.07	0.01	-0.11	0.01	0.22*	0.14	1						
12	2.6 Ability of complementors to protect its IP	-0.06	-0.11	0.01	0.09	-0.11	0.04	0.06	-0.06	0.16	0.01	0.14	1					
13	2.7 Number of platform ecosystems connected to	-0.24*	-0.16	-0.22*	-0.13	-0.06	0.04	-0.06	-0.05	0.06	-0.03	0.05	0	1				
14	2.8 Importance of current end-users in ecosystem (market size)	0.1	0.09	0.06	-0.12	0.21*	-0.04	0.03	-0.2*	0	-0.09	-0.02	0.15	0.02	1			
15	2.9 Importance of potential end-users in ecosystem (market growth)	0.19*	0.12	0.19*	0.01	0.16	0.14	0.07	-0.02	-0.05	0.01	-0.04	0.18*	-0.11	0.72	1		
16	2.10 Firm size	-0.03	-0.04	-0.01	0.06	-0.03	0.06	0	0.05	0.17	-0.2*	-0.06	0.15	0.34*	0.08	0.07	1	
17	2.11 Relation to the platform provider	0.09	0.02	0.13	-0.02	0.18*	0.11	0.15	-0.13	-0.04	-0.17	0.25*	0.21	-0.04	0.18*	0.34*	0.01	1

Note: * $p<0.05$

Appendix K – Extended hypotheses tests

Regression: Salesforce.com with Return on investment as dependent variable

```
      Source |       SS       df       MS              Number of obs =      85
-------------+------------------------------           F( 10,    74) =    5.65
       Model |  29.1032302    10  2.91032302           Prob > F      =  0.0000
    Residual |  38.1202992    74  .515139179           R-squared     =  0.4329
-------------+------------------------------           Adj R-squared =  0.3563
       Total |  67.2235294    84  .800280112           Root MSE      =  .71773

------------------------------------------------------------------------------
       v_680 |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       v_233 |   .3559276    .072713     4.89   0.000     .2110439    .5008114
       v_234 |   .0824395   .0797728     1.03   0.305    -.0765112    .2413902
       v_231 |   .3811238    .138255     2.76   0.007     .1056447     .656603
       v_258 |   .0547695   .0732135     0.75   0.457    -.0911116    .2006505
       v_259 |   .0666137   .0631512     1.05   0.295    -.0592178    .1924453
       v_253 |  -.0289675    .071674    -0.40   0.687     -.171781    .1138461
       v_248 |  -.0587566   .0726901    -0.81   0.421    -.2035948    .0860816
       v_247 |   .0087557   .0752235     0.12   0.908    -.1411303    .1586418
       v_252 |   .0426946   .1073265     0.40   0.692    -.1711581    .2565473
       v_206 |  -.0912941   .0478789    -1.91   0.060     -.186695    .0041067
       v_226 |          0  (omitted)
       _cons |    .620858   .9471868     0.66   0.514    -1.266453    2.508169
------------------------------------------------------------------------------
```

Regression: SugarCRM with Return on investment as dependent variable

```
      Source |       SS       df       MS              Number of obs =      39
-------------+------------------------------           F( 11,    27) =    4.13
       Model |  7.06811742    11  .642556129           Prob > F      =  0.0013
    Residual |  4.20111335    27  .155596791           R-squared     =  0.6272
-------------+------------------------------           Adj R-squared =  0.4753
       Total |  11.2692308    38  .296558704           Root MSE      =  .39446

------------------------------------------------------------------------------
       v_680 |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       v_233 |   .0522773   .0661207     0.79   0.436    -.0833912    .1879458
       v_234 |   .0726363   .0714259     1.02   0.318    -.0739175    .2191901
       v_231 |   .3901003   .1223804     3.19   0.004     .1389965    .6412041
       v_258 |  -.0557454   .0765295    -0.73   0.473    -.2127709    .1012801
       v_259 |   .2973917   .1078371     2.76   0.010     .0761282    .5186551
       v_253 |   .1324994   .0760399     1.74   0.093    -.0235216    .2885204
       v_248 |   .0366891   .1109586     0.33   0.743    -.1909792    .2643574
       v_247 |   .0129419   .0713981     0.18   0.858    -.1335549    .1594387
       v_252 |  -.1602785   .0808785    -1.98   0.058    -.3262274    .0056704
       v_206 |   .0036826   .0605712     0.06   0.952    -.1205992    .1279645
       v_226 |   .2549362   .1098764     2.32   0.028     .0294884     .480384
       _cons |   .1915765   .7347588     0.26   0.796    -1.316024    1.699177
------------------------------------------------------------------------------
```

Test H1b: The coefficient for "Need for downstream adaptation" is significant different in the regressions of the subsamples?

```
 ( 1)  [regSF_mean]v_259 - [regSugar_mean]v_259 = 0

           chi2(  1) =    6.30
         Prob > chi2 =    0.0121
```

Test H2: The coefficient for "Importance of anonymous platform interface" is significant different in the regressions of the subsamples?

```
. test [regSF_mean]v_253=[regSugar_mean]v_253

 ( 1)  [regSF_mean]v_253 - [regSugar_mean]v_253 = 0

           chi2(  1) =    4.96
         Prob > chi2 =    0.0259
```

Bibliography

Anvaari, M. and Jansen, S. (2010), “Evaluating architectural openness in mobile software platforms”, in *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ACM, Copenhagen, Denmark, pp. 85–92.

Baldwin, C.Y. (2007), “Where do transactions come from? Modularity, transactions, and the boundaries of firms”, *Industrial and Corporate Change*, Vol. 17 No. 1, pp. 155–195.

Baldwin, C.Y. and Clark, K.B. (1997), “Managing in an Age of Modularity”, *Harvard Business Review*, Vol. 75 No. 5, pp. 84–93.

Baldwin, C.Y. and Clark, K.B. (2000), *Design rules*, MIT Press, Cambridge, Mass.

Baldwin, C.Y. and Clark, K.B. (2006), “The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model?”, *Management Science*, Vol. 52 No. 7, pp. 1116–1127.

Baldwin, C.Y. and Henkel, J. (2011), “The Impact of Modularity on Intellectual Property and Value Appropriation”, *Harvard Business School, Working Paper 12-040.*

Baldwin, C.Y. and Woodard, J.C. (2009), “The Architecture of Platforms: A Unified View”, in Gawer, A. (Ed.), *Platforms, Markets and Innovation*, Cheltenham, U.K. and Northampton, Mass.: Elgar.

Bansal, P. and Corley, K. (2011), “The coming Age for Qualitative Research: Embracing the Diversity of Qualitative Methods”, *Academy of Management Journal*, Vol. 54 No. 2, pp. 233–237.

Bland, J.M. and Altman, D.G. (1997), “Statistics notes: Cronbach's alpha”, *BMJ*, Vol. 314 No. 7080, p. 572.

Bonaccorsi, A., Giannangeli, S. and Rossi, C. (2006), “Entry Strategies Under Competing Standards: Hybrid Business Models in the Open Source Software Industry”, *Management Science*, Vol. 52 No. 7, pp. 1085–1098.

Boudreau, K.J. (2010), “Open Platform Strategies and Innovation: Granting Access vs. Devolving Control”, *Management Science*, Vol. 56 No. 10, pp. 1849–1872.

Boudreau, K.J. and Lakhani, K.R. (2009), “How to Manage Outside Innovation”, *MIT Sloan Management Review*, Vol. 50 No. 4, pp. 69–76.

Brandenburger, A.M. and Nalebuff, B.J. (1996), *Co-opetition: A Revolution Mindset That Combines Competition and Cooperation*, Doubleday Dell Publishing, New York.

Burkard, C., Draisbach, T. and Buxmann, P. (2011), “Software Ecosystems: Vendor-Sided Characteristics of Online Marketplaces”, in Heiss, H.-U. (Ed.), *Informatik 2011: Informatik schafft Communities ; 41. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 4.10. bis 7.10.2011, TU Berlin*, Ges. für Informatik, Bonn.

Carver, B.W. (2005), “Share and Share Alike: Understanding and Enforcing Open Source and Free Software Licenses”, *Berkeley Technology Law Journal*, Vol. 20 No. 1, pp. 443–481.

Chung, L. and do Prado Leite, J. (2009), “On Non-Functional Requirements in Software Engineering”, in Borgida, A., Chaudhri, V., Giorgini, P. and Yu, E. (Eds.), *Conceptual Modeling: Foundations and Applications*, *Lecture Notes in Computer Science*, Vol. 5600, Springer Berlin / Heidelberg, pp. 363–379.

Creswell, J.W. (2003), *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches,* 2nd edition, Sage Publications, Thousand Oaks.

Cronbach, L.J. (1951), “Coefficient alpha and the internal structure of tests”, *Psychometrika*, Vol. 16 No. 3, pp. 297–334.

Cusumano, M.A. (2010), *Staying power: Six enduring principles for managing strategy and innovation in an uncertain world (lessons from Microsoft, Apple, Intel, Google, Toyota and more)*, Oxford University Press, Oxford, New York.

Dubé, L. and Paré, G. (2003), “Rigor in Information Systems Positivist Case Research: Current Practices, Trends, and Recommondations”, *MIS Quarterly*, Vol. 27 No. 4, pp. 597–635.

Echambadi, R. and Hess, J.D. (2007), “Mean-Centering Does Not Alleviate Collinearity Problems in Moderated Multiple Regression Models”, *Marketing Science*, Vol. 26 No. 3, pp. 438–445.

Echambadi, R., Campbell, B. and Agarwal, R. (2006), “Encouraging Best Practice in Quantitative Management Research: An Incomplete List of Opportunities”, *Journal of Management Studies*, Vol. 43 No. 8, pp. 1801–1820.

Edmondson, A.C. and McManus, S.E. (2007), “Methodological Fit in Management Field Research”, *Academy of Management Review*, Vol. 32 No. 4, pp. 1155–1179.

Eisenhardt, K.M. (1989), “Building Theories from Case Study Research”, *Academy of Management Review*, Vol. 14 No. 4, pp. 532–550.

Eisenhardt, K.M. and Grabner, M.E. (2007), “Theory Building from Cases: Opportunities and Challenges”, *Academy of Management Journal*, Vol. 50 No. 1, pp. 25–32.

Eisenmann, T., Parker, G. and van Alstyne, M.W. (2006), “Strategies for Two-Sided Markets”, *Harvard Business Review*, Vol. 84 No. 10, pp. 92–101.

Eisenmann, T.R., Parker, G. and van Alstyne, M. (2009), “Opening Platforms: How, When and Why?”, in Gawer, A. (Ed.), *Platforms, Markets and Innovation*, Cheltenham, U.K. and Northampton, Mass.: Elgar, pp. 131–162.

Eppinger, S.D. (1991), “Model-Based Approaches to Managing Concurrent Engineering”, *Journal of Engineering Design*, Vol. 2 No. 4, pp. 283–290.

Eppinger, S.D., Whitney, D.E., Smith, R.P. and Gebala, D.A. (1994), “A Model-Based Method for Organizing Tasks in Product Development”, *Research in Engineering Design*, Vol. 6 No. 1, pp. 1–13.

Ethiraj, S.K., Levinthal, D. and Roy, R.R. (2008), "The Dual Role of Modularity: Innovation and Imitation", *Management Science*, Vol. 54 No. 5, pp. 939–955.

Farrell, J. and Klemperer, P. (2007), "Coordination and Lock-In: Competition with Switching Costs and Network Effects", in Armstrong, M. and Porter, R. (Eds.), *Handbook of Industrial Organization: Volume 3*, Elsevier, Amsterdam, pp. 1967–2072.

Favaro, J. and Pfleeger, S.L. (2011), "Guest Editors' Introduction: Software as a Business", *IEEE Software*, Vol. 28 No. 4, pp. 22–25.

Flick, U. (2011), *Introducing Research Methodology: A Beginner's Guide to Doing a Research Project*, Sage Publications, London.

Fricker, S.A. (2012), "Software Product Management", in Maedche, A., Botzenhardt, A. and Neer, L. (Eds.), *Management for Professionals*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 53–81.

Gawer, A. (2009), "Platform Dynamics and Strategies: From Products to Services", in Gawer, A. (Ed.), *Platforms, Markets and Innovation*, Cheltenham, U.K. and Northampton, Mass.: Elgar, pp. 45–76.

Gawer, A. and Cusumano, M.A. (2002), *Platform leadership: How Intel, Microsoft, and Cisco drive industry innovation*, Harvard Business School Press, Boston, Mass.

Gawer, A. and Henderson, R. (2007), "Platform Owner Entry and Innovation in Complementary Markets: Evidence from Intel", *Journal of Economics and Management Strategy*, Vol. 16 No. 1, pp. 1–34.

Gephart, R.P., JR. (2004), "Qualitative Research and the Academy of Management Journal", *Academy of Management Journal,* 2004, pp. 454–462.

Gibbert, M., Ruigrok, W. and Wicki, B. (2008), "What passes as a rigorous case study?", *Strategic Management Journal*, Vol. 29 No. 13, pp. 1465–1474.

Glaser, B. and Strauss, A. (2008), *The discovery of grounded theory: Strategies for qualitative research*, Aldine Transaction, New Brunswick and London.

Greenwood, R. and Suddaby, R. (2006), "Institutional Entrepreneurship in Mature Fields: The Big Five Accounting Firms.", *Academy of Management Journal*, Vol. 49 No. 1, pp. 27–48.

Hecker, F. (1999), "Setting up shop: The business of open-source software. Software, IEEE", *Software, IEEE (Software, IEEE)*, Vol. 16 No. 1, pp. 45–51.

Henkel, J. (2011), *IP-Modularität - In offenen Innovationsprozessen profitieren durch IP-orientierte Modularisierung, Münchener Innovations-Konferenz*, Munich.

Henkel, J. and Baldwin, C.Y. (2010), "Modularity for Value Appropriation - How to Draw the Boundaries of Intellectual Property", *Harvard Business School Finance Working Paper No. 11-054*.

Huang, P., Ceccagnoli, M., Forman, C. and Wu, D. (2009), "Participation in a Platform Ecosystem: Appropriability, Competition and Access to the Installed Base", Working Paper No. 09-14.

Jansen, S. and Cusumano, M.A. (2012), "Defining Software Ecosystems: A Survey of Software Platforms and Business Network Governance", in *Proceedings of the Forth International Workshop on Software Ecosystems, Cambridge, MA, USA, June 18th, 2012.*, Vol. 879.

Jansen, S., Brinkkemper, S., Souer, J. and Luinenburg, L. (2012), "Shades of gray: Opening up a software producing organization with the open software enterprise model. Software Ecosystems", *Journal of Systems and Software*, Vol. 85 No. 7, pp. 1495–1510.

Jeppesen, L.B. and Molin, M.J. (2003), "Consumers as Co-developers: Learning and Innovation Outside the Firm", *Technology Analysis & Strategic Management*, Vol. 15 No. 3, pp. 363–383.

Kaiser, H.F. (1960), "The Application of Electronic Computers to Factor Analysis", *Educational and Psychological Measurement*, Vol. 20 No. 1, pp. 141–151.

Katz, M.L. and Shapiro, K. (1985), "Network Externalities, Competition, and Compatibility", *American Economic Review*, Vol. 75 No. 3, pp. 424–440.

Kittlaus, H.-B. and Clough, P.N. (2009), *Software product management and pricing: Key success factors for software organizations*, Springer, Berlin.

Kude, T., Dibbern, J. and Heinzl, A. (2012), "Why Do Complementors Participate? An Analysis of Partnership Networks in the Enterprise Software Industry", *IEEE Transactions on Engineering Management*, Vol. 59 No. 2, pp. 250–265.

LaMantia, M.J., Yuanfang Cai, MacCormack, A.D. and Rusnak, J. (2008), "Analyzing the Evolution of Large-Scale Software Systems Using Design Structure Matrices and Design Rule Theory: Two Exploratory Cases", *Software Architecture, 2008. WICSA 2008. Seventh Working IEEE/IFIP Conference on*, pp. 83–92.

Lance, C.E. and Vandenberg, R.J. (2009), *Statistical and methodological myths and urban legends: Doctrine, verity and fable in the organizational and social sciences*, Routledge, New York.

Langlois, R.N. (2002), "Modularity in Technology and Organization", *Journal of Economic Behavior & Organization*, Vol. 49 No. 1, pp. 19–37.

Lee, M.K.O. and Cheung, C.M.K. (2004), "Internet Retailing Adoption by Small-to-Medium Sized Enterprises (SMEs): A Multiple-Case Study", *Information Systems Frontiers*, Vol. 6 No. 4, pp. 385–397.

Lerner, J. and Tirole, J. (2002), "Some Simple Economics of Open Source", *The Journal of Industrial Economics*, Vol. 50 No. 2, pp. 197–234.

Lindman, J., Rossi, M. and Puustell, A. (2011), "Matching Open Source Software Licenses with Corresponding Business Models. Software, IEEE", *Software, IEEE (Software, IEEE)*, Vol. 28 No. 4, pp. 31–35.

MacCormack, A., Rusnak, J. and Baldwin, C. (2006), “Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code”, *Management Science No.* 52, pp. 1015–1030.

Mahoney, J. (2006), “A Tale of Two Cultures: Contrasting Quantitative and Qualitative Research”, *Political Analysis*, Vol. 14 No. 3, pp. 227–249.

Mann, H.B. and Whitney, D.R. (1947), “On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other”, *The Annals of Mathematical Statistics*, Vol. 18 No. 1, pp. 50–60.

Miles, M.B. and Huberman, A.M. (1994), *Qualitative data analysis: An expanded sourcebook,* 2. ed, Sage, Thousand Oaks, Calif. [u.a.].

Osterwalder, A. (2004), “The Business Model Ontology. A Proposition in a Design Science Approach”, Dissertation, École des Hautes Études Commerciales, Université de Lausanne, Lausanne, 2004.

Parker, G. and van Alstyne, M.W. (2009), “Six Challenges in Platform Licensing and Open Innovation”, *Communications & Strategies*, Vol. 1 No. 74, pp. 17–36.

Parnas, D.L. (1972), “On the Criteria to be used in Decomposing Systems into Modules”, *Communications of the ACM*, Vol. 15 No. 12, pp. 1053–1058.

Parnas, D.L., Clements, P.C. and Weiss, D.M. (1985), “The Modular Structure of Complex Systems”, *IEEE Transactions on Software Engineering*, SE-11 No. 3, pp. 259–266.

Perrons, R.K. (2009), “The Open Kimono: How Intel Balances Trust and Power to Maintain Platform Leadership”, *Research Policy*, Vol. 38 No. 8, pp. 1300–1312.

Pettigrew, A.M. (2010), “Longitudinal Field Research on Change: Theory and Practice”, in Olk, P. (Ed.), *Strategy Process*, Elgar Reference Collection. Strategic Management series. Cheltenham, U.K. and Northampton, Mass.: Elgar, pp. 473–498.

Popp, K.M. (2011), “Software Industry Business Models. Software, IEEE”, *Software, IEEE (Software, IEEE)*, Vol. 28 No. 4, pp. 26–30.

Pratt, M.G. (2009), “For the Lack of a Boilerplate: Tips on Writing up (and Reviewing) Qualitative Research”, *Academy of Management Journal*, Vol. 52 No. 5, pp. 856–862.

Raymond, E.S. (2001), *The cathedral and the bazaar: Musings on Linux and Open Source by an accidental revolutionary,* Rev. ed., O'Reilly, Cambridge, Mass.

Riehle, D. (2012), “The single-vendor commercial open course business model”, *Information Systems and e-Business Management*, Vol. 10 No. 1, pp. 5–17.

Rochet, J.-C. and Tirole, J. (2003), “Platform Competition in Two-Sided Markets”, *Journal of the European Economic Association*, Vol. 1 No. 4, pp. 990–1029.

Ruscio, J. and Roche, B. (2012), “Determining the number of factors to retain in an exploratory factor analysis using comparison data of known factorial structure”, *Psychological Assessment*, Vol. 24 No. 2, pp. 282–292.

Sako, M. and Helper, S. (1998), “Determinants of Trust in Supplier Relations. Evidence from the Automotive Industry in Japan and the United States”, *Journal of Economic Behavior & Organization*, Vol. 34 No. 3, pp. 387–417.

Sanchez, R. and Mahoney, J. (1996), “Modularity, Flexibility, and Knowledge Management in Product and Organization Design”, *Strategic Management Journal*, 17 Winter special Issue, pp. 63–76.

Santos J. Reynaldo A. (1999), “Cronbach's Alpha: A Tool for Assessing the Reliability of Scales”, available at: http://www.joe.org/joe/1999april/tt3.php/tt2.php.

Schilling, M.A. (2000), “Toward a General Modular Systems Theory and Its Application to Interfirm Product Modularity”, *Academy of Management Review*, Vol. 25 No. 2, pp. 312–334.

Schreiner, K. (2012), “IP Modularity in Software Platform Ecosystems”, Master Thesis, Technische Universität München, Munich, 2012.

Sosa, M.E., Eppinger, S.D. and Rowles, C.M. (2004), “The Misalignment of Product Architecture and Organizational Structure in Complex Product Development”, *Management Science*, Vol. 50 No. 12, pp. 1674–1689.

Steensma, H.K. and Corley, K.G. (2000), “On the Performance of Technology Sourcing Partnerships. The Interaction between Partner Interdependence and Technology Attributes”, *Academy of Management Journal*, Vol. 43 No. 6, pp. 1045–1067.

Steward, D. (1981), “The Design Structure System. A Method for Managing the Design of Complex Systems”, *IEEE Transations on Engineering Management*, Vol. 28 No. 3, pp. 71–84.

Suddaby, R. (2006), “From the Editors: What Grounded Theory is not”, *Academy of Management Journal,* 2006, pp. 633–642.

Teece, D.J. (1986), “Profiting from Technological Innovation: Implications for Integration, Collaboration, Licensing and Public Policy”, *Ricerche Economiche*, Vol. 40 No. 4, pp. 607–643.

Ulrich, K.T. (1995), “The Role of Product Architecture in the Manufacturing Firm”, *Research Policy*, Vol. 24, pp. 419–440.

Waltl, J., Henkel, J. and Baldwin, C.Y. (2012), “IP Modularity in Software Ecosystems: How SugarCRM’s IP and Business Model Shape Its Product Architecture. Software Business”, in Cusumano, M.A., Iyer, B., Venkatraman, N., Aalst, W., Mylopoulos, J., Rosemann, M., Shaw, M.J. and Szyperski, C. (Eds.), *Lecture Notes in Business Information Processing*, Vol. 114, Springer Berlin Heidelberg, pp. 94–106.

Weill, P., Malone, T.W., D'Urso, V.T., Herman, G. and Woerner, S. (2005), “Do Some Business Models Perform Better than Others? A Study of the 1000 Largest US Firms”.

West, J. (2003), “How Open Is Open Enough? Melding Proprietary and Open Source Platform Strategies”, *Research Policy*, Vol. 32 No. 7, pp. 1259–1285.

Wiegers, K.E. (2003), *Software requirements, second edition,* 2nd ed., Microsoft Press, Redmond, Wash.

Wilcoxon, F. (1945), “Individual Comparisons by Ranking Methods”, *Biometrics Bulletin*, Vol. 1 No. 6, pp. 80–83.

William Band (2010), *The Forrester Wave™: CRM Suites For Midsized Organizations, Q2 2010*, Cambridge, Mass.

Williamson, O.E. (1979), “Transaction-Cost Economics. The Governance of Contractual Relations”, *Journal of Law and Economics*, Vol. 22 No. 2, pp. 233–261.

Yin, R.K. (2009), *Case study research: Design and methods,* 4th ed., Sage Publications, Los Angeles, Calif.

Yourdon, E. and Constantine, L.L. (1979), *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, Prentice Hall, Englewood Cliffs.

Zaheer, A., McEvily, B. and Perrone, V. (1998), “Does Trust Matter? Exploring the effects of Interorganizational and Interpersonal Trust on Performance”, *Organization Science*, Vol. 9 No. 2, pp. 141–159.