Architecture Challenges for Internal Software Ecosystems: A Large-Scale Industry Case Study

Klaus-Benedikt Schultis
Siemens Corporate Technology
Erlangen, Germany
klaus-benedikt.schultis.ext@siemens.com

Christoph Elsner
Siemens Corporate Technology
Erlangen, Germany
christoph.elsner@siemens.com

Daniel Lohmann
Friedrich-Alexander University
Erlangen-Nuremberg, Germany
lohmann@cs.fau.de

ABSTRACT

The idea of software ecosystems encourages organizations to open software projects for external businesses, governing the cross-organizational development by architectural and other measures. Even within a single organization, this paradigm can be of high value for large-scale decentralized software projects that involve various internal, yet self-contained organizational units. However, this intra-organizational decentralization causes architecture challenges that must be understood to reason about suitable architectural measures.

We present an in-depth case study on collaboration and architecture challenges in two of these large-scale software projects at Siemens. We performed a total of 46 hours of semi-structured interviews with 17 leading software architects from all involved organizational units. Our major findings are: (1) three collaboration models on a continuum that ranges from high to low coupling, (2) a classification of architecture challenges, together with (3) a qualitative and quantitative exposure of the identified recurring issues along each collaboration model. Our study results provide valuable insights for both industry and academia: Practitioners that find themselves in one of the collaboration models can use empirical evidence on challenges to make informed decisions about counteractive measures. Researchers can focus their attention on challenges faced by practitioners to make software engineering more effective.

Categories and Subject Descriptors

D.2 [**Software Engineering**]

General Terms

Design, Management, Human Factors, Documentation

Keywords

Software ecosystem, software product line, software architecture, decentralized software engineering, collaboration, case study

1. INTRODUCTION

> *"The most serious misconception that can be done by a platform provider is to believe that the platform is used as intended."*
>
> —Platform chief architect at Siemens

Software product lines proved successful to enable reuse of software within an organization [4, 32]. Adopting software product lines involves the development of core assets for a defined scope and the creation of products by reusing them in a prescribed way [9]. Both activities are governed by a single, common management and target improvements for common business drivers like time to market, cost and quality [9, 32].

In contrast, in software ecosystems, organizations open their platform to external businesses in order to leverage a variety of externally developed functionality [4, 21]. Those businesses are usually autonomous, act within a decentralized environment and cannot be directed by a common management [4, 21]. Platform provider and external businesses have widely varying business drivers, like innovation, market expansion, profit or visibility [21, 24]. A prominent example is Apple's iOS where Apple acts as platform provider for thousands of autonomous application developers.

We have investigated two large-scale software projects (about 500 and 950 developers) within Siemens where the reality is somewhere between those two development paradigms. They involve a set of internal organizational units that are self-contained profit centers with own business objectives, organizational independent with own product management, and have to a wide extent autonomous processes and software-engineering life cycles. Thus, the view on the organizational structure moves from strict hierarchies towards more decentralized topologies. In the context of this paper, we define those systems as *internal software ecosystems (ISECOs).*

This decentralization with independent spheres of authority significantly impacts collaboration. Bosch et al. perceived software architecture as enabler for effective collaboration in large-scale software engineering [6]. However, in order to make informed decisions about suitable architectural measures, it is necessary to understand the modes of collaboration and resulting challenges in detail. This raises our research questions:

RQ1: How do organizational units collaborate in such a decentralized environment? Are there common collaboration models?

RQ2: Which architecture challenges become particularly crucial due to collaboration? Can we relate them to collaboration models?

Our contributions are (1) the identification of three different collaboration models on a continuum that ranges from high to low architecture and process coupling; (2) a classification of architecture challenges, including platform openness strategy, composition of decentralized developed software, preservation of the organizational-units independence, guarantee of software qualities across the ecosystem and compliance to architectural intentions and cross-cutting regulations; together with (3) a qualitative and quantitative exposure of respective recurring issues along each collaboration model.

FSE'14 , November 16–22, 2014, Hong Kong, China
Copyright 2014 ACM 978-1-4503-3056-5/14/11 ...$15.00.

Whereas software product lines [1, 9, 23, 32] and open software ecosystems [16, 21, 24] already received attention in literature, to this end, no empirical results exist on collaboration and emerging architecture challenges for intra-organizational, yet decentralized software engineering. To fill this significant gap, we conducted an in-depth qualitative case study involving two of the largest ISECOs at Siemens. To that end, we performed (A) a literature review as well as (B) 2 workshops and 7 unstructured pre-interviews with architects of the study systems to draw a guideline for (C) 17 semi-structured interviews (lasting 135-240 minutes each) involving the main architects of all organizational units and (D) inspected both developer networks and dozens of documents that are applied to coordinate development. We (E) draw our conclusions per interview based on the grounded theory method [19], (F) received feedback from the interviewees and (G) performed a cross-interview analysis.

The paper is laid out as follows: In Section 2, we depict both study systems and outline our research method. In Section 3, we introduce the collaboration models and characterize them along the dimensions of the business, architecture, process, and organization (BAPO) model [32]. In Section 4, we classify architecture challenges and discuss respective recurring issues along each collaboration model. In Section 5, we discuss threats to validity as well as implications for practitioners and researchers. In Section 6, we provide an overview on related work. In Section 7, we conclude the paper.

2. METHODOLOGY

This section depicts the research settings of both ISECOs we have investigated and outlines our research method.

2.1 Study Systems

Both study systems, ISECO-A and ISECO-B, comprise a keystone that provides a platform and multiple clients that build applications upon it. The keystone acts in a creative role but does not have power to direct. Compared to open ecosystems, the number of clients is low and involved parties can and commonly do communicate directly if required. Moreover, clients develop only one to a handful, but large applications. Below, the study systems are described in more detail. We use the notation (x/y) to roughly denote for each organizational unit the number of developers and development years, respectively.

ISECO-A initially involved KEYSTONE-A (100/11), CLIENT-A1 (110/10), CLIENT-A2 (100/10) and CLIENT-A3 (85/10). They defined the original scope, develop a set of software products for a shared market and collaborate based on a strategic reuse approach.

Later on, the clients TRACY (40/4), SUSAN (40/6) and STEPHEN (40/7) recognized the high reuse potential of the platform and joined the ecosystem. They are outside of the scope and independently develop software products for different markets. The units collaborate based on a platform reuse approach without strong strategic coupling. In the remainder, we depict the successful reuse story of SUSAN, the initial struggles when including STEPHEN and the transition of TRACY into the group of strategic clients above. Figure 1 depicts a simplified illustration of ISECO-A.

ISECO-B started with KEYSTONE-B (300/8), client DONNIE (150/7), CLIENT-B2 (70/7) and CLIENT-B3 (30/7). As for ISECO-A, they set the initial scope and develop a set of software products for a shared market. Compared to other clients, DONNIE is more dominate and takes more action to influence the ecosystem.

Later on, the platform scope was explicitly broadened to involve client PINO (150/2), CLIENT-B5 (200/1) and CLIENT-B6 (60/1). They apply a decoupled and composition-oriented approach to develop software products for different, partially overlapping markets. PINO was the first added client and acts as a pioneer.

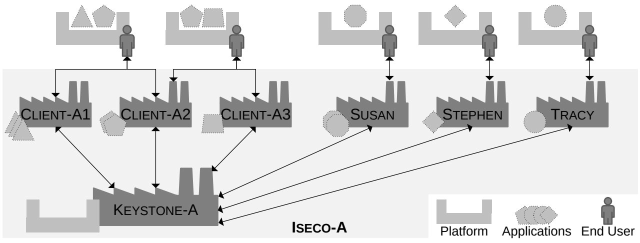

Figure 1: A simplified illustration of ISECO-A.

2.2 Research Method

Our qualitative case study targets the identification of collaboration models and emerging architecture challenges within ISECO-A and ISECO-B. The study was carried out over a period of one year and involved main architects of all organizational units. We structured our case study based on the guidelines by Runeson and Höst [28], developed a case study protocol and followed the method outlined in Figure 2.

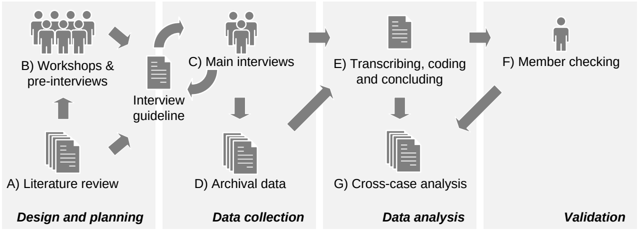

Figure 2: Research method.

A) Literature review. The first researcher analyzed existing literature for software ecosystems to collect already known issues that target architecture and collaboration, such as [5, 6, 17, 20, 21, 24, 16, 18]. He condensed them to discuss and refine insights with the second and third researcher in several iterations.

B) Workshops & pre-interviews. The first and second researcher performed two half-day workshops involving 7 respectively 8 architects from Siemens with an average professional experience of about 20 years to discuss how they do qualitatively perceive collaboration and emerging architecture challenges for ISECOs. Furthermore, they performed 7 unstructured interviews [28] (lasting about 2 hours each) with architects of ISECO-A and ISECO-B to get a basic understanding of the study systems and to commonly sketch collaboration among organizational units in a scenario-based fashion. Two of those architects were interviewed again during the main interviews.

As a result, we identified four categories of architecture challenges that become particular crucial: First, least-impairing *co-evolution* of platform and applications is required. Second, platform *interfaces* must be kept *stable* or must *evolve predictably*. Third, internal and external *qualities* must be guaranteed across the ecosystem. Fourth, *technical integration* must be enabled to compose the software developed in a decentralized manner. Based on those insights, we defined interview questions that target the identification of problems that have been arisen for each of those categories and measures that are applied to counter them. Furthermore, we defined one open question that targets the identification of further architecture challenges we were not aware of. We used those questions and the sketched scenarios to draw a guideline for our main interviews.

C) Main interviews. We contacted both platform chief architects, briefed them on the goal and process of our investigation and asked them for suitable interviewees for each organizational unit. We contacted the nominated architects to clarify if they feel suitable or if they would recommend another one. Thus, we got 17 architects for our main interviews: 1 for each client, 2 for the keystone of ISECO-A and 3 for the keystone of ISECO-B. Their professional experience ranges from 11-28 years with an average of about 21 years, and most of them are architects of the study systems since their initial phase. Each architect was interviewed separately. The interviews were semi-structured [28], lasted 135-240 minutes with an average of about 160 minutes and were digitally recorded with the interviewee's consent. The first researcher guided the interviews, the second researcher took notes and participated in discussions. The interview guideline was refined when new insights were gained.

Each interview was structured as follows: *(1) Briefing.* We started with a short briefing on the objective of the interview and case study, and introduced our notion of decentralized software engineering and the categories of challenges we had identified. *(2) Model collaboration.* Based on the sketched scenarios, we discussed the collaboration among the interviewee's organizational unit and other ones. *(3) Analyze architecture challenges.* Finally, we performed the main interview based on our interview guideline. Its semi-structured nature allowed us to reorder questions depended on the course of discussion while guaranteeing that all of them are handled. We did not only discuss challenges, but also measures that are applied to counter them. However, these are beyond the scope of this paper.

D) Archival data. Afterwards, most architects provided archival data that turned out to be relevant during the interviews. Thus, we inspected both developer networks, dozens of documents applied to coordinate development and results of an ATAM [8] review that was previously performed for ISECO-B. In doing so, we achieved triangulation across data sources which improves validity [28, 29].

E) Transcribing, coding and concluding. Data analysis was carried out in parallel with data collection, which is recommended practice to consider new insights [28]. The first researcher fully transcribed the audio recordings, coded the transcripts as described by Seaman [29] and documented first insights in the form of memos. In total, this results in 2732 chunks of coded text by using a hierarchy of 168 codes and sub-codes. Conclusions were drawn and summarized separately for each interview: This was an iterative process were the first researcher searched for patterns and trends in the coded data, grouped the data accordingly and analyzed related archival data and the notes that were taken during the interview. The results were carefully checked by the second researcher.

F) Member checking. Member checking is a recommended method to validate conclusions [29]. We sent our conclusion summaries (averagely about 3000 words) to the respective architects and sought for feedback. All architects agreed with our conclusions, only minor adaptations were suggested.

G) Cross-interview analysis. We performed a cross-interview analysis inspired by Eisenhardt [11]. Using the same technique as above, the first researcher created codes for the findings and coded the conclusion summaries while considering the interviewees' feedback. As proposed by Runeson et al. [28], he arranged the coded data in a table where rows represent findings, columns represent interviews and cells are marked if a finding is confirmed by the respective interview. This technique supported us to group study subjects by findings and to identify cross-cutting issues that are confirmed by multiple data sources, which increases validity [28, 29]. The results were carefully reviewed by the other two researchers.

3. RQ1: COLLABORATION MODELS

Software development within an ISECO context results in dependencies among both software assets and the responsible organizational units. This requires collaboration across organizational units in software engineering. We identified three different collaboration models ranging from high to low coupling. As mentioned before, the involved organizational units commonly are self-contained profit centers with own business objectives and autonomous software-engineering life cycles and processes. They are organizationally independent and have their own product management and R&D department. Although there is the option for an escalation and decisions through common top management, this is no generally viable way for deciding on day-to-day collaboration practices. Decision making on this operational level consists much more in negotiation and mutual agreement than in strict management decisions.

Below, we provide a brief introduction for each collaboration model and a characterization along the BAPO [32] dimensions.

Product-Line Engineering (PLE) for ISECOs. This collaboration model was the starting point for both ISECOs. It comprises the keystone and clients, in the following called *core clients*, who set the original scope. Those organizational units are the core participants and thus can significantly influence the direction of the ecosystem. The keystone is responsible for the platform and acts in a creative role, but does not have authority to give directions. The core clients build their applications upon the platform and, for this purpose, require additional features, changes and support by the keystone. Their influence differs regarding their strategical importance, active participation, human resources and geographical proximity. In fact, both ISECOs contain core clients that are more dominant compared to the others, for example DONNIE within ISECO-B. Finally, in some cases, core clients collaborate to generate synergies.

Platform Reuse for ISECOs. This collaboration model enlarges the original setting of ISECO-A. It additionally involves clients, in the following called *consumer clients*, that were not considered in the initial platform scope, but intend to leverage the high reuse potential. The key goal is to leverage reuse synergies within Siemens without requiring to extend the number of core clients. The resulting relation is similar to a supplier-consumer relationship, where the keystone offers its platform to consumer clients to build their applications upon. The keystone provides basic maintenance support for the current stage of development and previous releases of the platform. Consumer clients have less influence on the direction of the ISECO and less priority for feature and change requests. The fact that the platform may be still under development, and is not explicitly designed for their needs results in significant effort and initiative on consumer-client side. Creating awareness for this fact is essential for successful consumer-client applications. Whereas this worked well for SUSAN, the collaboration practices required significant readjustment in case of STEPHEN, as the initial expectation of a "ready-to-use" platform could not be fulfilled.

Decoupled PLE for ISECOs. Similar to product line engineering [3], organizations tend to stretch successful ISECOs significantly beyond their original scope to additionally involve related, not yet considered organizational units. This was the case for ISECO-B. Those organizational units, in the following called *extended core clients*, have related but also unique application requirements. They want to reuse selected sets of core assets with high reuse potential only, but not the platform as a whole. The keystone provides basic maintenance support for released sets of the core assets. Regarding their priority for feature and change requests, their position within the ISECO and their influence, the extended core clients are on a par with core clients and the keystone. Nevertheless, as for product line engineering [3], the broadened scope requires a more decoupled and composition-oriented software development approach.

In Table 1, we characterize the three models along the business, architecture, process and organization (BAPO) dimensions.

Table 1: A characterization of the three identified collaboration models along the BAPO dimensions.

1. PLE for ISECOs	2. Platform Reuse for ISECOs	3. Decoupled PLE for ISECOs
Business:	***Business:***	***Business:***
∗ Organizational units are self-contained profit centers that have *different* incentives and business objectives, but *similar* business cases.	∗ Consumer clients are self-contained profit centers that have *different* incentives, business objectives and *different* business cases.	∗ Extended core clients are self-contained profit centers that have *different* incentives, business objectives and *different* business cases.
∗ Organizational units *jointly* distribute a common set of products.	∗ Consumer clients distribute their applications *independently*, along with the reused platform.	∗ Extended core clients distribute their applications *independently*, along with the core assets.
∗ *Synergies* among applications provide added value for customers.	∗ Consumer clients develop *stand-alone* applications.	∗ Extended core clients develop *stand-alone* applications.
∗ Organizational units provide the *core funding* for the ISECO.	∗ Financial models for refunding exist, but there is more emphasis on leveraging *reuse synergies* than on exact cost accounting.	∗ Extended core clients *refund* the keystone for the deliveries, based on financing models.
Architecture:	***Architecture:***	***Architecture:***
∗ *Deep integration* of and among core-client applications.	∗ *Decoupling* of consumer-client applications.	∗ Reuse of mature and communal *core assets only*.
∗ *Strongly interconnected* architecture.	∗ *Partially decoupled* architecture.	∗ *Highly decoupled* architecture.
∗ Multiple core clients share *one common instance* of the platform.	∗ Each consumer client gets a *separate instance* of the delivered platform.	∗ Each extended core client gets a *separate instance* of the delivered core assets.
Process:	***Process:***	***Process:***
∗ Organizational units have dedicated, yet interconnected software-engineering life cycles and processes, and need to *synchronize* on crucial milestones.	∗ Consumer clients have widely autonomous software-engineering life cycles and processes, but there is an implicit *need to synchronize* to stay in pace of the platform.	∗ Extended core clients have autonomous software-engineering life cycles and processes. They *independently* select length, frequency and time of their iteration cycles.
∗ *Coupled* product release schedules.	∗ *Partially*[1] *coupled* product release schedules.	∗ *Decoupled* product release schedules.
∗ *Commitment* for collaborative development and to align processes and toolchains.	∗ Widely *independent* development and *limited commitment and possibility* to align processes and toolchains.	∗ *Commitment* for collaborative development, but *limited possibility* to align processes and toolchains.
Organization:	***Organization:***	***Organization:***
∗ Organizational units are associated through *similar* and also *overlapping* domains.	∗ Organizational units are partially associated through *related* domains.	∗ Organizational units are associated through *similar* and also *overlapping* domains.
∗ It is beneficial if organizational units belong to the *same* organizational sector to facilitate the collaboration required to apply PLE techniques.	∗ Organizational units generally can *span over several* organizational sectors.	∗ It is beneficial if organizational units belong to the *same* organizational sector to facilitate the collaboration required to apply PLE techniques.

Collaboration models on a continuum that ranges from high (left) to low (right) coupling regarding processes and architecture, but not the influence on other ecosystem partners.

4. RQ2: ARCHITECTURE CHALLENGES

In this section, we present the architecture challenges we have identified for ISECOs. We reclassify the categories of challenges identified in the workshops and pre-interviews (see Section 2.2) to simplify reporting. That is, we divide *co-evolution* into *independent platform development* and *independent application development* to present the challenge in separate from the two different perspectives. Furthermore, we aggregate *independent platform development* and *interface stability*, as these two challenges are strongly related. Finally, we identified two further cross-cutting challenges, that are *platform openness dilemma* and *compliant software development*. The classification is depicted in Figure 3. The challenges are related to each other and can often not be treated strictly separately.

Below, we discuss each challenge for the three collaboration models. Our collection comprises those issues that turned out as most crucial for the respective challenges and collaboration models within ISECO-A and ISECO-B. Nevertheless, we see the possibility that many of our findings may be generalized to further similar ecosystems. We discuss the generality of our study in Section 5.

In the remainder of this paper, we use the notation (x/y) to denote that a finding is confirmed by data of x out of y interviews that were relevant for the respective collaboration model. None of the findings is contradicted by any collected data.

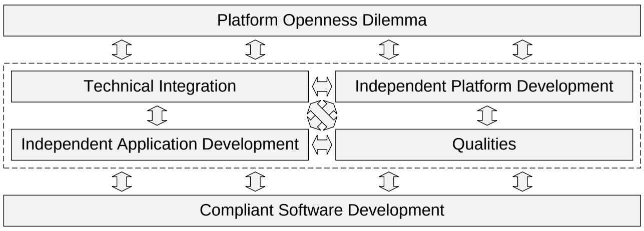
Platform Openness Dilemma
Technical Integration
Independent Platform Development
Independent Application Development
Qualities
Compliant Software Development

Figure 3: Architecture challenges for ISECOs.

4.1 Platform Openness Dilemma

The platform openness dilemma targets the selection of an appropriate point in time where the platform should be opened to the clients. There are contrary forces that justify either an early *platform opening approach* or a late one. On the one hand, requirements engineering was to a major part performed in the organizational units who have the appropriate knowledge in their domain. Therefore, an early involvement of the clients was essential in order to understand their development use cases. On the other hand, providing a platform to clients that is still growing, changing and maturing almost inevitably results in technical debt that needs to be costly refactored as a part of a *maturing process*. The focal question is: Which level of maturity is required before opening the platform and to which extent the clients' business pressure and requirements are missed if the platform is opened too late? For both ISECOs, the schedule pressure for product releases of multiple clients required the keystone to open the platform in early development phases. Below, the challenge is discussed in detail for each collaboration model.

4.1.1 PLE for ISECOs

Platform opening approach. For both ISECOs, the core-clients' requirements differed according to their business objectives. All of them were involved in architecture decision making, so that coming to mutual agreements could take significant amounts of time. Furthermore, the clients' development use cases were not completely fixed right from the beginning. The mission to meet the different and partially unknown needs led, for both ISECOs, to an optimistic opening approach that supported different concepts. A large number of interfaces[2] were publicly accessible and implemented extensibility

[1] For ISECO-A, SUSAN and TRACY release their products few months after the platform release. STEPHEN releases decoupled.

[2] The term *interface* is broadly defined for this paper, that is an interface is everything that is offered to clients, for example APIs, extension points, services or scripting languages.

in advance. This development context delivered more than one way to achieve the same goal. In conjunction with the difficulty to collect data on how clients are actually using the platform, and the not yet matured architecture guidance and governance (G&G)[3] measures, technical debt was accumulated, for instance, through unintentional dependencies, unexpected platform usages or inconsistencies.

Maturing process. The maturing process worked iteratively based on the clients' feedback. Yet, due to the feature-driven development, the remediation of technical debt requires strong arguments for the benefit of its removal. Especially changes without visible customer benefit have proven difficult to negotiate. The business impact needs to be argued to the product management of all affected organizational units, whose primary focus is on the realization of features their customers request. The explicit and systematic management of technical debt has been identified as one of the core issues to achieve transparency and take informed decisions.

Findings 1.1: The number of core clients, their schedule pressure, and their unknown needs require an optimistic opening approach (9/11). This provokes technical debt (9/11) that must be managed explicitly (8/11).

4.1.2 Platform Reuse for ISECOs

Platform opening approach. The primary focus of the platform did not include the business cases of consumer clients. Their development use cases are not well known and of lower priority compared to core clients. Combined with an optimistic opening approach, this increases the risk of unanticipated platform usage and, consequently, of technical debt. This was especially the case for STEPHEN, who joined the ecosystem in the initial phase where the absence of experiences hampered the identification of technical risks. In fact, this led to problems regarding performance and memory consumption, as concepts were used formally right, but significantly differed in the expected qualities, which was not anticipated by the keystone.

Maturing process. The platform matures based on the needs of core clients. Whereas the mutual understanding of the business cases has improved, the keystone is, due to the organizational distance, still not constantly aware about the actual platform usage by consumer clients. Thus, as part of the maturing process, consumer clients have to spend considerable efforts to adapt to changed or discontinued platform interfaces. This is a challenge all clients of this collaboration model were concerned with.

Findings 1.2: The low involvement in platform scope and decisions significantly increases technical risks during the optimistic opening approach for consumer clients (5/5).

4.1.3 Decoupled PLE for ISECOs

Platform opening approach. The keystone's core competence has not been in the domain of the extended core clients. Hence, an early alignment and involvement is even more important to understand their needs. On the other hand, a more restrictive opening approach is required since reactive removal of technical debt will be extremely costly within this decoupled context. However, this is a challenging and time-consuming task. For ISECO-B, the core assets are interconnected within the architecture and need to be carved out first. Additionally, the core assets have dependencies to core-client applications that should not be broken. Finally, for maintenance reasons the aspired result is only one single code base for the keystone. In fact, the carve-out activities are still in progress, but extended core clients have already started to develop their applications upon.

Maturing process. The decoupled software development requires using already matured core assets only, which possess a high degree of stability. For ISECO-B, the main challenge is to complete the carve-out with least possible side-effects. Experience has shown that forbidden, but programmatically reachable parts of the platform are used under schedule pressure when needed, carrying the risk of technical debt. Consequently, there is the need to explicitly define those parts that are allowed to use and to either strictly govern extended core clients for adherence or perform explicit violation management right from the beginning.

Findings 1.3: The decoupling requires a restrictive opening approach (5/5), although extended core clients need to be involved in early stages (5/5). Architecture G&G must be established to avoid or, at least, to explicitly manage technical debt (5/5).

4.2 Technical Integration

This challenge is concerned with processes and architecture measures that need to be applied to enable clients to integrate their applications into the platform. Below, we discuss the specific challenges that arise for each collaboration model in more detail.

4.2.1 PLE for ISECOs

For both ISECOs, the keystone and core clients develop a common set of products that are jointly distributed. Thereby, synergies among client applications shall generate added value for customers. Furthermore, clients often need to extend the platform with specific functionality that is not offered as commonality[4]. Thus, the applications need to be integrated deeply into all layers of the architecture. Additionally, in some cases, there is the need to enable integration among client applications, which increases complexity once more. The deep integration implies dependencies that restrict independent development, discussed in detail in a challenge below. Finally, the coupled product release schedules as well as the strongly interconnected architecture require short integration cycles and a comprehensive understanding of the others' contributions.

Findings 2.1: Core clients require a deep integration of their applications (11/11), short integration cycles (10/11), and an understanding of the others' contributions (10/11).

4.2.2 Platform Reuse for ISECOs

SUSAN, STEPHEN and TRACY have the solely responsibility to integrate their applications. In principle, they can select their integration frequency independently. In practice, the need for added features, improved qualities and feedback call for achieving short integration cycles. To shorten the comparatively long loops was a critical factor for TRACY's decision to become a core client. The fact that the platform is not explicitly aligned with the consumer-clients' needs inevitably results in more integration efforts. Awareness about this and appropriate measures are crucial for the client's success. For instance, SUSAN proactively investigates about a quarter of her work for contributing to platform development. Finally, some consumer-client applications not only build upon the platform but also on top of several core-client applications that are reused. The integration problem is rather obvious: Those applications were initially not designed as platform and thus have no stability guarantees and lack supporting architecture G&G.

Findings 2.2: Consumer clients have to cope with increased integration effort (5/5), and intend to shorten integration cycles despite sacrificing autonomy (4/5).

[3] According to Harrison et al. [15], *architecture guidance and governance* is the practice and orientation by which software architectures are managed and controlled at an organizational-wide level.

[4] For ISECO-A and ISECO-B, *commonalities* are features that are required by more than one core client or extended core client.

4.2.3 Decoupled PLE for ISECOs

ISECO-B comprises several extended core clients with autonomous software-engineering life cycles and decoupled release schedules. They only reuse selected core assets of the platform. There is the need for integration processes and architecture measures that enable them to independently integrate their applications into provided sets of the core assets. In addition, they also need to integrate existing components they want to reuse. For ISECO-B, there is even the challenge to cope with interoperability, as some of their existing components were designed for different runtime infrastructures.

Findings 2.3: Extended core clients require processes and measures that allow to integrate the core assets with their applications independently and with little effort (5/5).

4.3 Independent Platform Development

Independent platform development targets the capability of the platform to evolve independently. The platform regularly needs to incorporate new features to fulfill the clients' requirements. Likewise, out-dated features need to be removed to keep the complexity of the platform manageable. Finally, accumulated technical debt needs to be refactored. The architecture challenge is twofold. First, the keystone must consider *dependencies* to client applications, which restrict the keystone's possibilities of action. Second, *breaking changes* must be managed, communicated and need to be carried out incrementally over a long period to give clients enough time to align their applications. Below, we discuss the recurring issues of this challenge for each collaboration model.

4.3.1 PLE for ISECOs

Dependency management. As mentioned above, the optimistic opening approach exposed a vast number of interfaces accessible at the beginning. Along with the initially lacking G&G, the architecture eroded as clients used non-matured or for their use case improper interfaces, in parts explicitly allowed by the keystone due to the lack of alternatives. This led to undesired dependencies that reduce the evolution capability of the platform.

As a part of the maturing process, it has been becoming more and more clear which interfaces should be exposed to clients. Today, there is the need to close the platform reactively, based on the clients' feedback. However, the reactive closure necessitates refactoring activities that, generally, do require clients to change working applications while having the pressure to release new features. Furthermore, clients often require those interfaces and, if they are commonalities, the keystone first needs to offer alternatives. There is the need to manage those dependencies explicitly. That is, existing dependencies need to be removed incrementally when alternatives are available, and new undesired dependencies need to be tracked and are only allowed temporarily upon explicit consultation. Thereby, consensus among all affected parties is required as the keystone does not have authority to give directions. Furthermore, the keystone relies on the clients' cooperation as adherence to architecture guidelines cannot be strictly enforced.

Breaking changes. Interfaces cannot be kept strictly static and need to evolve over time. Thus, the keystone must manage breaking changes for interfaces that are explicitly intended for client use. Breaking changes not only concern the syntax but often also non-functional aspects, like behavior and assumptions, which turned out to be the key challenge. As consensus is desired, core clients can veto the adaptation if they can provide conclusive arguments. A transparent decision process weighting up the different arguments is required at this point. There is also the need for a change process that allows to negotiate and communicate changes significantly ahead of time of adjustments to give clients the opportunity to assess the impact, to vote and to align their applications if mutually agreed.

Findings 3.1: The optimistic opening approach lead to undesired dependencies of applications (8/11) that block the platform's evolution capability (8/11) and must be managed explicitly (7/11). Breaking changes must be mutually agreed (11/11), communicated in due time (9/11) and carried out incrementally (10/11).

4.3.2 Platform Reuse for ISECOs

Dependency management. The platform evolves when required by the business of the core clients, even in the case that the adaptation break applications of consumer clients. Nonetheless, the keystone tries to consider their needs and platform adaptations are not appropriate when they endanger the business success of the consumer clients. The actual difficulty is rooted in the different business cases, the low influence in platform scope and decisions and, for ISECO-A, the large organizational distance. The keystone does generally not know about the exact interface dependencies of consumer clients to the platform. In order to consider them there is the need to provide these information to the keystone along with the estimated impact of a potential adaptation.

Breaking changes. As for core clients, semantic changes turned out to be the key problem. As mentioned before, consumer clients are of lower priority, so their possibility to veto changes is much more restrained. As a result, consumer clients restrict the independence of the keystone only to a limited extent.

Findings 3.2: The platform commonly evolves when required by its core business, even if adaptations break applications of consumer clients (5/5). The keystone tries to consider them (5/5), but does commonly not know which interfaces are used and how they are used (5/5).

4.3.3 Decoupled PLE for ISECOs

Dependency management. The keystone needs to deliver selected sets of core assets to extended core clients. In order to fulfill their varying business cases, extended core clients need different combinations of the core assets, partially in different versions. As a consequence of the decoupled product release schedules, it must be possible to develop, build, release and deliver those required sets independently from each other. Hence, strict dependency management between the different core assets is required.

As for core clients, also the dependencies among client applications and core assets need to be managed to preserve the keystone's independence. The decoupled development context and the potential large number of extended core clients make this particularly challenging, as the keystone will not be able to track dependencies. Consequently, significantly more restrictive approaches are required. There is the need to explicitly define those dependencies that are allowed and to govern clients for adherence as the keystone will need to act resting upon those assumptions. However, the not yet finished carve-out activities already required four violations at PINO's pioneer application that are noticed with concern.

Breaking changes. As for core clients, the keystone must manage breaking changes. In particular, there is the need to differentiate between changes that affect clients who built their applications upon released core assets and need to migrate to newer versions, and those that only affect clients who build their applications in parallel upon the current stage of development in order to release their application upon the latest version. Again, the focus is on behavioral changes.

For changes between releases, the situation is as follows. The decoupled life cycles and the higher heterogeneity among extended core clients require a higher degree of stability, longer periods to undertake the adjustments and a more predictable change process compared to core clients. However, the keystone cannot always assess the impact of interface changes as, first, the keystone's core competence is not in the domain of extended core clients and, second, clients partially need to deviate from architecture guidelines to achieve their business objectives. Hence, they must support the keystone in testing interfaces according their actual usage.

Breaking changes during development are allowed, but need to be published through a change process attending the development phase, similar to the change process for core clients.

Findings 3.3: The keystone must manage dependencies among core assets (4/5). Allowed dependencies to applications must be defined in advance and adherence must be governed (4/5). Breaking changes of released core assets are not allowed or must be undertaken predictably over long periods (5/5). Extended core clients must support the keystone in testing interfaces according their actual usage (4/5).

4.4 Independent Application Development

All clients are self-contained profit centers. They aim at preserving their independence for being able to optimally achieve their objectives. For that reason, some clients chose to explicitly *decouple* their development from the keystone to fulfill their varying business cases and to reduce the impact of platform adaptations. Furthermore, not all *features and changes* required can be processed by the keystone instantly. Hence, innovations, even if there is reuse potential among clients, sometimes take place on client side. To enable reuse afterwards, those innovations either need to be handed over to the keystone or need to be shared directly with other clients. Below, we discuss the challenge in more detail along each collaboration model.

4.4.1 PLE for ISECOs

Decoupling. For both ISECOs, the keystone and core clients develop a common set of products that are jointly distributed. Thereby, synergies shall generate added value for customers. Thus, a decoupling of the client applications is only partially desired. Nevertheless, clients partially need to decouple in order to optimally achieve their objectives and to reduce the impact of platform changes.

Feature and change requests. A crucial challenge for both ISECOs are the limited development capacities of the keystone. As a bottleneck the keystone may slow down the innovation potential of the ecosystem. Multiple clients require plenty of platform adaptation and features. In addition, the generic and reusable implementation within the platform is normally time-consuming. Nevertheless, commonalities shall not be realized many-times on client-side. In fact, the keystone can frequently not satisfy the great demand instantly, resulting in long waiting times. Consequently, in some cases, clients of both ISECOs tend to develop required innovations with platform impact by themselves, in particular, for example, influential clients like DONNIE. Further clients may then want to reuse those developments afterwards. To enable reuse, exchange mechanisms are required so that either the keystone can incorporate the innovations into the platform, or that clients can share them directly among each other. This is a challenge all interviewees were concerned with.

For example, DONNIE handed several components over to the keystone. The architecture challenge was twofold. First, the transfer implicated loss of control and effort for DONNIE, as the components were interconnected with his application architecture. Second, the keystone needed to incorporate them into the platform without breaking applications build upon. This has been quite challenging as there has not been immediate benefit for all other clients, who still work with an old variant of the components. Due to that, the keystone still needs to maintain both variants. However, to reduce maintenance effort, the old components will be discontinued in the future. Thus, clients will either need to adapt their code or take over the maintenance for the old components. This examples illustrates how single clients can significantly influence the ecosystem.

Findings 4.1: The limited development capacities of the keystone limit innovation capabilities (11/11). Clients develop innovations with platform impact by themselves (9/11). To enable reuse, either the keystone must incorporate them into the platform (9/11) or the clients must share them directly among each other (5/11).

4.4.2 Platform Reuse for ISECOs

Decoupling. As already mentioned, platform adaptations are generally performed when required by the core business. Furthermore, the change process is not aligned with the needs of consumer clients. Instead, they have to take the initiative to get informed about changes and to estimate their impact. The high effort that results from changes, especially for those that were not announced in due time, is a problem all consumer clients are concerned with. It is one of the most crucial reasons for TRACY to become a core client. As a consequence, consumer clients must decouple their applications to a large extent to reduce the impact of changes, even if at the expense of other qualities, for example, lower performance.

Whereas SUSAN and TRACY decoupled their applications right from the beginning, STEPHEN initially built his application directly upon the platform. The changing interfaces of the evolving platform resulted in adaptation effort and the final decision to refactor the application to be less dependent on the platform.

Feature and change requests. Due to the comparatively low refunding of the deliveries and, for ISECO-A, large organizational distance, additional features and changes generally need to be requested officially, in some cases with top management support. They are of lower priority and processed dependent on the current workload for the core business and the urgency of the request. This often results in long waiting times or requests that may not be processed at all and, thus, finally require workarounds by consumer clients.

Findings 4.2: Consumer clients must decouple applications to reduce the impact of platform changes (4/5). The low priority for feature and change requests often cause long waiting times or require workarounds by consumer clients (5/5).

4.4.3 Decoupled PLE for ISECOs

Decoupling. Each extended core client has a different business case and partially needs to make design decisions different from the keystone's recommendations. For example, PINO wants to execute core assets in dedicated processes, without using the container the keystone provides. The question that needs to be answered is the following: Which degree of independence do the extended core clients require for fulfilling their business cases and how many error sources are provoked through this additional flexibility? This is a controversial issue all parties of this collaboration model discussed. The fact is, independence is an important characteristic that the keystone must enable if the clients' business drivers require. However, this also implies the need for additional architecture G&G to hinder architectural erosion that the additional flexibility enables.

Feature and change requests. As discussed for core clients, the limited development capacities by the keystone are also a crucial challenge for this collaboration model. In addition, the keystone's core competence is not in the application domain of extended core clients. So even if several clients require the same feature, the keystone might not be able to provide it. Therefore, similar to *PLE for ISECOs*, there is the need to develop those features on the client side, ideally even collaborative across client organizational units.

Findings 4.3: The varying business cases of extended core clients require additional flexibility, (5/5) which provokes error sources (4/5) calling for additional architecture G&G measures (4/5). Similar to *PLE for ISECOs*, they partially need to develop commonalities by themselves (5/5), in some cases aspiring joint development across clients (5/5).

4.5 Qualities

Products that are developed within the ecosystem consist of developments of several organizational units. The compliance with quality requirements generally need to be managed across all involved participants. The most prominent qualities are the developers' habitability and maintainability with regard to the *internal software quality* and reliability, time behavior and memory utilization with regard to the *external software quality*. Below, we discuss the recurring issues of this challenge along each collaboration model in more detail.

4.5.1 PLE for ISECOs

Internal software quality. The varying needs of the numerous clients require a very generic, highly configurable and extendable platform, often by means of metadata or internal domain-specific languages (DSLs) for which there exists only limited tool support. For both ISECOs, this led to a rather uncomfortable and error-prone development context for metadata and DSLs. Error sources are often not obvious and can only by identified by means of time-consuming debugging. In addition, the resulting complexity extends the settling-in periods for new developers. Most interviewees argued that there is the need to establish a more habitable development environment that supports developers to understand the construction and intentions of the system in this regard. Furthermore, clients feel the need to support their efficiency by providing more developer guidance and governance with early feedback. However, respective issues are often specific to the development context and the creation and maintenance of counteractive measures, such as tool support, is generally costly.

The interviewed architects considered it essential to strive for consistent internal quality and to balance it with feature development accordingly. However, external quality issues with direct customer and product management visibility tend to gain more attention. To achieve transparency, there is the need for a quality model that correlates respective measures with the relevance for the product objectives of all organizational units.

External software quality. Core clients develop their applications upon one common platform instance and share available resources. Hence, there is the need to ensure that applications do not impair each other in order to preserve the reliability of the overall system. Furthermore, for both ISECOs, this situation leads to challenges regarding time behavior and memory utilization, and requires the keystones to manage resource consumption, for example to restrict memory usage. Reliability, time behavior and memory utilization are the qualities the interviewees were most concerned with.

Frequently, those quality issues do not arise until several applications are executed simultaneously. Therefore, they need to be handled jointly by the keystone and multiple clients. For both ISECOs, there is the need to guide developers with quality patterns. Furthermore, there is the need for measures that allow to visualize and analyze causes and effects of quality problems across organizational units. This must be done close to point in time where code is created in order to assist developers with early feedback.

Findings 5.1: ISECOs require special attention on measures that increase the developers' habitability (10/11). There is the need to achieve transparency on the business impact of internal quality issues (9/11). External qualities often must be handled across organizational units (11/11). The most crucial ones are time behavior (11/11), memory utilization (9/11) and reliability (7/11).

4.5.2 Platform Reuse for ISECOs

Internal software quality. With regard to the developers' habitability, this is a similar situation as for core clients. In addition, platform and G&G measures are not explicitly geared to the needs of consumer clients. For instance, whereas some interfaces reveal too much functionality or are not needed at all, some required functionality is missing. This may result in a lack of orientation points for developers. Moreover, the low priority for support and, for ISECO-A, the large organizational distance, complicate the identification of suitable contact persons and the access to architecture G&G. In terms of habitability, the consumer-clients' architects are in charge to align the environment with the needs of their developers.

External software quality. As a result of the different business cases, consumer clients generally have different requirements regarding external qualities. For example, SUSAN needs to process much more data compared to core clients. In fact, the business cases of all consumer clients are at the capacity limit of the platform, leading to considerable challenges regarding time behavior and memory utilization. In order to consider them in further decisions processes, the keystone must be aware about their actual platform usage, their quality requirements and the quality problems they have. Furthermore, consumer clients need to know how the platform should be used to accomplish their business. There is also the need to visualize the problems and to guide and govern developers. However, most important is the common commitment to jointly analyze problems and identify solutions, and own initiative by consumer clients.

Findings 5.2: ISECOs require special attention on measures that increase the developers' habitability (5/5). Consumer clients must cope with increased external-quality problems (5/5) due to their different business cases (5/5).

4.5.3 Decoupled PLE for ISECOs

Internal software quality. The situation is similar as for core clients. In addition, for ISECO-B, extended core clients generally also make use of legacy components that are developed and maintained independently, partially by use of different technologies and tools. Consequently, it is the business of the clients' architects to bring the different worlds together in order to create a more habitable environment for their developers.

External software quality. As the keystone's core competence is not in the application domain of extended core clients, they must achieve transparency for their quality requirements, and they need to be involved in concept development and architecture decision making. Thereby, the identification of tradeoffs to satisfy the differing needs turned out as particularly challenging. Furthermore, the keystone is not able to test core assets extensively with regard to external qualities. Instead, as with the challenge *Independent Platform Development*, it is the clients' responsibility to support the keystone in testing and to provide feedback. As for core clients, there is the need to visualize arising defects along with the responsible organizational units and to guide and govern developers.

Findings 5.3: ISECOs require special attention on measures that increase the developers' habitability (4/5). With regard to external qualities, extended core clients must support the keystone in architecture decision making (5/5) and testing (4/5).

4.6 Compliant Software Development

Compliant software development is a cross-cutting issue that relates to all challenges discussed so far. It targets regulations that must be handled across the ecosystem and the establishment and execution of architecture G&G to assist and check for compliance. This requires to *decide on topics* that are most relevant for the ecosystem, to *decide on processes and measures* to execute G&G and to *define roles and responsibilities* across the ecosystem that support the overall approach. In doing so, the keystone acts as driving force behind but does not have power to direct. Below, these topics are addressed in detail for each collaboration model.

4.6.1 PLE for ISECOs

Decide on topics. The main topics are covered by all challenges discussed so far and target intended platform reuse and those issues that need to be regulated across organizational units, for instance architecture dependencies, interface changes or coding styles. An important step towards establishing compliance is to have the organization enabled for exercising compliance. For that reason, costs and benefits need to be transparent to all organizational units upfront and agreed. What matters is to build a consistent understanding and consensus on topics and respective regulations. Different expectations may lead to effort on all sides. For instance, for ISECO-B, there initially was no consistent understanding on which parts of the platform serve as sample application only and are not intended for direct reuse. This led to undesired dependencies which, today, increase maintenance effort.

Decide on processes and measures. In order to enable compliant software development, there is the need to communicate architectural intentions and imposed regulations. The standard that is required regarding the management of technical knowledge is expectably high in such a decentralized context. Relevant information need to be tailored and communicated coherently with regard to the varying development use cases of all organizational units.

Furthermore, it is necessary to establish a process that supports architecture G&G activities as their execution requires an effective collaboration across organizational units. In addition, there is the need to provide tools that govern developers for compliance, foster their mindset through early feedback and motivate them through defined goals and continuous improvement. There is also the need to provide tools to support architects to systematically and continuously monitor compliance. To reduce effort, the compliance process should be automated where possible. However, manual checks like reviews are unavoidable as expert knowledge is often required. As the establishment of processes and tools is time-consuming an active contribution by the clients is required.

As this is a consensus-based environment strict adherence to all defined guidelines cannot be enforced across the ecosystem. It is the self-responsibility of each organizational unit to maintain compliance. Hence, it is highly necessary to establish an organizational mindset towards compliance. Unfortunately, feature and schedule pressure within both ISECOs regularly result in violations by several organizational units, requiring countermeasures to avoid architecture erosion. Moreover, regulations, processes and measures were not fully defined at the beginning, but evolved progressively over time. Consequently, there is also the need to allow for pragmatism if required, but violations must be managed explicitly in order to keep them in mind and to remedy them incrementally later on.

Define roles and responsibilities. Besides establishing a set of regulations and mechanisms to be checked at various phases of the development processes, it's equally important to agree on and establish roles and responsibilities across organizational units that support the compliance process. For both ISECOs, initially not specified contact persons and responsibilities led to loss of development efficiencies and technical debt as required guidance could often not be requested and governance was not executed continuously.

Findings 6.1: Ensuring compliance requires the establishment of architecture G&G (11/11). This includes transparency on costs and benefits (9/11), consensus on regulations (10/11), tailored guidance (8/11), supporting processes and tools (10/11), early feedback (10/11), defined contact persons (8/11) and pragmatism (11/11) along with explicit violation management (8/11).

4.6.2 Platform Reuse for ISECOs

Decide on topics. The platform is not explicitly designed for needs of consumer clients. Consequently, as discussed for the challenges above, they require guidance that is tailored to their specific development use cases to reuse the platform appropriately. What matters most is not a set of commonly agreed regulations, but the communication of general architecture concepts. Especially the transfer of tacit knowledge proved difficult. Often, there are different ways to achieve the same goal. Consumer clients must be aware about intentions behind the concepts to determine those that are suitable for their purposes and to apply them correctly. It is a learning process that requires both, continuous feedback by consumer clients to make the keystone understand their needs, and continuous feedback by the keystone to ensure proper platform usage with regard to their development use cases. As consumer clients are of lower priority, this process necessitates a considerable degree of own initiative on the side of consumer clients. This was an essential differentiator among SUSAN and STEPHEN. Whereas SUSAN were committed to seek for and give feedback right from the beginning STEPHEN initially tended to develop his application widely independent.

Decide on processes and measures. It is the consumer-clients' responsibility to establish processes and measures in order to maintain compliance to the specifically-tailored guidance offered by the keystone. In addition, they require access to the architecture G&G measures that are offered to core clients and need to align them with the needs of their developers. For all consumer clients, late provision of this G&G led to temporary deficits in platform usage.

Define roles and responsibilities. There is the need to define roles that are responsible to filter, prioritize and mediate requests. This is essential to enable effective collaboration within this more decoupled context. For instance, initially missing contact persons led to development inefficiencies for STEPHEN, and to increased expenditure for the keystone to process the number of requests that were submitted directly over bilateral relations among developers.

Findings 6.2: Ensuring appropriate platform reuse by consumer clients requires their continuous commitment to communicate their development use cases (5/5), specifically-tailored guidance and feedback by the keystone (4/5), access to the architecture G&G of core clients (3/5) and roles who mediate requests (4/5).

4.6.3 Decoupled PLE for ISECOs

As this situation is similar to collaboration model *PLE for ISECOs* we dwell on differences only: Extended core clients have different development landscapes and define, build, test and distribute their products autonomously. Within this decoupled context it will be hardly possible to track compliance across the whole ecosystem. If organizational units do not adhere to regulations it will not be possible to guarantee that clients will be able to follow the evolution of future core-asset releases with reasonable effort. Consequently, this situation requires a more decentralized compliance process and significantly more strict adherence to architecture guidelines on client side. Even so, they have varying business cases that partially require to deviate. In terms of sustainability, there is the strong need to agree upfront on regulations and consequences for not adhering.

Findings 6.3: Ensuring compliance has the same enabler as discussed for core clients (5/5), but requires a more decentralized compliance process (4/5) and significantly more strict adherence to architecture guidelines on client side (5/5).

5. DISCUSSION

Limitations and generality. In the following, we discuss construct, internal, conclusion, and external validity threats.

Threats to *construct validity* concern the relation between theory and observation. We mainly performed interviews that rely on the participants' statements, which might be subjective. To limit this effect, we based our findings exclusively on statements that are confirmed by multiple interviews. Furthermore, interview questions might be interpreted differently by researchers and interviewees. To address this, we reflected the notion of decentralized software engineering to their projects and discussed collaboration at the beginning. Regarding completeness, we closed each session with open discussions to check if there are challenges we were not aware of.

Threats to *internal validity* concern co-factors that could influence our results. In our case, interviewees might have given answers that do not fully reflect reality as they were recorded. To address this, we guaranteed anonymity and assured that we will seek for feedback on conclusions to avoid misunderstandings. In addition, results might be biased as we only interviewed architects but no other roles. Due to the number of organizational units this was necessary to keep the effort manageable. However, all architects were well experienced, in central positions and worked closely together with product managers and developers. We asked them to consider all viewpoints.

Threats to *conclusion validity* concern the relation between treatment and results. As this is a qualitative study, data analysis depends on our interpretation. The main work was performed by the first researcher but results were carefully checked by the two others. Additionally, as described in Section 2.2, we used recommended methods to improve validity, such as triangulation, study protocols, member checking and spending sufficient time with the cases.

Threats to *external validity* concern the generalization of our results. We investigated only two of the largest ISECOs at Siemens. Hence, there is the possibility that some results are specific to them. However, our findings stem from characteristics of the respective collaboration models and do not depend on technologies, programming languages or tools. Each finding is confirmed by data of multiple independent organizational units, and the interviewees' average professional experience was more than two decades. Thus, we believe that most of our findings do also hold for other ISECOs that employ similar collaboration models.

Implications for practitioners. Software reuse has been a long standing ambition of the software industry. ISECOs are a powerful approach to enable intra-organizational reuse across various products developed by multiple self-contained organizational units. However, they require a range of specific architectural measures to enable effective software engineering in such a decentralized environment. Practitioners who follow our collaboration models, either adopting new ISECOs or transforming existing ones, can consider our findings to carefully reason on suitable architectural measures in advance. If they do not, according to our results, it is likely that they will face those challenges later on and need to employ counteractive measures reactively, which generally results in increased efforts.

Implications for researchers. Software engineering aims at providing design principles, methods and tools for improving software development. Research in software engineering is dependent on input from industry to identify crucial real-world problems worth to explore. We have outlined a broad field of real-world challenges faced by practitioners in two large-scale projects that involve various self-contained organizational units. We hope that this also inspires other researchers to address these challenges.

6. RELATED WORK

Rommes et al. [27] discuss architecture, process and organization aspects of their medical imaging product line, which involves a set of independent product groups that are distributed across their organization. Van Ommering et al. [22] coin the term product population for their decentralized software product line. Toft et al. [31] present a community-driven approach that allows to share components across their products without involving a central platform organizational unit. Dinkelacker et al. [10] depict the adoption of open-source software development practices within their organization. There are a couple of further researchers who analyze large-scale intra-organizational development [2, 7, 25, 30].

All of them investigate projects where the development takes place in several self-contained organizational units. Based on experience, they discuss challenges they face and approaches that are applied to counter them. However, they do not study different modes of collaboration nor do they relate resulting architecture challenges to them. We carried out an in-depth case study to this end.

Grinter et al. [13] study four models that are applied to coordinate geographically distributed R&D work. They describe the respective benefits and difficulties as well as general challenges. Based on action research, Bosch et al. [6] present architecture, process and organization challenges as well as success factors of five collaboration models for large global software development.

While their work focuses on product lines, global development and open ecosystems, we focus on intra-organizational projects that involve independent spheres of authority. In this sense, our investigation on ISECOs complements their work.

There are also some studies that investigate the emerging discipline on software ecosystems. For example, Hanssen [14] studies a software product line organization and its transition towards a software ecosystem. Greiler et al. [12] conduct a study of testing practices for the Eclipse ecosystem. They identify the need to involve ecosystem partners in platform testing, which is in line with our findings 3.3 and 5.3. Robbes et al. [26] analyze the impact of API deprecations on developers. These studies do investigate open software ecosystems that are quite different from ISECOs.

7. CONCLUSION

Intra-organizational, yet decentralized software projects that involve multiple self-contained organizational units require suitable architectural measures instead of detailed managerial orders to coordinate development. We conducted an in-depth case study on collaboration and architecture challenges for two of the largest (500 and 950 developers) decentralized software projects within Siemens. We found three collaboration models, ranging from high to low coupling. For each collaboration model, we have identified a range of recurring issues and condensed them to a total of 18 key findings. These allow practitioners who find themselves in one of the collaboration models to carefully reason on suitable architectural measures in advance. In addition, they outline a broad field of real-world challenges that need to be investigated by the research community.

8. ACKNOWLEDGMENTS

We thank all participants of our study, especially the 17 interviewees, for spending their valuable time. The authors would also like to thank the anonymous reviewers for their valuable feedback.

9. REFERENCES

[1] J. Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. ACM Press Series. Addison Wesley Publishing Company Incorporated, 2000.

[2] J. Bosch. Software product lines: Organizational alternatives. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, pages 91–100, Washington, DC, USA, 2001. IEEE Computer Society.

[3] J. Bosch. The challenges of broadening the scope of software product families. *Communications of the ACM*, 49(12):41–44, Dec. 2006.

[4] J. Bosch. From software product lines to software ecosystems. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 111–119, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[5] J. Bosch. Architecture challenges for software ecosystems. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ECSA '10, pages 93–95, New York, NY, USA, 2010. ACM.

[6] J. Bosch and P. Bosch-Sijtsema. Softwares product lines, global development and ecosystems: Collaboration in software engineering. In I. Mistrík, J. Grundy, A. Hoek, and J. Whitehead, editors, *Collaborative Software Engineering*, pages 77–92. Springer Berlin Heidelberg, 2010.

[7] L. G. Bratthall, R. van der Geest, H. Hofmann, E. Jellum, Z. Korendo, R. Martinez, M. Orkisz, C. Zeidler, and J. S. Andersson. Integrating hundred's of products through one architecture: The industrial it architecture. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 604–614, New York, NY, USA, 2002. ACM.

[8] P. Clements, R. Kazman, and M. Klein. *Evaluating software architectures: methods and case studies*. SEI series in software engineering. Addison-Wesley, 2002.

[9] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. The SEI series in software engineering. Addison Wesley Professional, 2002.

[10] J. Dinkelacker, P. K. Garg, R. Miller, and D. Nelson. Progressive open source. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 177–184, New York, NY, USA, 2002. ACM.

[11] K. M. Eisenhardt. Building theories from case study research. *Academy of Management Review*, 14(4):532–550, 1989.

[12] M. Greiler, A. v. Deursen, and M.-A. Storey. Test confessions: A study of testing practices for plug-in systems. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 244–254, Piscataway, NJ, USA, 2012. IEEE Press.

[13] R. E. Grinter, J. D. Herbsleb, and D. E. Perry. The geography of coordination: Dealing with distance in R&D work. In *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work*, GROUP '99, pages 306–315, New York, NY, USA, 1999. ACM.

[14] G. K. Hanssen. A longitudinal case study of an emerging software ecosystem: Implications for practice and theory. *Journal of Systems and Software*, 85(7):1455–1466, July 2012.

[15] R. Harrison and T. O. Group. *TOGAF Version 8.1.1 Enterprise Edition*. Togaf Series. Van Haren Publishing, 2007.

[16] S. Jansen, M. Cusumano, and S. Brinkkemper. *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry*. Edward Elgar Publishing, Incorporated, 2013.

[17] R. Kazman, M. Gagliardi, and W. Wood. Scaling up software architecture analysis. *Journal of Systems and Software*, 85(7):1511–1519, July 2012.

[18] K. Manikas and K. M. Hansen. Software ecosystems - a systematic literature review. *Journal of Systems and Software*, 86(5):1294–1306, May 2013.

[19] P. Y. Martin. Grounded Theory and Organizational Research. *The Journal of Applied Behavioral Science*, 22(2):141–157, Apr. 1986.

[20] J. D. McGregor. Ecosystems, continued. *Journal of Object Technology*, 8(7):7–23, Nov. 2009. (column).

[21] D. G. Messerschmitt and C. Szyperski. *Software Ecosystem: Understanding an Indispensable Technology and Industry*. The MIT Press, July 2005.

[22] R. C. v. Ommering and J. Bosch. Widening the scope of software product lines - from variation to composition. In *Proceedings of the Second International Conference on Software Product Lines*, SPLC 2, pages 328–347, London, UK, UK, 2002. Springer-Verlag.

[23] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.

[24] K. Popp and R. Meyer. *Profit from Software Ecosystems: Business Models, Ecosystems and Partnerships in the Software Industry*. Books on Demand, 2010.

[25] D. Riehle, J. Ellenberger, T. Menahem, B. Mikhailovski, Y. Natchetoi, B. Naveh, and T. Odenwald. Open collaboration within corporations using software forges. *Software, IEEE*, 26(2):52–58, March 2009.

[26] R. Robbes, M. Lungu, and D. Röthlisberger. How do developers react to api deprecation?: The case of a smalltalk ecosystem. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 56:1–56:11, New York, NY, USA, 2012. ACM.

[27] E. Rommes and J. G. Wijnstra. Implementing a reuse strategy: Architecture, process and organization aspects of a medical imaging product family. In *Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences - Volume 09*, HICSS '05, pages 312.1–, Washington, DC, USA, 2005. IEEE Computer Society.

[28] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, Apr. 2009.

[29] C. B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, 25(4):557–572, July 1999.

[30] J. Sherrill, J. Averett, and G. Humphrey. Implementing a product line-based architecture in ada. In *Proceedings of the 2001 Annual ACM SIGAda International Conference on Ada*, SIGAda '01, pages 39–46, New York, NY, USA, 2001. ACM.

[31] P. Toft, D. Coleman, and J. Ohta. A cooperative model for cross-divisional product development for a software product line. In *Proceedings of the First Conference on Software Product Lines: Experience and Research Directions*, pages 111–132, Norwell, MA, USA, 2000. Kluwer Academic Publishers.

[32] F. van der Linden, K. Schmid, and E. Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer, 2007.